```
**********************************************************
    1380 Tue Jan 14 16:48:28 2014
new/usr/src/cmd/dtrace/Makefile.com
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
#

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

include ../../Makefile.cmd
include ../../Makefile.ctf
#endif /* ! codereview */

CFLAGS += $(CCVERBOSE)
CFLAGS64 += $(CCVERBOSE)
LDLIBS += -ldtrace -lproc -lctf -lelf

FILEMODE = 0555

CLEANFILES += $(OBJS)

.KEEP_STATE:

all: $(PROG)

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

clean:
        -$(RM) $(CLEANFILES)

lint: lint_SRCS

%.o: ../%.c
```



```
        $(COMPILE.c) $<
        $(POST_PROCESS_O)
#endif /* ! codereview */

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

```
**********************************************************
   46449 Tue Jan 14 16:48:28 2014
new/usr/src/cmd/dtrace/dtrace.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
#endif /* ! codereview */
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>

#include <dtrace.h>
#include <stdlib.h>
#include <stdarg.h>
#include <stdio.h>
#include <strings.h>
#include <unistd.h>
#include <limits.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <alloca.h>
#include <libgen.h>
#include <libproc.h>

typedef struct dtrace_cmd {
        void (*dc_func)(struct dtrace_cmd *);   /* function to compile arg */
        dtrace_probespec_t dc_spec;             /* probe specifier context */
        char *dc_arg;                           /* argument from main argv */
        const char *dc_name;                    /* name for error messages */
        const char *dc_desc;                    /* desc for error messages */
        dtrace_prog_t *dc_prog;                 /* program compiled from arg */
```



```
        char dc_ofile[PATH_MAX];                /* derived output file name */
} dtrace_cmd_t;

#define DMODE_VERS      0       /* display version information and exit (-V) */
#define DMODE_EXEC      1       /* compile program for enabling (-a/e/E) */
#define DMODE_ANON      2       /* compile program for anonymous tracing (-A) */
#define DMODE_LINK      3       /* compile program for linking with ELF (-G) */
#define DMODE_LIST      4       /* compile program and list probes (-l) */
#define DMODE_HEADER    5       /* compile program for headergen (-h) */

#define E_SUCCESS       0
#define E_ERROR         1
#define E_USAGE         2

static const char DTRACE_OPTSTR[] =
        "3:6:aAb:Bc:CD:ef:FGhHi:I:lL:m:n:o:p:P:qs:SU:vVwx:X:Z";

static char **g_argv;
static int g_argc;
static char **g_objv;
static int g_objc;
static dtrace_cmd_t *g_cmdv;
static int g_cmdc;
static struct ps_prochandle **g_psv;
static int g_psc;
static int g_pslive;
static char *g_pname;
static int g_quiet;
static int g_flowindent;
static int g_intr;
static int g_impatient;
static int g_newline;
static int g_total;
static int g_cflags;
static int g_oflags;
static int g_verbose;
static int g_exec = 1;
static int g_mode = DMODE_EXEC;
static int g_status = E_SUCCESS;
static int g_grabanon = 0;
static const char *g_ofile = NULL;
static FILE *g_ofp = stdout;
static dtrace_hdl_t *g_dtp;
static char *g_etcfile = "/etc/system";
static const char *g_etcbegin = "* vvvv Added by DTrace";
static const char *g_etcend = "* ^^^^ Added by DTrace";

static const char *g_etc[] =  {
"*",
"* The following forceload directives were added by dtrace(1M) to allow for",
"* tracing during boot.  If these directives are removed, the system will",
"* continue to function, but tracing will not occur during boot as desired.",
"* To remove these directives (and this block comment) automatically, run",
"* \"dtrace -A\" without additional arguments.  See the \"Anonymous Tracing\"",
"* chapter of the Solaris Dynamic Tracing Guide for details.",
"*",
NULL };

static int
usage(FILE *fp)
{
        static const char predact[] = "[[ predicate ] action ]";

        (void) fprintf(fp, "Usage: %s [-32|-64] [-aACeFGhHlqSvVwZ] "
            "[-b bufsz] [-c cmd] [-D name[=def]]\n\t[-I path] [-L path] "
            "[-o output] [-p pid] [-s script] [-U name]\n\t"
```

```
123             "[-x opt[=val]] [-X a|c|s|t]\n\n"
124             "\t[-P provider %s]\n"
125             "\t[-m [ provider: ] module %s]\n"
126             "\t[-f [[ provider: ] module: ] func %s]\n"
127             "\t[-n [[[ provider: ] module: ] func: ] name %s]\n"
128             "\t[-i probe-id %s] [ args ... ]\n\n", g_pname,
129             predact, predact, predact, predact, predact);
130
131         (void) fprintf(fp, "\tpredicate -> '/' D-expression '/'\n");
132         (void) fprintf(fp, "\t   action -> '{' D-statements '}'\n");
133
134         (void) fprintf(fp, "\n"
135             "\t-32 generate 32-bit D programs and ELF files\n"
136             "\t-64 generate 64-bit D programs and ELF files\n\n"
137             "\t-a  claim anonymous tracing state\n"
138             "\t-A  generate driver.conf(4) directives for anonymous tracing\n"
139             "\t-b  set trace buffer size\n"
140             "\t-c  run specified command and exit upon its completion\n"
141             "\t-C  run cpp(1) preprocessor on script files\n"
142             "\t-D  define symbol when invoking preprocessor\n"
143             "\t-e  exit after compiling request but prior to enabling probes\n"
144             "\t-f  enable or list probes matching the specified function name\n"
145             "\t-F  coalesce trace output by function\n"
146             "\t-G  generate an ELF file containing embedded dtrace program\n"
147             "\t-h  generate a header file with definitions for static probes\n"
148             "\t-H  print included files when invoking preprocessor\n"
149             "\t-i  enable or list probes matching the specified probe id\n"
150             "\t-I  add include directory to preprocessor search path\n"
151             "\t-l  list probes matching specified criteria\n"
152             "\t-L  add library directory to library search path\n"
153             "\t-m  enable or list probes matching the specified module name\n"
154             "\t-n  enable or list probes matching the specified probe name\n"
155             "\t-o  set output file\n"
156             "\t-p  grab specified process-ID and cache its symbol tables\n"
157             "\t-P  enable or list probes matching the specified provider name\n"
158             "\t-q  set quiet mode (only output explicitly traced data)\n"
159             "\t-s  enable or list probes according to the specified D script\n"
160             "\t-S  print D compiler intermediate code\n"
161             "\t-U  undefine symbol when invoking preprocessor\n"
162             "\t-v  set verbose mode (report stability attributes, arguments)\n"
163             "\t-V  report DTrace API version\n"
164             "\t-w  permit destructive actions\n"
165             "\t-x  enable or modify compiler and tracing options\n"
166             "\t-X  specify ISO C conformance settings for preprocessor\n"
167             "\t-Z  permit probe descriptions that match zero probes\n");
168
169         return (E_USAGE);
170 }
171
172 static void
173 verror(const char *fmt, va_list ap)
174 {
175         int error = errno;
176
177         (void) fprintf(stderr, "%s: ", g_pname);
178         (void) vfprintf(stderr, fmt, ap);
179
180         if (fmt[strlen(fmt) - 1] != '\n')
181                 (void) fprintf(stderr, ": %s\n", strerror(error));
182 }
183
184 /*PRINTFLIKE1*/
185 static void
186 fatal(const char *fmt, ...)
187 {
188         va_list ap;
```



```
189
190         va_start(ap, fmt);
191         verror(fmt, ap);
192         va_end(ap);
193
194         exit(E_ERROR);
195 }
196
197 /*PRINTFLIKE1*/
198 static void
199 dfatal(const char *fmt, ...)
200 {
201         va_list ap;
202
203         va_start(ap, fmt);
204
205         (void) fprintf(stderr, "%s: ", g_pname);
206         if (fmt != NULL)
207                 (void) vfprintf(stderr, fmt, ap);
208
209         va_end(ap);
210
211         if (fmt != NULL && fmt[strlen(fmt) - 1] != '\n') {
212                 (void) fprintf(stderr, ": %s\n",
213                     dtrace_errmsg(g_dtp, dtrace_errno(g_dtp)));
214         } else if (fmt == NULL) {
215                 (void) fprintf(stderr, "%s\n",
216                     dtrace_errmsg(g_dtp, dtrace_errno(g_dtp)));
217         }
218
219         /*
220          * Close the DTrace handle to ensure that any controlled processes are
221          * correctly restored and continued.
222          */
223         dtrace_close(g_dtp);
224
225         exit(E_ERROR);
226 }
227
228 /*PRINTFLIKE1*/
229 static void
230 error(const char *fmt, ...)
231 {
232         va_list ap;
233
234         va_start(ap, fmt);
235         verror(fmt, ap);
236         va_end(ap);
237 }
238
239 /*PRINTFLIKE1*/
240 static void
241 notice(const char *fmt, ...)
242 {
243         va_list ap;
244
245         if (g_quiet)
246                 return; /* -q or quiet pragma suppresses notice()s */
247
248         va_start(ap, fmt);
249         verror(fmt, ap);
250         va_end(ap);
251 }
252
253 /*PRINTFLIKE1*/
254 static void
```

```
255 oprintf(const char *fmt, ...)
256 {
257         va_list ap;
258         int n;
259
260         if (g_ofp == NULL)
261                 return;
262
263         va_start(ap, fmt);
264         n = vfprintf(g_ofp, fmt, ap);
265         va_end(ap);
266
267         if (n < 0) {
268                 if (errno != EINTR) {
269                         fatal("failed to write to %s",
270                             g_ofile ? g_ofile : "<stdout>");
271                 }
272                 clearerr(g_ofp);
273         }
274 }
275
276 static char **
277 make_argv(char *s)
278 {
279         const char *ws = "\f\n\r\t\v ";
280         char **argv = malloc(sizeof (char *) * (strlen(s) / 2 + 1));
281         int argc = 0;
282         char *p = s;
283
284         if (argv == NULL)
285                 return (NULL);
286
287         for (p = strtok(s, ws); p != NULL; p = strtok(NULL, ws))
288                 argv[argc++] = p;
289
290         if (argc == 0)
291                 argv[argc++] = s;
292
293         argv[argc] = NULL;
294         return (argv);
295 }
296
297 static void
298 dof_prune(const char *fname)
299 {
300         struct stat sbuf;
301         size_t sz, i, j, mark, len;
302         char *buf;
303         int msg = 0, fd;
304
305         if ((fd = open(fname, O_RDONLY)) == -1) {
306                 /*
307                  * This is okay only if the file doesn't exist at all.
308                  */
309                 if (errno != ENOENT)
310                         fatal("failed to open %s", fname);
311                 return;
312         }
313
314         if (fstat(fd, &sbuf) == -1)
315                 fatal("failed to fstat %s", fname);
316
317         if ((buf = malloc((sz = sbuf.st_size) + 1)) == NULL)
318                 fatal("failed to allocate memory for %s", fname);
319
320         if (read(fd, buf, sz) != sz)
```



```
321                 fatal("failed to read %s", fname);
322
323         buf[sz] = '\0';
324         (void) close(fd);
325
326         if ((fd = open(fname, O_WRONLY | O_TRUNC)) == -1)
327                 fatal("failed to open %s for writing", fname);
328
329         len = strlen("dof-data-");
330
331         for (mark = 0, i = 0; i < sz; i++) {
332                 if (strncmp(&buf[i], "dof-data-", len) != 0)
333                         continue;
334
335                 /*
336                  * This is only a match if it's in the 0th column.
337                  */
338                 if (i != 0 && buf[i - 1] != '\n')
339                         continue;
340
341                 if (msg++ == 0) {
342                         error("cleaned up old anonymous "
343                             "enabling in %s\n", fname);
344                 }
345
346                 /*
347                  * We have a match.  First write out our data up until now.
348                  */
349                 if (i != mark) {
350                         if (write(fd, &buf[mark], i - mark) != i - mark)
351                                 fatal("failed to write to %s", fname);
352                 }
353
354                 /*
355                  * Now scan forward until we scan past a newline.
356                  */
357                 for (j = i; j < sz && buf[j] != '\n'; j++)
358                         continue;
359
360                 /*
361                  * Reset our mark.
362                  */
363                 if ((mark = j + 1) >= sz)
364                         break;
365
366                 i = j;
367         }
368
369         if (mark < sz) {
370                 if (write(fd, &buf[mark], sz - mark) != sz - mark)
371                         fatal("failed to write to %s", fname);
372         }
373
374         (void) close(fd);
375         free(buf);
376 }
377
378 static void
379 etcsystem_prune(void)
380 {
381         struct stat sbuf;
382         size_t sz;
383         char *buf, *start, *end;
384         int fd;
385         char *fname = g_etcfile, *tmpname;
```

```
387         if ((fd = open(fname, O_RDONLY)) == -1)
388                 fatal("failed to open %s", fname);
389
390         if (fstat(fd, &sbuf) == -1)
391                 fatal("failed to fstat %s", fname);
392
393         if ((buf = malloc((sz = sbuf.st_size) + 1)) == NULL)
394                 fatal("failed to allocate memory for %s", fname);
395
396         if (read(fd, buf, sz) != sz)
397                 fatal("failed to read %s", fname);
398
399         buf[sz] = '\0';
400         (void) close(fd);
401
402         if ((start = strstr(buf, g_etcbegin)) == NULL)
403                 goto out;
404
405         if (strlen(buf) != sz) {
406                 fatal("embedded nul byte in %s; manual repair of %s "
407                     "required\n", fname, fname);
408         }
409
410         if (strstr(start + 1, g_etcbegin) != NULL) {
411                 fatal("multiple start sentinels in %s; manual repair of %s "
412                     "required\n", fname, fname);
413         }
414
415         if ((end = strstr(buf, g_etcend)) == NULL) {
416                 fatal("missing end sentinel in %s; manual repair of %s "
417                     "required\n", fname, fname);
418         }
419
420         if (start > end) {
421                 fatal("end sentinel preceeds start sentinel in %s; manual "
422                     "repair of %s required\n", fname, fname);
423         }
424
425         end += strlen(g_etcend) + 1;
426         bcopy(end, start, strlen(end) + 1);
427
428         tmpname = alloca(sz = strlen(fname) + 80);
429         (void) snprintf(tmpname, sz, "%s.dtrace.%d", fname, getpid());
430
431         if ((fd = open(tmpname,
432             O_WRONLY | O_CREAT | O_EXCL, sbuf.st_mode)) == -1)
433                 fatal("failed to create %s", tmpname);
434
435         if (write(fd, buf, strlen(buf)) < strlen(buf)) {
436                 (void) unlink(tmpname);
437                 fatal("failed to write to %s", tmpname);
438         }
439
440         (void) close(fd);
441
442         if (chown(tmpname, sbuf.st_uid, sbuf.st_gid) != 0) {
443                 (void) unlink(tmpname);
444                 fatal("failed to chown(2) %s to uid %d, gid %d", tmpname,
445                     (int)sbuf.st_uid, (int)sbuf.st_gid);
446         }
447
448         if (rename(tmpname, fname) == -1)
449                 fatal("rename of %s to %s failed", tmpname, fname);
450
451         error("cleaned up forceload directives in %s\n", fname);
452 out:
```



```
453         free(buf);
454 }
455
456 static void
457 etcsystem_add(void)
458 {
459         const char *mods[20];
460         int nmods, line;
461
462         if ((g_ofp = fopen(g_ofile = g_etcfile, "a")) == NULL)
463                 fatal("failed to open output file '%s'", g_ofile);
464
465         oprintf("%s\n", g_etcbegin);
466
467         for (line = 0; g_etc[line] != NULL; line++)
468                 oprintf("%s\n", g_etc[line]);
469
470         nmods = dtrace_provider_modules(g_dtp, mods,
471             sizeof (mods) / sizeof (char *) - 1);
472
473         if (nmods >= sizeof (mods) / sizeof (char *))
474                 fatal("unexpectedly large number of modules!");
475
476         mods[nmods++] = "dtrace";
477
478         for (line = 0; line < nmods; line++)
479                 oprintf("forceload: drv/%s\n", mods[line]);
480
481         oprintf("%s\n", g_etcend);
482
483         if (fclose(g_ofp) == EOF)
484                 fatal("failed to close output file '%s'", g_ofile);
485
486         error("added forceload directives to %s\n", g_ofile);
487 }
488
489 static void
490 print_probe_info(const dtrace_probeinfo_t *p)
491 {
492         char buf[BUFSIZ];
493         char *user;
494 #endif /* ! codereview */
495         int i;
496
497         oprintf("\n\tProbe Description Attributes\n");
498
499         oprintf("\t\tIdentifier Names: %s\n",
500             dtrace_stability_name(p->dtp_attr.dtat_name));
501         oprintf("\t\tData Semantics:   %s\n",
502             dtrace_stability_name(p->dtp_attr.dtat_data));
503         oprintf("\t\tDependency Class: %s\n",
504             dtrace_class_name(p->dtp_attr.dtat_class));
505
506         oprintf("\n\tArgument Attributes\n");
507
508         oprintf("\t\tIdentifier Names: %s\n",
509             dtrace_stability_name(p->dtp_arga.dtat_name));
510         oprintf("\t\tData Semantics:   %s\n",
511             dtrace_stability_name(p->dtp_arga.dtat_data));
512         oprintf("\t\tDependency Class: %s\n",
513             dtrace_class_name(p->dtp_arga.dtat_class));
514
515         oprintf("\n\tArgument Types\n");
516
517         for (i = 0; i < p->dtp_argc; i++) {
518                 if (p->dtp_argv[i].dtt_flags & DTT_FL_USER)
```

```
519                         user = "userland ";
520                 else
521                         user = "";
522 #endif /* ! codereview */
523                 if (ctf_type_name(p->dtp_argv[i].dtt_ctfp,
524                     p->dtp_argv[i].dtt_type, buf, sizeof (buf)) == NULL)
525                         (void) strlcpy(buf, "(unknown)", sizeof (buf));
526                 oprintf("\t\targs[%d]: %s%s\n", i, user, buf);
 28                 oprintf("\t\targs[%d]: %s\n", i, buf);
527         }

529         if (p->dtp_argc == 0)
530                 oprintf("\t\tNone\n");

532         oprintf("\n");
533 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    5512 Tue Jan 14 16:48:29 2014
new/usr/src/cmd/dtrace/test/tst/common/Makefile
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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) 2012 by Delphix. All rights reserved.
# Copyright (c) 2013, Joyent, Inc. All rights reserved.
#

#
# WARNING: Do not include Makefile.ctf here. That will cause tests to
# break.
# Copyright (c) 2012, Joyent, Inc. All rights reserved.
#

include $(SRC)/Makefile.master
include ../Makefile.com

SNOOPDIR = $(SRC)/cmd/cmd-inet/usr.sbin/snoop
SNOOPOBJS = nfs4_xdr.o
SNOOPSRCS = ${SNOOPOBJS:%.o=%.c}
CLOBBERFILES += nfs/$(SNOOPOBJS)

RPCSVCDIR = $(SRC)/head/rpcsvc
RPCSVCOBJS = nfs_prot.o
RPCSVCSRCS = ${RPCSVCOBJS:%o=%c}
CLOBBERFILES += nfs/$(RPCSVCOBJS) $(RPCSVCDIR)/$(RPCSVCSRCS)
CLOBBERFILES += usdt/forker.h usdt/lazyprobe.h

fasttrap/tst.fasttrap.exe := LDLIBS += -ldtrace
fasttrap/tst.stack.exe := LDLIBS += -ldtrace

sysevent/tst.post.exe := LDLIBS += -lsysevent
sysevent/tst.post_chan.exe := LDLIBS += -lsysevent
```



```
ustack/tst.bigstack.exe := COPTFLAG += -xO1

GCC = $(ONBLD_TOOLS)/bin/$(MACH)/cw -_gcc

nfs/%.o: $(SNOOPDIR)/%.c
        $(COMPILE.c) -o $@ $< -I$(SNOOPDIR)
        $(POST_PROCESS_O)
nfs/tst.call.exe: nfs/tst.call.o nfs/$(SNOOPOBJS)
        $(LINK.c) -o $@ nfs/tst.call.o nfs/$(SNOOPOBJS) $(LDLIBS) -lnsl
        $(POST_PROCESS) ; $(STRIP_STABS)
$(RPCSVCDIR)/%.c: $(RPCSVCDIR)/%.x
        $(RPCGEN) -Cc $< > $@
nfs/$(RPCSVCOBJS): $(RPCSVCDIR)/$(RPCSVCSRCS)
        $(COMPILE.c) -o $@ $(RPCSVCDIR)/$(RPCSVCSRCS)
        $(POST_PROCESS_O)
nfs/tst.call3.exe: nfs/tst.call3.o nfs/$(RPCSVCOBJS)
        $(LINK.c) -o $@ nfs/tst.call3.o nfs/$(RPCSVCOBJS) \
        $(LDLIBS) -lnsl -lrpcsvc
        $(POST_PROCESS) ; $(STRIP_STABS)

pid/tst.gcc.exe: pid/tst.gcc.c
        $(GCC) -o pid/tst.gcc.exe pid/tst.gcc.c $(LDFLAGS)
        $(POST_PROCESS) ; $(STRIP_STABS)

json/tst.usdt.o: json/usdt.h

json/usdt.h: json/usdt.d
        $(DTRACE) -h -s json/usdt.d -o json/usdt.h

json/usdt.o: json/usdt.d json/tst.usdt.o
        $(COMPILE.d) -o json/usdt.o -s json/usdt.d json/tst.usdt.o

json/tst.usdt.exe: json/tst.usdt.o json/usdt.o
        $(LINK.c) -o json/tst.usdt.exe json/tst.usdt.o json/usdt.o $(LDLIBS)
        $(POST_PROCESS) ; $(STRIP_STABS)

#
# Tests that use the next three programs rely on the binaries having
# valid CTF data.
#
uctf/tst.aouttype.exe: uctf/tst.aouttype.c
        $(COMPILE.c) $(CTF_FLAGS) -o uctf/tst.aouttype.o uctf/tst.aouttype.c
        $(CTFCONVERT) -i -L VERSION uctf/tst.aouttype.o
        $(LINK.c) -o uctf/tst.aouttype.exe uctf/tst.aouttype.o $(LDLIBS)
        $(CTFMERGE) -L VERSION -o $@ uctf/tst.aouttype.o
        $(POST_PROCESS) ; $(STRIP_STABS)

uctf/tst.chasestrings.exe: uctf/tst.chasestrings.c
        $(COMPILE.c) $(CTF_FLAGS) -o uctf/tst.chasestrings.o uctf/tst.chasestrin
        $(CTFCONVERT) -i -L VERSION uctf/tst.chasestrings.o
        $(LINK.c) -o uctf/tst.chasestrings.exe uctf/tst.chasestrings.o $(LDLIBS)
        $(CTFMERGE) -L VERSION -o $@ uctf/tst.chasestrings.o
        $(POST_PROCESS) ; $(STRIP_STABS)

uctf/tst.printtype.exe: uctf/tst.printtype.c
        $(COMPILE.c) $(CTF_FLAGS) -o uctf/tst.printtype.o uctf/tst.printtype.c
        $(CTFCONVERT) -i -L VERSION uctf/tst.printtype.o
        $(LINK.c) -o uctf/tst.printtype.exe uctf/tst.printtype.o $(LDLIBS)
        $(CTFMERGE) -L VERSION -o $@ uctf/tst.printtype.o
        $(POST_PROCESS) ; $(STRIP_STABS)

#
# This program should never have any ctf data in it.
#
uctf/tst.libtype.exe:
```

```
122         $(LINK.c) -o uctf/tst.libtype.exe uctf/tst.libtype.c $(LDLIBS)
123         $(POST_PROCESS) ; $(STRIP_STABS)

125 #endif /* ! codereview */
126 usdt/tst.args.exe: usdt/tst.args.o usdt/args.o
127         $(LINK.c) -o usdt/tst.args.exe usdt/tst.args.o usdt/args.o $(LDLIBS)
128         $(POST_PROCESS) ; $(STRIP_STABS)

130 usdt/args.o: usdt/args.d usdt/tst.args.o
131         $(COMPILE.d) -o usdt/args.o -s usdt/args.d usdt/tst.args.o

133 usdt/tst.argmap.exe: usdt/tst.argmap.o usdt/argmap.o
134         $(LINK.c) -o usdt/tst.argmap.exe \
135             usdt/tst.argmap.o usdt/argmap.o $(LDLIBS)
136         $(POST_PROCESS) ; $(STRIP_STABS)

138 usdt/argmap.o: usdt/argmap.d usdt/tst.argmap.o
139         $(COMPILE.d) -o usdt/argmap.o -s usdt/argmap.d usdt/tst.argmap.o

141 usdt/tst.forker.exe: usdt/tst.forker.o usdt/forker.o
142         $(LINK.c) -o usdt/tst.forker.exe \
143             usdt/tst.forker.o usdt/forker.o $(LDLIBS)
144         $(POST_PROCESS) ; $(STRIP_STABS)

146 usdt/forker.o: usdt/forker.d usdt/tst.forker.o
147         $(COMPILE.d) -o usdt/forker.o -s usdt/forker.d usdt/tst.forker.o

149 usdt/tst.forker.o: usdt/forker.h

151 usdt/forker.h: usdt/forker.d
152         $(DTRACE) -h -s usdt/forker.d -o usdt/forker.h

154 usdt/tst.lazyprobe.exe: usdt/tst.lazyprobe.o usdt/lazyprobe.o
155         $(LINK.c) -o usdt/tst.lazyprobe.exe \
156             usdt/tst.lazyprobe.o usdt/lazyprobe.o $(LDLIBS)
157         $(POST_PROCESS) ; $(STRIP_STABS)

159 usdt/lazyprobe.o: usdt/lazyprobe.d usdt/tst.lazyprobe.o
160         $(COMPILE.d) -xlazyload -o usdt/lazyprobe.o \
161             -s usdt/lazyprobe.d usdt/tst.lazyprobe.o

163 usdt/tst.lazyprobe.o: usdt/lazyprobe.h

165 usdt/lazyprobe.h: usdt/lazyprobe.d
166         $(DTRACE) -h -s usdt/lazyprobe.d -o usdt/lazyprobe.h

168 SUBDIRS = java_api
169 include ../../Makefile.subdirs
```

```
**********************************************************
    1662 Tue Jan 14 16:48:29 2014
new/usr/src/cmd/dtrace/test/tst/common/pid/tst.provregex1.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

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

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

  28 #
  29 # This test verifies that specifying a glob in a pid provider name
  30 # (e.g., p*d$target) works.
  31 #

  33 if [ $# != 1 ]; then
  34         echo expected one argument: '<'dtrace-path'>'
  35         exit 2
  36 fi

  38 dtrace=$1
  39 DIR=${TMPDIR:-/tmp}/dtest.$$

  41 mkdir $DIR
  42 cd $DIR

  44 cat > Makefile <<EOF
  45 all: main

  47 main: main.o
  48         gcc -m32 -o main main.o
  48         gcc -o main main.o

  50 main.o: main.c
  51         gcc -m32 -c main.c
  51         gcc -c main.c
  52 EOF

  54 cat > main.c <<EOF
```



```
  55 void
  56 go(void)
  57 {
  58 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2286 Tue Jan 14 16:48:30 2014
new/usr/src/cmd/dtrace/test/tst/common/pid/tst.provregex2.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

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

#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# This test verifies that probes will be picked up after a dlopen(3C)
# when the pid provider is specified as a glob (e.g., p*d$target.)
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
DIR=${TMPDIR:-/tmp}/dtest.$$

mkdir $DIR
cd $DIR

cat > Makefile <<EOF
all: main altlib.so

main: main.o
        gcc -m32 -o main main.o
        gcc -o main main.o

main.o: main.c
        gcc -m32 -c main.c
        gcc -c main.c

altlib.so: altlib.o
        gcc -m32 -shared -o altlib.so altlib.o -lc
```



```
        gcc -shared -o altlib.so altlib.o -lc

altlib.o: altlib.c
        gcc -m32 -fPIC -c altlib.c
        gcc -c altlib.c
EOF

cat > altlib.c <<EOF
void
go(void)
{
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1932 Tue Jan 14 16:48:30 2014
new/usr/src/cmd/dtrace/test/tst/common/pid/tst.provregex3.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

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

#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# This test verifies that a regex in the provider name will match
# USDT probes as well as pid probes (e.g., p*d$target matches both
# pid$target and pyramid$target.)
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
DIR=${TMPDIR:-/tmp}/dtest.$$

mkdir $DIR
cd $DIR

cat > Makefile <<EOF
all: main

main: main.o prov.o
        gcc -m32 -o main main.o prov.o
        gcc -o main main.o prov.o

main.o: main.c prov.h
        gcc -m32 -c main.c
        gcc -c main.c

prov.h: prov.d
```



```
        $dtrace -h -s prov.d

prov.o: prov.d main.o
        $dtrace -G -32 -s prov.d main.o
EOF

cat > prov.d <<EOF
provider pyramid {
        probe entry();
};
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2780 Tue Jan 14 16:48:31 2014
new/usr/src/cmd/dtrace/test/tst/common/pid/tst.provregex4.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

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

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

 28 #
 29 # This test verifies that USDT probes will be picked up after a dlopen(3C)
 30 # when a regex in the provider name matches both USDT probes and pid probes
 31 # (e.g., p*d$target matches both pid$target and pyramid$target.)
 32 #

 34 if [ $# != 1 ]; then
 35         echo expected one argument: '<'dtrace-path'>'
 36         exit 2
 37 fi

 39 dtrace=$1
 40 DIR=${TMPDIR:-/tmp}/dtest.$$

 42 mkdir $DIR
 43 cd $DIR

 45 cat > Makefile <<EOF
 46 all: main altlib.so

 48 main: main.o provmain.o
 49         gcc -m32 -o main main.o provmain.o
 49         gcc -o main main.o provmain.o

 51 main.o: main.c prov.h
 52         gcc -m32 -c main.c
 52         gcc -c main.c

 54 prov.h: prov.d
```



```
 55         $dtrace -h -s prov.d

 57 provmain.o: prov.d main.o
 58         $dtrace -G -32 -o provmain.o -s prov.d main.o

 60 altlib.so: altlib.o provalt.o
 61         gcc -m32 -shared -o altlib.so altlib.o provalt.o -lc
 61         gcc -shared -o altlib.so altlib.o provalt.o -lc

 63 altlib.o: altlib.c prov.h
 64         gcc -m32 -c altlib.c
 64         gcc -c altlib.c

 66 provalt.o: prov.d altlib.o
 67         $dtrace -G -32 -o provalt.o -s prov.d altlib.o
 68 EOF

 70 cat > prov.d <<EOF
 71 provider pyramid {
 72         probe entry();
 73 };
______unchanged_portion_omitted_
```

```
**********************************************************
    2111 Tue Jan 14 16:48:31 2014
new/usr/src/cmd/dtrace/test/tst/common/printa/tst.largeusersym.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
______unchanged_portion_omitted_
  51 EOF

  53 gcc -m32 -o test test.c
  53 gcc -o test test.c
  54 if [ $? -ne 0 ]; then
  55         print -u2 "failed to compile test.c"
  56         exit 1
  57 fi

  59 script()
  60 {
  61         $dtrace -c ./test -qs /dev/stdin <<EOF
  62         profile:::profile-1001hz
  63         /pid == \$target/
  64         {
  65                 @[arg1] = count();
  66         }

  68         tick-1s
  69         /n++ > 10/
  70         {
  71                 printa("%A %@d\n", @);
  72                 exit(0);
  73         }
  74 EOF
  75 }
______unchanged_portion_omitted_
```

```
**********************************************************
     536 Tue Jan 14 16:48:31 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/err.invalidpid.d
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 Joyent, Inc.  All rights reserved.
 */

#pragma D option quiet

BEGIN
{
        trace((pidfoo‘int)0);
}
#endif /* ! codereview */
```

****
537 Tue Jan 14 16:48:32 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/err.invalidpid2.d
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 Joyent, Inc.  All rights reserved.
 */

#pragma D option quiet

BEGIN
{
        trace((pid8foo‘int)0);
}
#endif /* ! codereview */
```

```
**********************************************************
     534 Tue Jan 14 16:48:32 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/err.invalidpid3.d
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 Joyent, Inc.  All rights reserved.
 */

#pragma D option quiet

BEGIN
{
        trace((pid0‘int)0);
}
#endif /* ! codereview */
```

```
**********************************************************
     841 Tue Jan 14 16:48:32 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/err.invalidtype.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# While it's hard to be completely certain that a type of the name we want
# doesn't exist, we're going to try to pick a name which is rather unique.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
t="season_8_mountain_of_madness_t"
pid=$$

rc=`$dtrace -n "BEGIN{ trace(pid$pid`$t)0); }"`

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
     903 Tue Jan 14 16:48:32 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/err.invalidtype2.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
   1 #! /usr/bin/ksh
   2 #
   3 #
   4 # This file and its contents are supplied under the terms of the
   5 # Common Development and Distribution License ("CDDL"), version 1.0.
   6 # You may only use this file in accordance with the terms of version
   7 # 1.0 of the CDDL.
   8 #
   9 # A full copy of the text of the CDDL should have accompanied this
  10 # source.  A copy of the CDDL is also available via the Internet at
  11 # http://www.illumos.org/license/CDDL.
  12 #

  14 #
  15 # Copyright (c) 2013 Joyent, Inc. All rights reserved.
  16 #

  18 #
  19 # While it's hard to be completely certain that a type of the name we want
  20 # doesn't exist, we're going to try to pick a name which is rather
  21 # unique. This time we're also going to use the pid$target alias.
  22 #

  24 if [ $# != 1 ]; then
  25         echo expected one argument: '<'dtrace-path'>'
  26         exit 2
  27 fi

  29 dtrace=$1
  30 t="season_8_mountain_of_madness_t"
  31 pid=$$

  33 rc=`$dtrace -n "BEGIN{ trace(pid`$t)0); }"` -p $pid

  35 exit $rc
  36 #endif /* ! codereview */
```

```
**********************************************************
    2049 Tue Jan 14 16:48:32 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/err.user64mode.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# This test is purposefully using a 64-bit DTrace and thus 64-bit types
# when compared with a 32-bit process. This test uses the userland
# keyword and so the implicit copyin should access illegal memory and
# thus exit.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
t="zelda_info_t"
exe="tst.chasestrings.exe"

elfdump "./$exe" | grep -q '.SUNW_ctf'
if [[ $? -ne 0 ]]; then
        echo "CTF does not exist in $exe, that's a bug" >&2
        exit 1
fi

./$exe &
pid=$!

$dtrace -64 -qs /dev/stdin <<EOF
typedef struct info {
        char    *zi_gamename;
        int     zi_ndungeons;
        char    *zi_villain;
        int     zi_haszelda;
} info_t;

pid$pid::has_princess:entry
/next == 0/
{
        this->t = (userland info_t *)arg0;
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            stringof(this->t->zi_gamename), this->t->zi_ndungeons,
```



```
            stringof(this->t->zi_villain), this->t->zi_haszelda);
        next = 1;
}

pid$pid::has_dungeons:entry
/next == 1/
{
        this->t = (userland info_t *)arg0;
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            stringof(this->t->zi_gamename), this->t->zi_ndungeons,
            stringof(this->t->zi_villain), this->t->zi_haszelda);
        next = 2;
}

pid$pid::has_villain:entry
/next == 2/
{
        this->t = (userland info_t *)arg0;
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            stringof(this->t->zi_gamename), this->t->zi_ndungeons,
            stringof(this->t->zi_villain), this->t->zi_haszelda);
        exit(0);
}

ERROR
{
        exit(1);
}
EOF
rc=$?

kill -9 $pid

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
     990 Tue Jan 14 16:48:33 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.aouttype.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 2013 (c) Joyent, Inc. All rights reserved.
 */

/*
 * This test tries to make sure that we have CTF data for a type that only this
 * binary would reasonably have. In this case, the
 * season_7_lisa_the_vegetarian_t.
 */
#include <unistd.h>

typedef struct season_7_lisa_the_vegetarian {
        int fr_salad;
} season_7_lisa_the_vegetarian_t;

int
sleeper(season_7_lisa_the_vegetarian_t *lp)
{
        for (;;) {
                sleep(lp->fr_salad);
        }
        /*NOTREACHED*/
        return (0);
}

int
main(void)
{
        season_7_lisa_the_vegetarian_t l;
        l.fr_salad = 100;

        sleeper(&l);

        return (0);
}
#endif /* ! codereview */
```

```
**********************************************************
     917 Tue Jan 14 16:48:33 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.aouttype.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# Lookup a type that is inside a.out.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
t="season_7_lisa_the_vegetrian_t *"
exe="tst.aouttype.exe"

elfdump "./$exe" | grep -q '.SUNW_ctf'
if [[ $? -ne 0 ]]; then
        echo "CTF does not exist in $exe, that's a bug" >&2
        exit 1
fi

./$exe &
pid=$!

rc=`$dtrace -n "BEGIN{ trace((pid$pid\`$t)0); exit(0); }"`

kill -9 $pid

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
    1600 Tue Jan 14 16:48:33 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.chasestrings.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 2013 (c) Joyent, Inc. All rights reserved.
 */

/*
 * This test takes data from the current binary which is basically running in a
 * loop between two functions and our goal is to have two unique types that they
 * contain which we can print.
 */

#include <unistd.h>

typedef struct zelda_info {
        char    *zi_gamename;
        int     zi_ndungeons;
        char    *zi_villain;
        int     zi_haszelda;
} zelda_info_t;

static int
has_princess(zelda_info_t *z)
{
        return (z->zi_haszelda);
}

static int
has_dungeons(zelda_info_t *z)
{
        return (z->zi_ndungeons != 0);
}

static const char *
has_villain(zelda_info_t *z)
{
        return (z->zi_villain);
}

int
main(void)
{
        zelda_info_t oot;
        zelda_info_t la;
        zelda_info_t lttp;

        oot.zi_gamename = "Ocarina of Time";
```



```
        oot.zi_ndungeons = 10;
        oot.zi_villain = "Ganondorf";
        oot.zi_haszelda = 1;

        la.zi_gamename = "Link's Awakening";
        la.zi_ndungeons = 9;
        la.zi_villain = "Nightmare";
        la.zi_haszelda = 0;

        lttp.zi_gamename = "A Link to the Past";
        lttp.zi_ndungeons = 12;
        lttp.zi_villain = "Ganon";
        lttp.zi_haszelda = 1;

        for (;;) {
                (void) has_princess(&oot);
                (void) has_dungeons(&la);
                (void) has_villain(&lttp);
                sleep(1);
        }

        return (0);
}
#endif /* ! codereview */
```

```
**********************************************************
    1946 Tue Jan 14 16:48:34 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.chasestrings.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# This test is checking that we can read members and that pointers inside
# members point to valid data that is intelligible, eg. strings.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
t="zelda_info_t"
exe="tst.chasestrings.exe"

elfdump "./$exe" | grep -q '.SUNW_ctf'
if [[ $? -ne 0 ]]; then
        echo "CTF does not exist in $exe, that's a bug" >&2
        exit 1
fi

./$exe &
pid=$!

$dtrace -qs /dev/stdin <<EOF
pid$pid::has_princess:entry
/next == 0/
{
        this->t = (pid$pid\`$t *)(copyin(arg0, sizeof (pid$pid\`$t)));
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            copyinstr((uintptr_t)this->t->zi_gamename), this->t->zi_ndungeons,
            copyinstr((uintptr_t)this->t->zi_villain), this->t->zi_haszelda);
        next = 1;
}

pid$pid::has_dungeons:entry
/next == 1/
{
        this->t = (pid$pid\`$t *)(copyin(arg0, sizeof (pid$pid\`$t)));
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
```



```
            copyinstr((uintptr_t)this->t->zi_gamename), this->t->zi_ndungeons,
            copyinstr((uintptr_t)this->t->zi_villain), this->t->zi_haszelda);
        next = 2;
}

pid$pid::has_villain:entry
/next == 2/
{
        this->t = (pid$pid\`$t *)(copyin(arg0, sizeof (pid$pid\`$t)));
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            copyinstr((uintptr_t)this->t->zi_gamename), this->t->zi_ndungeons,
            copyinstr((uintptr_t)this->t->zi_villain), this->t->zi_haszelda);
        exit(0);
}
EOF
rc=$?

kill -9 $pid

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
     195 Tue Jan 14 16:48:34 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.chasestrings.ksh.out
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
   1 game: Ocarina of Time, dungeon: 10, villain: Ganondorf, zelda: 1
   2 game: Link's Awakening, dungeon: 9, villain: Nightmare, zelda: 0
   3 game: A Link to the Past, dungeon: 12, villain: Ganon, zelda: 1

   5 #endif /* ! codereview */
```

```
**********************************************************
     648 Tue Jan 14 16:48:34 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.libtype.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 2013 (c) Joyent, Inc. All rights reserved.
 */

/*
 * We're linked against libc which has types, though we do not.
 */
#include <unistd.h>

int
main(void)
{
        for (;;) {
                sleep(1000);
        }
        /*NOTREACHED*/
        return (0);
}
#endif /* ! codereview */
```

```
**********************************************************
     986 Tue Jan 14 16:48:34 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.libtype.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# Here we want to make sure that the program in question does not have ctf data
# in its a.out; however, we can get types out of a linked libc.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
t="int"
exe="tst.libtype.exe"

elfdump "./$exe" | grep -q '.SUNW_ctf'
if [[ $? -eq 0 ]]; then
        echo "CTF exists in $exe, that's a bug" >&2
        exit 1
fi

./$exe &
pid=$!

rc=`$dtrace -n "BEGIN{ trace((pid$pid\`$t)0); exit(0); }"`

kill -9 $pid

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
     848 Tue Jan 14 16:48:34 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.linkmap.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# We should be able to see both strstr from libc and from ld on an
# alternate linkmap.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1

$dtrace -q -p $$ -s /dev/stdin  <<EOF
pid\$target:LM1\'ld.so.1:strstr:entry,
pid\$target:libc.so.1:strstr:entry
{
        exit (0);
}

BEGIN
{
        exit (0);
}
EOF
rc=$?

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
    1192 Tue Jan 14 16:48:35 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.pidprint.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# Use print() on userland CTF types and verify we get the data we expect.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
t="final_fantasy_info_t"
exe="tst.printtype.exe"

elfdump "./$exe" | grep -q '.SUNW_ctf'
if [[ $? -ne 0 ]]; then
        echo "CTF does not exist in $exe, that's a bug" >&2
        exit 1
fi

./$exe &
pid=$!

$dtrace -qs /dev/stdin <<EOF
pid$pid::ff_getgameid:entry
/next == 0/
{
        print(*args[0]);
        printf("\n");
        next = 1;
}

pid$pid::ff_getpartysize:entry
/next == 1/
{
        print(*args[0]);
        printf("\n");
        next = 2;
}
```



```
pid$pid::ff_getsummons:entry
/next == 2/
{
        print(*args[0]);
        printf("\n");
        exit(0);
}
EOF
rc=$?

kill -9 $pid

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
    1277 Tue Jan 14 16:48:35 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.pidprinttarg.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# Use print() on userland CTF types and verify we get the data we
# expect. This time, use $target to make sure that path works correctly.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
t="final_fantasy_info_t"
exe="tst.printtype.exe"

elfdump "./$exe" | grep -q '.SUNW_ctf'
if [[ $? -ne 0 ]]; then
        echo "CTF does not exist in $exe, that's a bug" >&2
        exit 1
fi

./$exe &
pid=$!

$dtrace -p $pid -qs /dev/stdin <<EOF
pid\$target::ff_getgameid:entry
/next == 0/
{
        print(*args[0]);
        printf("\n");
        next = 1;
}

pid\$target::ff_getpartysize:entry
/next == 1/
{
        print(*args[0]);
        printf("\n");
        next = 2;
}
```



```
pid\$target::ff_getsummons:entry
/next == 2/
{
        print(*args[0]);
        printf("\n");
        exit(0);
}
EOF
rc=$?

kill -9 $pid

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
    1415 Tue Jan 14 16:48:35 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.printtype.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 2013 (c) Joyent, Inc. All rights reserved.
 */

/*
 * The point of this is to use print() on various functions to make sure that we
 * can print basic structures. Note that we purposefully are making sure that
 * there are no pointers here.
 */
#include <unistd.h>

typedef struct final_fantasy_info {
        int ff_gameid;
        int ff_partysize;
        int ff_hassummons;
} final_fantasy_info_t;

static int
ff_getgameid(final_fantasy_info_t *f)
{
        return (0);
}

static int
ff_getpartysize(final_fantasy_info_t *f)
{
        return (0);
}

static int
ff_getsummons(final_fantasy_info_t *f)
{
        return (0);
}

int
main(void)
{
        final_fantasy_info_t ffiii, ffx, ffi;

        ffi.ff_gameid = 1;
        ffi.ff_partysize = 4;
        ffi.ff_hassummons = 0;

        ffiii.ff_gameid = 6;
```



```
        ffiii.ff_partysize = 4;
        ffiii.ff_hassummons = 1;

        ffx.ff_gameid = 10;
        ffx.ff_partysize = 3;
        ffx.ff_hassummons = 1;

        for (;;) {
                ff_getgameid(&ffi);
                ff_getpartysize(&ffx);
                ff_getsummons(&ffiii);
                sleep(1);
        }
        /*NOTREACHED*/
        return (0);
}
#endif /* ! codereview */
```

```
**********************************************************
    1324 Tue Jan 14 16:48:35 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.printtype.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# Use print() on userland CTF types and verify we get the data we expect.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
t="final_fantasy_info_t"
exe="tst.printtype.exe"

elfdump "./$exe" | grep -q '.SUNW_ctf'
if [[ $? -ne 0 ]]; then
        echo "CTF does not exist in $exe, that's a bug" >&2
        exit 1
fi

./$exe &
pid=$!

$dtrace -qs /dev/stdin <<EOF
pid$pid::ff_getgameid:entry
/next == 0/
{
        print(*(pid$pid\'$t *)(copyin(arg0, sizeof (pid$pid\'$t))));
        printf("\n");
        next = 1;
}

pid$pid::ff_getpartysize:entry
/next == 1/
{
        print(*(pid$pid\'$t *)(copyin(arg0, sizeof (pid$pid\'$t))));
        printf("\n");
        next = 2;
}
```



```
pid$pid::ff_getsummons:entry
/next == 2/
{
        print(*(pid$pid\'$t *)(copyin(arg0, sizeof (pid$pid\'$t))));
        printf("\n");
        exit(0);
}
EOF
rc=$?

kill -9 $pid

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
     311 Tue Jan 14 16:48:36 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.printtype.ksh.out
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
   1 final_fantasy_info_t {
   2     int ff_gameid = 0x1
   3     int ff_partysize = 0x4
   4     int ff_hassummons = 0
   5 }
   6 final_fantasy_info_t {
   7     int ff_gameid = 0xa
   8     int ff_partysize = 0x3
   9     int ff_hassummons = 0x1
  10 }
  11 final_fantasy_info_t {
  12     int ff_gameid = 0x6
  13     int ff_partysize = 0x4
  14     int ff_hassummons = 0x1
  15 }

  17 #endif /* ! codereview */
```

```
**********************************************************
    1354 Tue Jan 14 16:48:36 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.printtypetarg.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# Use print() on userland CTF types and verify we get the data we
# expect. Use the pid‘ alias for $target.
#

if [ $# != 1 ]; then
        echo expected one argument: ’<’dtrace-path’>’
        exit 2
fi

dtrace=$1
t="final_fantasy_info_t"
exe="tst.printtype.exe"

elfdump "./$exe" | grep -q ’.SUNW_ctf’
if [[ $? -ne 0 ]]; then
        echo "CTF does not exist in $exe, that’s a bug" >&2
        exit 1
fi

./$exe &
pid=$!

$dtrace -p $pid -qs /dev/stdin <<EOF
pid\$target::ff_getgameid:entry
/next == 0/
{
        print(*(pid\‘$t *)(copyin(arg0, sizeof (pid\‘$t))));
        printf("\n");
        next = 1;
}

pid\$target::ff_getpartysize:entry
/next == 1/
{
        print(*(pid\‘$t *)(copyin(arg0, sizeof (pid\‘$t))));
        printf("\n");
        next = 2;
}
```



```
pid\$target::ff_getsummons:entry
/next == 2/
{
        print(*(pid\‘$t *)(copyin(arg0, sizeof (pid\‘$t))));
        printf("\n");
        exit(0);
}
EOF
rc=$?

kill -9 $pid

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
    1882 Tue Jan 14 16:48:36 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.userlandkey.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# Simple test that if we manually use the userland keyword that it
# works.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
t="zelda_info_t"
exe="tst.chasestrings.exe"

elfdump "./$exe" | grep -q '.SUNW_ctf'
if [[ $? -ne 0 ]]; then
        echo "CTF does not exist in $exe, that's a bug" >&2
        exit 1
fi

./$exe &
pid=$!

$dtrace -32 -qs /dev/stdin <<EOF
typedef struct info {
        char    *zi_gamename;
        int     zi_ndungeons;
        char    *zi_villain;
        int     zi_haszelda;
} info_t;

pid$pid::has_princess:entry
/next == 0/
{
        this->t = (userland info_t *)arg0;
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            stringof(this->t->zi_gamename), this->t->zi_ndungeons,
            stringof(this->t->zi_villain), this->t->zi_haszelda);
        next = 1;
```



```
}

pid$pid::has_dungeons:entry
/next == 1/
{
        this->t = (userland info_t *)arg0;
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            stringof(this->t->zi_gamename), this->t->zi_ndungeons,
            stringof(this->t->zi_villain), this->t->zi_haszelda);
        next = 2;
}

pid$pid::has_villain:entry
/next == 2/
{
        this->t = (userland info_t *)arg0;
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            stringof(this->t->zi_gamename), this->t->zi_ndungeons,
            stringof(this->t->zi_villain), this->t->zi_haszelda);
        exit(0);
}
EOF
rc=$?

kill -9 $pid

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
     195 Tue Jan 14 16:48:36 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.userlandkey.ksh.out
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
   1 game: Ocarina of Time, dungeon: 10, villain: Ganondorf, zelda: 1
   2 game: Link's Awakening, dungeon: 9, villain: Nightmare, zelda: 0
   3 game: A Link to the Past, dungeon: 12, villain: Ganon, zelda: 1

   5 #endif /* ! codereview */
```

```
**********************************************************
    1665 Tue Jan 14 16:48:36 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.userstrings.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# 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 Joyent, Inc. All rights reserved.
#

#
# This test is checking that we can read members and that pointers inside
# members point to valid data that is intelligible, eg. strings.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
exe="tst.chasestrings.exe"

elfdump "./$exe" | grep -q '.SUNW_ctf'
if [[ $? -ne 0 ]]; then
        echo "CTF does not exist in $exe, that's a bug" >&2
        exit 1
fi

./$exe &
pid=$!

$dtrace -qs /dev/stdin <<EOF
pid$pid::has_princess:entry
/next == 0/
{
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            stringof(args[0]->zi_gamename), args[0]->zi_ndungeons,
            stringof(args[0]->zi_villain), args[0]->zi_haszelda);
        next = 1;
}

pid$pid::has_dungeons:entry
/next == 1/
{
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            stringof(args[0]->zi_gamename), args[0]->zi_ndungeons,
            stringof(args[0]->zi_villain), args[0]->zi_haszelda);
        next = 2;
```



```
}

pid$pid::has_villain:entry
/next == 2/
{
        printf("game: %s, dungeon: %d, villain: %s, zelda: %d\n",
            stringof(args[0]->zi_gamename), args[0]->zi_ndungeons,
            stringof(args[0]->zi_villain), args[0]->zi_haszelda);
        exit(0);
}
EOF
rc=$?

kill -9 $pid

exit $rc
#endif /* ! codereview */
```

```
**********************************************************
     195 Tue Jan 14 16:48:37 2014
new/usr/src/cmd/dtrace/test/tst/common/uctf/tst.userstrings.ksh.out
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
   1 game: Ocarina of Time, dungeon: 10, villain: Ganondorf, zelda: 1
   2 game: Link's Awakening, dungeon: 9, villain: Nightmare, zelda: 0
   3 game: A Link to the Past, dungeon: 12, villain: Ganon, zelda: 1

   5 #endif /* ! codereview */
```

```
**********************************************************
    2119 Tue Jan 14 16:48:37 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.corruptenv.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
#!/usr/bin/ksh
#
# 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.
#

#
# This test verifies that a program that corrupts its own environment
# without inducing a crash does not crash solely due to drti.o's use of
# getenv(3C).
#

PATH=/usr/bin:/usr/sbin:$PATH

if (( $# != 1 )); then
        print -u2 'expected one argument: <dtrace-path>'
        exit 2
fi

#
# jdtrace does not implement the -h option that is required to generate
# C header files.
#
if [[ "$1" == */jdtrace ]]; then
        exit 0
fi

dtrace="$1"
startdir="$PWD"
dir=$(mktemp -td drtiXXXXXX)
if (( $? != 0 )); then
        print -u2 'Could not create safe temporary directory'
        exit 2
fi
```



```
cd "$dir"

cat > Makefile <<EOF
all: main

main: main.o prov.o
        gcc -m32 -o main main.o prov.o
        gcc -o main main.o prov.o

main.o: main.c prov.h
        gcc -m32 -c main.c
        gcc -c main.c

prov.h: prov.d
        $dtrace -h -s prov.d

prov.o: prov.d main.o
        $dtrace -G -32 -s prov.d main.o
EOF

cat > prov.d <<EOF
provider tester {
        probe entry();
};
```

*unchanged_portion_omitted_*

```
**********************************************************
    2697 Tue Jan 14 16:48:37 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.dlclose1.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

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

#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# This test verifies that USDT providers are removed when its associated
# load object is closed via dlclose(3dl).
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
DIR=/var/tmp/dtest.$$

mkdir $DIR
cd $DIR

cat > Makefile <<EOF
all: main livelib.so deadlib.so

main: main.o prov.o
        gcc -m32 -o main main.o
        gcc -o main main.o

main.o: main.c
        gcc -m32 -c main.c
        gcc -c main.c


livelib.so: livelib.o prov.o
```



```
        gcc -m32 -shared -o livelib.so livelib.o prov.o -lc
        gcc -shared -o livelib.so livelib.o prov.o -lc

livelib.o: livelib.c prov.h
        gcc -m32 -fPIC -c livelib.c
        gcc -c livelib.c

prov.o: livelib.o prov.d
        $dtrace -G -s prov.d livelib.o

prov.h: prov.d
        $dtrace -h -s prov.d


deadlib.so: deadlib.o
        gcc -m32 -shared -o deadlib.so deadlib.o -lc
        gcc -shared -o deadlib.so deadlib.o -lc

deadlib.o: deadlib.c
        gcc -m32 -fPIC -c deadlib.c
        gcc -c deadlib.c

clean:
        rm -f main.o livelib.o prov.o prov.h deadlib.o

clobber: clean
        rm -f main livelib.so deadlib.so
EOF

cat > prov.d <<EOF
provider test_prov {
        probe go();
};
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2887 Tue Jan 14 16:48:38 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.dlclose2.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

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

#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
DIR=/var/tmp/dtest.$$

mkdir $DIR
cd $DIR

cat > Makefile <<EOF
all: main livelib.so deadlib.so

main: main.o prov.o
        gcc -m32 -o main main.o
        gcc -o main main.o

main.o: main.c
        gcc -m32 -c main.c
        gcc -c main.c


livelib.so: livelib.o prov.o
        gcc -m32 -shared -o livelib.so livelib.o prov.o -lc
        gcc -shared -o livelib.so livelib.o prov.o -lc

livelib.o: livelib.c prov.h
        gcc -m32 -fPIC -c livelib.c
```



```
        gcc -c livelib.c

prov.o: livelib.o prov.d
        $dtrace -G -s prov.d livelib.o

prov.h: prov.d
        $dtrace -h -s prov.d


deadlib.so: deadlib.o
        gcc -m32 -shared -o deadlib.so deadlib.o -lc
        gcc -shared -o deadlib.so deadlib.o -lc

deadlib.o: deadlib.c
        gcc -m32 -fPIC -c deadlib.c
        gcc -c deadlib.c

clean:
        rm -f main.o livelib.o prov.o prov.h deadlib.o

clobber: clean
        rm -f main livelib.so deadlib.so
EOF

cat > prov.d <<EOF
provider test_prov {
        probe go();
};
```

*unchanged_portion_omitted_*

```
**********************************************************
    2752 Tue Jan 14 16:48:38 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.dlclose3.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

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

#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# This test verifies that performing a dlclose(3dl) on a library doesn't
# cause existing pid provider probes to become invalid.
#

if [ $# != 1 ]; then
        echo expected one argument: '<'dtrace-path'>'
        exit 2
fi

dtrace=$1
DIR=/var/tmp/dtest.$$

mkdir $DIR
cd $DIR

cat > Makefile <<EOF
all: main livelib.so deadlib.so

main: main.o prov.o
        gcc -m32 -o main main.o
        gcc -o main main.o

main.o: main.c
        gcc -m32 -c main.c
        gcc -c main.c


livelib.so: livelib.o prov.o
```



```
        gcc -m32 -shared -o livelib.so livelib.o prov.o -lc
        gcc -shared -o livelib.so livelib.o prov.o -lc

livelib.o: livelib.c prov.h
        gcc -m32 -fPIC -c livelib.c
        gcc -c livelib.c

prov.o: livelib.o prov.d
        $dtrace -G -s prov.d livelib.o

prov.h: prov.d
        $dtrace -h -s prov.d


deadlib.so: deadlib.o
        gcc -m32 -shared -o deadlib.so deadlib.o -lc
        gcc -shared -o deadlib.so deadlib.o -lc

deadlib.o: deadlib.c
        gcc -m32 -fPIC -c deadlib.c
        gcc -c deadlib.c

clean:
        rm -f main.o livelib.o prov.o prov.h deadlib.o

clobber: clean
        rm -f main livelib.so deadlib.so
EOF

cat > prov.d <<EOF
provider test_prov {
        probe go();
};
______unchanged_portion_omitted_
```

```
***********************************************************
    2066 Tue Jan 14 16:48:39 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.eliminate.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
  72 EOF

  74 gcc -m32 -c test.c
  74 gcc -c test.c
  75 if [ $? -ne 0 ]; then
  76         print -u2 "failed to compile test.c"
  77         exit 1
  78 fi
  79 $dtrace -G -32 -s prov.d test.o
  80 if [ $? -ne 0 ]; then
  81         print -u2 "failed to create DOF"
  82         exit 1
  83 fi
  84 gcc -m32 -o test test.o prov.o
  84 gcc -o test test.o prov.o
  85 if [ $? -ne 0 ]; then
  86         print -u2 "failed to link final executable"
  87         exit 1
  88 fi

  90 nm test.o | grep \$dtrace > /dev/null
  91 if [ $? -ne 0 ]; then
  92         print -u2 "no temporary symbols in the object file"
  93         exit 1
  94 fi

  96 nm test | grep \$dtrace > /dev/null
  97 if [ $? -eq 0 ]; then
  98         print -u2 "failed to eliminate temporary symbols"
  99         exit 1
 100 fi

 102 cd /
 103 /usr/bin/rm -rf $DIR

 105 exit 0
```

```
***********************************************************
    1848 Tue Jan 14 16:48:39 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.enabled.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
  61 EOF

  63 gcc -m32 -c test.c
  63 gcc -c test.c
  64 if [ $? -ne 0 ]; then
  65         print -u2 "failed to compile test.c"
  66         exit 1
  67 fi
  68 $dtrace -G -32 -s prov.d test.o
  69 if [ $? -ne 0 ]; then
  70         print -u2 "failed to create DOF"
  71         exit 1
  72 fi
  73 gcc -m32 -o test test.o prov.o
  73 gcc -o test test.o prov.o
  74 if [ $? -ne 0 ]; then
  75         print -u2 "failed to link final executable"
  76         exit 1
  77 fi

  79 script()
  80 {
  81         $dtrace -c ./test -qs /dev/stdin <<EOF
  82         test_prov\$target:::
  83         {
  84                 printf("%s:%s:%s\n", probemod, probefunc, probename);
  85         }
  86 EOF
  87 }
______unchanged_portion_omitted_
```

```
***********************************************************
    2133 Tue Jan 14 16:48:40 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.enabled2.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
  77 EOF

  79 gcc -m32 -c test.c
  79 gcc -c test.c
  80 if [ $? -ne 0 ]; then
  81         print -u2 "failed to compile test.c"
  82         exit 1
  83 fi
  84 $dtrace -G -32 -s prov.d test.o
  85 if [ $? -ne 0 ]; then
  86         print -u2 "failed to create DOF"
  87         exit 1
  88 fi

  90 gcc -m32 -o test test.o prov.o
  89 gcc -o test test.o prov.o
  91 if [ $? -ne 0 ]; then
  92         print -u2 "failed to link final executable"
  93         exit 1
  94 fi

  96 script()
  97 {
  98         ./test

 100         $dtrace -c ./test -qs /dev/stdin <<EOF
 101         test_prov\$target:::
 102         {
 103         }
 104 EOF
 105 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2314 Tue Jan 14 16:48:40 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.entryreturn.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
______unchanged_portion_omitted_
  70 EOF

  72 gcc -m32 -c test.c
  72 gcc -c test.c
  73 if [ $? -ne 0 ]; then
  74         print -u2 "failed to compile test.c"
  75         exit 1
  76 fi
  77 $dtrace -G -32 -s prov.d test.o
  78 if [ $? -ne 0 ]; then
  79         print -u2 "failed to create DOF"
  80         exit 1
  81 fi
  82 gcc -m32 -o test test.o prov.o
  82 gcc -o test test.o prov.o
  83 if [ $? -ne 0 ]; then
  84         print -u2 "failed to link final executable"
  85         exit 1
  86 fi

  88 script()
  89 {
  90         $dtrace -wqZFs /dev/stdin <<EOF
  91         BEGIN
  92         {
  93                 system("$DIR/test");
  94                 printf("\n");
  95         }

  97         test_prov*:::done
  98         /progenyof(\$pid)/
  99         {
 100                 exit(0);
 101         }

 103         test_prov*:::
 104         /progenyof(\$pid)/
 105         {
 106                 printf("\n");
 107         }
 108 EOF
 109 }
______unchanged_portion_omitted_
```

```
***********************************************************
    1970 Tue Jan 14 16:48:41 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.fork.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
  71 EOF

  73 gcc -m32 -c test.c
  73 gcc -c test.c
  74 if [ $? -ne 0 ]; then
  75         print -u2 "failed to compile test.c"
  76         exit 1
  77 fi
  78 $dtrace -G -32 -s prov.d test.o
  79 if [ $? -ne 0 ]; then
  80         print -u2 "failed to create DOF"
  81         exit 1
  82 fi
  83 gcc -m32 -o test test.o prov.o
  83 gcc -o test test.o prov.o
  84 if [ $? -ne 0 ]; then
  85         print -u2 "failed to link final executable"
  86         exit 1
  87 fi

  89 script() {
  90         $dtrace -c ./test -Zqs /dev/stdin <<EOF
  91         test_prov*:::
  92         {
  93                 printf("%s:%s:%s\n", probemod, probefunc, probename);
  94         }
  95 EOF
  96 }
______unchanged_portion_omitted_
```

```
***********************************************************
    1786 Tue Jan 14 16:48:41 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.header.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
```

```
 65 EOF

 67 gcc -m32 -c test.c
 67 gcc -c test.c
 68 if [ $? -ne 0 ]; then
 69         print -u2 "failed to compile test.c"
 70         exit 1
 71 fi
 72 $dtrace -G -32 -s prov.d test.o
 73 if [ $? -ne 0 ]; then
 74         print -u2 "failed to create DOF"
 75         exit 1
 76 fi
 77 gcc -m32 -o test test.o prov.o
 77 gcc -o test test.o prov.o
 78 if [ $? -ne 0 ]; then
 79         print -u2 "failed to link final executable"
 80         exit 1
 81 fi

 83 cd /
 84 /usr/bin/rm -rf $DIR
```

```
***********************************************************
    1981 Tue Jan 14 16:48:41 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.linkpriv.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
  62 EOF

  64 gcc -m32 -c test.c
  64 gcc -c test.c
  65 if [ $? -ne 0 ]; then
  66         print -u2 "failed to compile test.c"
  67         exit 1
  68 fi
  69 $dtrace -G -32 -s prov.d test.o
  70 if [ $? -ne 0 ]; then
  71         print -u2 "failed to create DOF"
  72         exit 1
  73 fi
  74 gcc -m32 -o test test.o prov.o
  74 gcc -o test test.o prov.o
  75 if [ $? -ne 0 ]; then
  76         print -u2 "failed to link final executable"
  77         exit 1
  78 fi

  80 cd /
  81 /usr/bin/rm -rf $DIR
```

```
***********************************************************
    2002 Tue Jan 14 16:48:42 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.linkunpriv.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
  64 EOF

  66 gcc -m32 -c test.c
  66 gcc -c test.c
  67 if [ $? -ne 0 ]; then
  68         print -u2 "failed to compile test.c"
  69         exit 1
  70 fi
  71 $dtrace -G -32 -s prov.d test.o
  72 if [ $? -ne 0 ]; then
  73         print -u2 "failed to create DOF"
  74         exit 1
  75 fi
  76 gcc -m32 -o test test.o prov.o
  76 gcc -o test test.o prov.o
  77 if [ $? -ne 0 ]; then
  78         print -u2 "failed to link final executable"
  79         exit 1
  80 fi

  82 cd /
  83 /usr/bin/rm -rf $DIR
```

```
**********************************************************
    1892 Tue Jan 14 16:48:42 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.multiple.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
______unchanged_portion_omitted_
  65 EOF

  67 gcc -m32 -c test.c
  67 gcc -c test.c
  68 if [ $? -ne 0 ]; then
  69         print -u2 "failed to compile test.c"
  70         exit 1
  71 fi
  72 $dtrace -G -32 -s prov.d test.o
  73 if [ $? -ne 0 ]; then
  74         print -u2 "failed to create DOF"
  75         exit 1
  76 fi
  77 gcc -m32 -o test test.o prov.o
  77 gcc -o test test.o prov.o
  78 if [ $? -ne 0 ]; then
  79         print -u2 "failed to link final executable"
  80         exit 1
  81 fi

  83 script() {
  84         $dtrace -c ./test -qs /dev/stdin <<EOF
  85         test_prov\$target:::
  86         {
  87                 printf("%s:%s:%s\n", probemod, probefunc, probename);
  88         }
  89 EOF
  90 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2005 Tue Jan 14 16:48:43 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.multiprov.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
______unchanged_portion_omitted_
EOF

        gcc -m32 -c $oogle.c
        cc -c $oogle.c

        if [ $? -ne 0 ]; then
                print -u2 "failed to compile $oogle.c"
                exit 1
        fi

        $dtrace -G -32 -s $oogle.d $oogle.o -o $oogle.d.o

        if [ $? -ne 0 ]; then
                print -u2 "failed to process $oogle.d"
                exit 1
        fi

        objs="$objs $oogle.o $oogle.d.o"
        echo $oogle'();' >> test.c
        echo $oogle'$target:::{@[probefunc] = count()}' >> test.d
done

echo "}" >> test.c

echo 'END{printa("%-10s %@d\\n", @)}' >> test.d

gcc -m32 -o test test.c $objs
cc -o test test.c $objs

if [ $? -ne 0 ]; then
        print -u2 "failed to compile test.c"
        exit 1
fi

$dtrace -s ./test.d -Zc ./test

if [ $? -ne 0 ]; then
        print -u2 "failed to execute test"
        exit 1
fi

cd /
/usr/bin/rm -rf $DIR
exit 0
```

```
***********************************************************
    1294 Tue Jan 14 16:48:44 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.noprobes.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
  47 EOF

  49 gcc -m32 -c test.c
  49 cc -c test.c
  50 $dtrace -G -32 -s doogle.d test.o -o doogle.d.o

  52 if [ $? -eq 0 ]; then
  53         print -u2 "dtrace succeeded despite having no probe sites"
  54         exit 1
  55 fi

  57 cd /
  58 /usr/bin/rm -rf $DIR
  59 exit 0
```

```
**********************************************************
    2322 Tue Jan 14 16:48:44 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.noreap.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
______unchanged_portion_omitted_
  52 EOF

  54 gcc -m32 -c test.c
  54 gcc -c test.c
  55 if [ $? -ne 0 ]; then
  56         print -u2 "failed to compile test.c"
  57         exit 1
  58 fi
  59 $dtrace -G -32 -s prov.d test.o
  60 if [ $? -ne 0 ]; then
  61         print -u2 "failed to create DOF"
  62         exit 1
  63 fi

  65 gcc -m32 -o test test.o prov.o
  64 gcc -o test test.o prov.o
  66 if [ $? -ne 0 ]; then
  67         print -u2 "failed to link final executable"
  68         exit 1
  69 fi

  71 script()
  72 {
  73         $dtrace -Zwqs /dev/stdin <<EOF

  75         BEGIN
  76         {
  77                 spec = speculation();
  78                 speculate(spec);
  79                 printf("this is speculative!\n");
  80         }

  82         test_prov*:::
  83         {
  84                 probeid = id;
  85         }

  87         tick-1sec
  88         /probeid == 0/
  89         {
  90                 printf("launching test\n");
  91                 system("./test");
  92         }

  94         tick-1sec
  95         /probeid != 0/
  96         {
  97                 printf("attempting re-enabling\n");
  98                 system("dtrace -e -x errtags -i %d", probeid);
  99                 attempts++;
 100         }

 102         tick-1sec
 103         /attempts > 10/
 104         {
```



```
 105                 exit(0);
 106         }
 107 EOF
 108 }
______unchanged_portion_omitted_
```

```
***********************************************************
    2405 Tue Jan 14 16:48:45 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.noreapring.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
  52 EOF

  54 gcc -m32 -c test.c
  54 gcc -c test.c
  55 if [ $? -ne 0 ]; then
  56         print -u2 "failed to compile test.c"
  57         exit 1
  58 fi
  59 $dtrace -G -32 -s prov.d test.o
  60 if [ $? -ne 0 ]; then
  61         print -u2 "failed to create DOF"
  62         exit 1
  63 fi

  65 gcc -m32 -o test test.o prov.o
  64 gcc -o test test.o prov.o
  66 if [ $? -ne 0 ]; then
  67         print -u2 "failed to link final executable"
  68         exit 1
  69 fi

  71 script()
  72 {
  73         $dtrace -Zwqs /dev/stdin <<EOF
  74         test_prov*:::
  75         {
  76                 probeid = id;
  77         }

  79         tick-1sec
  80         /probeid == 0/
  81         {
  82                 printf("launching test\n");
  83                 system("./test");
  84         }

  86         tick-1sec
  87         /probeid != 0/
  88         {
  89                 printf("attempting re-enabling\n");
  90                 system("dtrace -e -x errtags -i %d", probeid);
  91                 attempts++;
  92         }

  94         tick-1sec
  95         /attempts > 10/
  96         {
  97                 exit(0);
  98         }
  99 EOF
 100 }
______unchanged_portion_omitted_
```

```
***********************************************************
    1679 Tue Jan 14 16:48:45 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.onlyenabled.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
  62 EOF

  64 gcc -m32 -c test.c
  64 gcc -c test.c
  65 if [ $? -ne 0 ]; then
  66         print -u2 "failed to compile test.c"
  67         exit 1
  68 fi
  69 $dtrace -G -32 -s prov.d test.o
  70 if [ $? -ne 0 ]; then
  71         print -u2 "failed to create DOF"
  72         exit 1
  73 fi
  74 gcc -m32 -o test test.o prov.o
  74 gcc -o test test.o prov.o
  75 if [ $? -ne 0 ]; then
  76         print -u2 "failed to link final executable"
  77         exit 1
  78 fi

  80 cd /
  81 /usr/bin/rm -rf $DIR
```

****
2224 Tue Jan 14 16:48:46 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.reap.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
****
_____***unchanged_portion_omitted_***

```
 52 EOF

 54 gcc -m32 -c test.c
 54 gcc -c test.c
 55 if [ $? -ne 0 ]; then
 56         print -u2 "failed to compile test.c"
 57         exit 1
 58 fi
 59 $dtrace -G -32 -s prov.d test.o
 60 if [ $? -ne 0 ]; then
 61         print -u2 "failed to create DOF"
 62         exit 1
 63 fi

 65 gcc -m32 -o test test.o prov.o
 64 gcc -o test test.o prov.o
 66 if [ $? -ne 0 ]; then
 67         print -u2 "failed to link final executable"
 68         exit 1
 69 fi

 71 script()
 72 {
 73         $dtrace -Zwqs /dev/stdin <<EOF
 74         test_prov*:::
 75         {
 76                 probeid = id;
 77         }

 79         tick-1sec
 80         /probeid == 0/
 81         {
 82                 printf("launching test\n");
 83                 system("./test");
 84         }

 86         tick-1sec
 87         /probeid != 0/
 88         {
 89                 printf("attempting re-enabling\n");
 90                 system("dtrace -e -x errtags -i %d", probeid);
 91                 attempts++;
 92         }

 94         tick-1sec
 95         /attempts > 10/
 96         {
 97                 exit(0);
 98         }
 99 EOF
100 }
```

_____***unchanged_portion_omitted_***

```
***********************************************************
    1772 Tue Jan 14 16:48:46 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.reeval.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
EOF

gcc -m32 -c test.c
gcc -c test.c
if [ $? -ne 0 ]; then
        print -u2 "failed to compile test.c"
        exit 1
fi
$dtrace -G -32 -s prov.d test.o
if [ $? -ne 0 ]; then
        print -u2 "failed to create DOF"
        exit 1
fi
gcc -m32 -o test test.o prov.o
gcc -o test test.o prov.o
if [ $? -ne 0 ]; then
        print -u2 "failed to link final executable"
        exit 1
fi

script()
{
        $dtrace -wZs /dev/stdin <<EOF
        BEGIN
        {
                system("$DIR/test");
        }

        test_prov*:::
        {
                seen = 1;
        }

        proc:::exit
        /progenyof(\$pid) && execname == "test"/
        {
                exit(seen ? 0 : 2);
        }
EOF
}
______unchanged_portion_omitted_
```

```
***********************************************************
    1889 Tue Jan 14 16:48:47 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.static.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
***********************************************************
______unchanged_portion_omitted_
  63 EOF

  65 gcc -m32 -c test.c
  65 gcc -c test.c
  66 if [ $? -ne 0 ]; then
  67         print -u2 "failed to compile test.c"
  68         exit 1
  69 fi
  70 $dtrace -G -32 -s prov.d test.o
  71 if [ $? -ne 0 ]; then
  72         print -u2 "failed to create DOF"
  73         exit 1
  74 fi
  75 gcc -m32 -o test test.o prov.o
  75 gcc -o test test.o prov.o
  76 if [ $? -ne 0 ]; then
  77         print -u2 "failed to link final executable"
  78         exit 1
  79 fi

  81 script()
  82 {
  83         $dtrace -c ./test -qs /dev/stdin <<EOF
  84         test_prov\$target:::
  85         {
  86                 printf("%s:%s:%s\n", probemod, probefunc, probename);
  87         }
  88 EOF
  89 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2239 Tue Jan 14 16:48:47 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.static2.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
______unchanged_portion_omitted_
  67 EOF

  69 gcc -m32 -c test.c
  69 gcc -c test.c
  70 if [ $? -ne 0 ]; then
  71         print -u2 "failed to compile test.c"
  72         exit 1
  73 fi
  74 $dtrace -G -32 -s prov.d test.o
  75 if [ $? -ne 0 ]; then
  76         print -u2 "failed to create initial DOF"
  77         exit 1
  78 fi
  79 rm -f prov.o
  80 $dtrace -G -32 -s prov.d test.o
  81 if [ $? -ne 0 ]; then
  82         print -u2 "failed to create final DOF"
  83         exit 1
  84 fi
  85 gcc -m32 -o test test.o prov.o
  85 gcc -o test test.o prov.o
  86 if [ $? -ne 0 ]; then
  87         print -u2 "failed to link final executable"
  88         exit 1
  89 fi

  91 script()
  92 {
  93         $dtrace -c ./test -qs /dev/stdin <<EOF
  94         test_prov\$target:::
  95         {
  96                 printf("%s:%s:%s\n", probemod, probefunc, probename);
  97         }
  98 EOF
  99 }
______unchanged_portion_omitted_
```

****
1864 Tue Jan 14 16:48:47 2014
new/usr/src/cmd/dtrace/test/tst/common/usdt/tst.user.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
****
_____***unchanged_portion_omitted_***

```
 62 EOF

 64 gcc -m32 -c test.c
 64 gcc -c test.c
 65 if [ $? -ne 0 ]; then
 66         print -u2 "failed to compile test.c"
 67         exit 1
 68 fi
 69 $dtrace -G -32 -s prov.d test.o
 70 if [ $? -ne 0 ]; then
 71         print -u2 "failed to create DOF"
 72         exit 1
 73 fi
 74 gcc -m32 -o test test.o prov.o
 74 gcc -o test test.o prov.o
 75 if [ $? -ne 0 ]; then
 76         print -u2 "failed to link final executable"
 77         exit 1
 78 fi

 80 script() {
 81         $dtrace -c 'ppriv -e -s A=basic ./test' -Zqs /dev/stdin <<EOF
 82         test_prov\$target:::
 83         {
 84                 printf("%s:%s:%s\n", probemod, probefunc, probename);
 85         }
 86 EOF
 87 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    2322 Tue Jan 14 16:48:48 2014
new/usr/src/cmd/dtrace/test/tst/sparc/usdt/tst.tailcall.ksh
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
_____unchanged_portion_omitted_
  76 EOF

  78 /usr/ccs/bin/as -xregsym=no -P -D_ASM -o test.o test.s
  79 if [ $? -ne 0 ]; then
  80         print -u2 "failed to compile test.s"
  81         exit 1
  82 fi

  84 $dtrace -G -32 -s prov.d test.o
  85 if [ $? -ne 0 ]; then
  86         print -u2 "failed to create DOF"
  87         exit 1
  88 fi

  90 gcc -m32 -o test test.o prov.o
  90 gcc -o test test.o prov.o
  91 if [ $? -ne 0 ]; then
  92         print -u2 "failed to link final executable"
  93         exit 1
  94 fi

  96 $dtrace -c ./test -s /dev/stdin <<EOF
  97 test\$target:::fire
  98 /arg0 == 9 && arg1 == 19 && arg2 == 2006/
  99 {
 100         printf("%d/%d/%d", arg0, arg1, arg2);
 101         exit(0);
 102 }
_____unchanged_portion_omitted_
```

```
**********************************************************
   29102 Tue Jan 14 16:48:49 2014
new/usr/src/common/ctf/ctf_open.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013, Joyent, Inc.  All rights reserved.
 * Copyright (c) 2012, Joyent, Inc.  All rights reserved.
 */

#include <ctf_impl.h>
#include <sys/mman.h>
#include <sys/zmod.h>

static const ctf_dmodel_t _libctf_models[] = {
        { "ILP32", CTF_MODEL_ILP32, 4, 1, 2, 4, 4 },
        { "LP64", CTF_MODEL_LP64, 8, 1, 2, 4, 8 },
        { NULL, 0, 0, 0, 0, 0, 0 }
};
```

*unchanged_portion_omitted_*

```
/*
 * Dupliate a ctf_file_t and its underlying section information into a new
 * container. This works by copying the three ctf_sect_t's of the original
 * container if they exist and passing those into ctf_bufopen. To copy those, we
 * mmap anonymous memory with ctf_data_alloc and bcopy the data across. It's not
 * the cheapest thing, but it's what we've got.
 */
ctf_file_t *
ctf_dup(ctf_file_t *ofp)
{
        ctf_file_t *fp;
        ctf_sect_t ctfsect, symsect, strsect;
        ctf_sect_t *ctp, *symp, *strp;
        void *cbuf, *symbuf, *strbuf;
```



```
        int err;

        cbuf = symbuf = strbuf = NULL;
        /*
         * The ctfsect isn't allowed to not exist, but the symbol and string
         * section might not. We only need to copy the data of the section, not
         * the name, as ctf_bufopen will take care of that.
         */
        bcopy(&ofp->ctf_data, &ctfsect, sizeof (ctf_sect_t));
        cbuf = ctf_data_alloc(ctfsect.cts_size);
        if (cbuf == NULL) {
                (void) ctf_set_errno(ofp, ECTF_MMAP);
                return (NULL);
        }

        bcopy(ctfsect.cts_data, cbuf, ctfsect.cts_size);
        ctf_data_protect(cbuf, ctfsect.cts_size);
        ctfsect.cts_data = cbuf;
        ctfsect.cts_offset = 0;
        ctp = &ctfsect;

        if (ofp->ctf_symtab.cts_data != NULL) {
                bcopy(&ofp->ctf_symtab, &symsect, sizeof (ctf_sect_t));
                symbuf = ctf_data_alloc(symsect.cts_size);
                if (symbuf == NULL) {
                        (void) ctf_set_errno(ofp, ECTF_MMAP);
                        goto err;
                }
                bcopy(symsect.cts_data, symbuf, symsect.cts_size);
                ctf_data_protect(symbuf, symsect.cts_size);
                symsect.cts_data = symbuf;
                symsect.cts_offset = 0;
                symp = &symsect;
        } else {
                symp = NULL;
        }

        if (ofp->ctf_strtab.cts_data != NULL) {
                bcopy(&ofp->ctf_strtab, &strsect, sizeof (ctf_sect_t));
                strbuf = ctf_data_alloc(strsect.cts_size);
                if (strbuf == NULL) {
                        (void) ctf_set_errno(ofp, ECTF_MMAP);
                        goto err;
                }
                bcopy(strsect.cts_data, strbuf, strsect.cts_size);
                ctf_data_protect(strbuf, strsect.cts_size);
                strsect.cts_data = strbuf;
                strsect.cts_offset = 0;
                strp = &strsect;
        } else {
                strp = NULL;
        }

        fp = ctf_bufopen(ctp, symp, strp, &err);
        if (fp == NULL) {
                (void) ctf_set_errno(ofp, err);
                goto err;
        }

        fp->ctf_flags |= LCTF_MMAP;

        return (fp);

err:
        ctf_data_free(cbuf, ctfsect.cts_size);
        if (symbuf != NULL)
```

```
870                 ctf_data_free(symbuf, symsect.cts_size);
871         if (strbuf != NULL)
872                 ctf_data_free(strbuf, strsect.cts_size);
873         return (NULL);
874 }

876 /*
877 #endif /* ! codereview */
878  * Close the specified CTF container and free associated data structures.  Note
879  * that ctf_close() is a reference counted operation: if the specified file is
880  * the parent of other active containers, its reference count will be greater
881  * than one and it will be freed later when no active children exist.
882  */
883 void
884 ctf_close(ctf_file_t *fp)
885 {
886         ctf_dtdef_t *dtd, *ntd;

888         if (fp == NULL)
889                 return; /* allow ctf_close(NULL) to simplify caller code */

891         ctf_dprintf("ctf_close(%p) refcnt=%u\n", (void *)fp, fp->ctf_refcnt);

893         if (fp->ctf_refcnt > 1) {
894                 fp->ctf_refcnt--;
895                 return;
896         }

898         if (fp->ctf_parent != NULL)
899                 ctf_close(fp->ctf_parent);

901         /*
902          * Note, to work properly with reference counting on the dynamic
903          * section, we must delete the list in reverse.
904          */
905         for (dtd = ctf_list_prev(&fp->ctf_dtdefs); dtd != NULL; dtd = ntd) {
906                 ntd = ctf_list_prev(dtd);
907                 ctf_dtd_delete(fp, dtd);
908         }

910         ctf_free(fp->ctf_dthash, fp->ctf_dthashlen * sizeof (ctf_dtdef_t *));

912         if (fp->ctf_flags & LCTF_MMAP) {
913                 if (fp->ctf_data.cts_data != NULL)
914                         ctf_sect_munmap(&fp->ctf_data);
915                 if (fp->ctf_symtab.cts_data != NULL)
916                         ctf_sect_munmap(&fp->ctf_symtab);
917                 if (fp->ctf_strtab.cts_data != NULL)
918                         ctf_sect_munmap(&fp->ctf_strtab);
919         }

921         if (fp->ctf_data.cts_name != _CTF_NULLSTR &&
922             fp->ctf_data.cts_name != NULL) {
923                 ctf_free((char *)fp->ctf_data.cts_name,
924                     strlen(fp->ctf_data.cts_name) + 1);
925         }

927         if (fp->ctf_symtab.cts_name != _CTF_NULLSTR &&
928             fp->ctf_symtab.cts_name != NULL) {
929                 ctf_free((char *)fp->ctf_symtab.cts_name,
930                     strlen(fp->ctf_symtab.cts_name) + 1);
931         }

933         if (fp->ctf_strtab.cts_name != _CTF_NULLSTR &&
934             fp->ctf_strtab.cts_name != NULL) {
935                 ctf_free((char *)fp->ctf_strtab.cts_name,
```



```
936                     strlen(fp->ctf_strtab.cts_name) + 1);
937         }

939         if (fp->ctf_base != fp->ctf_data.cts_data && fp->ctf_base != NULL)
940                 ctf_data_free((void *)fp->ctf_base, fp->ctf_size);

942         if (fp->ctf_sxlate != NULL)
943                 ctf_free(fp->ctf_sxlate, sizeof (uint_t) * fp->ctf_nsyms);

945         if (fp->ctf_txlate != NULL) {
946                 ctf_free(fp->ctf_txlate,
947                     sizeof (uint_t) * (fp->ctf_typemax + 1));
948         }

950         if (fp->ctf_ptrtab != NULL) {
951                 ctf_free(fp->ctf_ptrtab,
952                     sizeof (ushort_t) * (fp->ctf_typemax + 1));
953         }

955         ctf_hash_destroy(&fp->ctf_structs);
956         ctf_hash_destroy(&fp->ctf_unions);
957         ctf_hash_destroy(&fp->ctf_enums);
958         ctf_hash_destroy(&fp->ctf_names);

960         ctf_free(fp, sizeof (ctf_file_t));
961 }

963 /*
964  * Return the CTF handle for the parent CTF container, if one exists.
965  * Otherwise return NULL to indicate this container has no imported parent.
966  */
967 ctf_file_t *
968 ctf_parent_file(ctf_file_t *fp)
969 {
970         return (fp->ctf_parent);
971 }

973 /*
974  * Return the name of the parent CTF container, if one exists.  Otherwise
975  * return NULL to indicate this container is a root container.
976  */
977 const char *
978 ctf_parent_name(ctf_file_t *fp)
979 {
980         return (fp->ctf_parname);
981 }

983 /*
984  * Import the types from the specified parent container by storing a pointer
985  * to it in ctf_parent and incrementing its reference count.  Only one parent
986  * is allowed: if a parent already exists, it is replaced by the new parent.
987  */
988 int
989 ctf_import(ctf_file_t *fp, ctf_file_t *pfp)
990 {
991         if (fp == NULL || fp == pfp || (pfp != NULL && pfp->ctf_refcnt == 0))
992                 return (ctf_set_errno(fp, EINVAL));

994         if (pfp != NULL && pfp->ctf_dmodel != fp->ctf_dmodel)
995                 return (ctf_set_errno(fp, ECTF_DMODEL));

997         if (fp->ctf_parent != NULL)
998                 ctf_close(fp->ctf_parent);

1000        if (pfp != NULL) {
1001                fp->ctf_flags |= LCTF_CHILD;
```

```
1002                 pfp->ctf_refcnt++;
1003         }

1005         fp->ctf_parent = pfp;
1006         return (0);
1007 }

1009 /*
1010  * Set the data model constant for the CTF container.
1011  */
1012 int
1013 ctf_setmodel(ctf_file_t *fp, int model)
1014 {
1015         const ctf_dmodel_t *dp;

1017         for (dp = _libctf_models; dp->ctd_name != NULL; dp++) {
1018                 if (dp->ctd_code == model) {
1019                         fp->ctf_dmodel = dp;
1020                         return (0);
1021                 }
1022         }

1024         return (ctf_set_errno(fp, EINVAL));
1025 }

1027 /*
1028  * Return the data model constant for the CTF container.
1029  */
1030 int
1031 ctf_getmodel(ctf_file_t *fp)
1032 {
1033         return (fp->ctf_dmodel->ctd_code);
1034 }

1036 void
1037 ctf_setspecific(ctf_file_t *fp, void *data)
1038 {
1039         fp->ctf_specific = data;
1040 }

1042 void *
1043 ctf_getspecific(ctf_file_t *fp)
1044 {
1045         return (fp->ctf_specific);
1046 }
```

```
**********************************************************
   23428 Tue Jan 14 16:48:49 2014
new/usr/src/common/ctf/ctf_types.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <ctf_impl.h>

ssize_t
ctf_get_ctt_size(const ctf_file_t *fp, const ctf_type_t *tp, ssize_t *sizep,
    ssize_t *incrementp)
{
        ssize_t size, increment;

        if (fp->ctf_version > CTF_VERSION_1 &&
            tp->ctt_size == CTF_LSIZE_SENT) {
                size = CTF_TYPE_LSIZE(tp);
                increment = sizeof (ctf_type_t);
        } else {
                size = tp->ctt_size;
                increment = sizeof (ctf_stype_t);
        }

        if (sizep)
                *sizep = size;
        if (incrementp)
                *incrementp = increment;

        return (size);
}
_____unchanged_portion_omitted_

/*
```



```
 * Lookup the given type ID and print a string name for it into buf.  Return
 * the actual number of bytes (not including \0) needed to format the name.
 */
static ssize_t
ctf_type_qlname(ctf_file_t *fp, ctf_id_t type, char *buf, size_t len,
    const char *qname)
ssize_t
ctf_type_lname(ctf_file_t *fp, ctf_id_t type, char *buf, size_t len)
{
        ctf_decl_t cd;
        ctf_decl_node_t *cdp;
        ctf_decl_prec_t prec, lp, rp;
        int ptr, arr;
        uint_t k;

        if (fp == NULL && type == CTF_ERR)
                return (-1); /* simplify caller code by permitting CTF_ERR */

        ctf_decl_init(&cd, buf, len);
        ctf_decl_push(&cd, fp, type);

        if (cd.cd_err != 0) {
                ctf_decl_fini(&cd);
                return (ctf_set_errno(fp, cd.cd_err));
        }

        /*
         * If the type graph's order conflicts with lexical precedence order
         * for pointers or arrays, then we need to surround the declarations at
         * the corresponding lexical precedence with parentheses.  This can
         * result in either a parenthesized pointer (*) as in int (*)() or
         * int (*)[], or in a parenthesized pointer and array as in int (*[])().
         */
        ptr = cd.cd_order[CTF_PREC_POINTER] > CTF_PREC_POINTER;
        arr = cd.cd_order[CTF_PREC_ARRAY] > CTF_PREC_ARRAY;

        rp = arr ? CTF_PREC_ARRAY : ptr ? CTF_PREC_POINTER : -1;
        lp = ptr ? CTF_PREC_POINTER : arr ? CTF_PREC_ARRAY : -1;

        k = CTF_K_POINTER; /* avoid leading whitespace (see below) */

        for (prec = CTF_PREC_BASE; prec < CTF_PREC_MAX; prec++) {
                for (cdp = ctf_list_next(&cd.cd_nodes[prec]);
                    cdp != NULL; cdp = ctf_list_next(cdp)) {

                        ctf_file_t *rfp = fp;
                        const ctf_type_t *tp =
                            ctf_lookup_by_id(&rfp, cdp->cd_type);
                        const char *name = ctf_strptr(rfp, tp->ctt_name);

                        if (k != CTF_K_POINTER && k != CTF_K_ARRAY)
                                ctf_decl_sprintf(&cd, " ");

                        if (lp == prec) {
                                ctf_decl_sprintf(&cd, "(");
                                lp = -1;
                        }

                        switch (cdp->cd_kind) {
                        case CTF_K_INTEGER:
                        case CTF_K_FLOAT:
                        case CTF_K_TYPEDEF:
                                if (qname != NULL)
                                        ctf_decl_sprintf(&cd, "%s`", qname);
#endif /* ! codereview */
                                ctf_decl_sprintf(&cd, "%s", name);
```

```
                                break;
                        case CTF_K_POINTER:
                                ctf_decl_sprintf(&cd, "*");
                                break;
                        case CTF_K_ARRAY:
                                ctf_decl_sprintf(&cd, "[%u]", cdp->cd_n);
                                break;
                        case CTF_K_FUNCTION:
                                ctf_decl_sprintf(&cd, "()");
                                break;
                        case CTF_K_STRUCT:
                        case CTF_K_FORWARD:
                                ctf_decl_sprintf(&cd, "struct ");
                                if (qname != NULL)
                                        ctf_decl_sprintf(&cd, "%s`", qname);
                                ctf_decl_sprintf(&cd, "%s", name);
                                ctf_decl_sprintf(&cd, "struct %s", name);
                                break;
                        case CTF_K_UNION:
                                ctf_decl_sprintf(&cd, "union ");
                                if (qname != NULL)
                                        ctf_decl_sprintf(&cd, "%s`", qname);
                                ctf_decl_sprintf(&cd, "%s", name);
                                ctf_decl_sprintf(&cd, "union %s", name);
                                break;
                        case CTF_K_ENUM:
                                ctf_decl_sprintf(&cd, "enum ");
                                if (qname != NULL)
                                        ctf_decl_sprintf(&cd, "%s`", qname);
                                ctf_decl_sprintf(&cd, "%s", name);
                                ctf_decl_sprintf(&cd, "enum %s", name);
                                break;
                        case CTF_K_VOLATILE:
                                ctf_decl_sprintf(&cd, "volatile");
                                break;
                        case CTF_K_CONST:
                                ctf_decl_sprintf(&cd, "const");
                                break;
                        case CTF_K_RESTRICT:
                                ctf_decl_sprintf(&cd, "restrict");
                                break;
                        }

                        k = cdp->cd_kind;
                }

                if (rp == prec)
                        ctf_decl_sprintf(&cd, ")");
        }

        if (cd.cd_len >= len)
                (void) ctf_set_errno(fp, ECTF_NAMELEN);

        ctf_decl_fini(&cd);
        return (cd.cd_len);
}

ssize_t
ctf_type_lname(ctf_file_t *fp, ctf_id_t type, char *buf, size_t len)
{
        return (ctf_type_qlname(fp, type, buf, len, NULL));
}

#endif /* ! codereview */
/*
 * Lookup the given type ID and print a string name for it into buf.  If buf
```



```
 * is too small, return NULL: the ECTF_NAMELEN error is set on 'fp' for us.
 */
char *
ctf_type_name(ctf_file_t *fp, ctf_id_t type, char *buf, size_t len)
{
        ssize_t rv = ctf_type_qlname(fp, type, buf, len, NULL);
        return (rv >= 0 && rv < len ? buf : NULL);
}

char *
ctf_type_qname(ctf_file_t *fp, ctf_id_t type, char *buf, size_t len,
    const char *qname)
{
        ssize_t rv = ctf_type_qlname(fp, type, buf, len, qname);
        ssize_t rv = ctf_type_lname(fp, type, buf, len);
        return (rv >= 0 && rv < len ? buf : NULL);
}


#endif /* ! codereview */
/*
 * Resolve the type down to a base type node, and then return the size
 * of the type storage in bytes.
 */
ssize_t
ctf_type_size(ctf_file_t *fp, ctf_id_t type)
{
        const ctf_type_t *tp;
        ssize_t size;
        ctf_arinfo_t ar;

        if ((type = ctf_type_resolve(fp, type)) == CTF_ERR)
                return (-1); /* errno is set for us */

        if ((tp = ctf_lookup_by_id(&fp, type)) == NULL)
                return (-1); /* errno is set for us */

        switch (LCTF_INFO_KIND(fp, tp->ctt_info)) {
        case CTF_K_POINTER:
                return (fp->ctf_dmodel->ctd_pointer);

        case CTF_K_FUNCTION:
                return (0); /* function size is only known by symtab */

        case CTF_K_ENUM:
                return (fp->ctf_dmodel->ctd_int);

        case CTF_K_ARRAY:
                /*
                 * Array size is not directly returned by stabs data.  Instead,
                 * it defines the element type and requires the user to perform
                 * the multiplication.  If ctf_get_ctt_size() returns zero, the
                 * current version of ctfconvert does not compute member sizes
                 * and we compute the size here on its behalf.
                 */
                if ((size = ctf_get_ctt_size(fp, tp, NULL, NULL)) > 0)
                        return (size);

                if (ctf_array_info(fp, type, &ar) == CTF_ERR ||
                    (size = ctf_type_size(fp, ar.ctr_contents)) == CTF_ERR)
                        return (-1); /* errno is set for us */

                return (size * ar.ctr_nelems);

        default:
                return (ctf_get_ctt_size(fp, tp, NULL, NULL));
```

```
389         }
390 }

392 /*
393  * Resolve the type down to a base type node, and then return the alignment
394  * needed for the type storage in bytes.
395  */
396 ssize_t
397 ctf_type_align(ctf_file_t *fp, ctf_id_t type)
398 {
399         const ctf_type_t *tp;
400         ctf_arinfo_t r;

402         if ((type = ctf_type_resolve(fp, type)) == CTF_ERR)
403                 return (-1); /* errno is set for us */

405         if ((tp = ctf_lookup_by_id(&fp, type)) == NULL)
406                 return (-1); /* errno is set for us */

408         switch (LCTF_INFO_KIND(fp, tp->ctt_info)) {
409         case CTF_K_POINTER:
410         case CTF_K_FUNCTION:
411                 return (fp->ctf_dmodel->ctd_pointer);

413         case CTF_K_ARRAY:
414                 if (ctf_array_info(fp, type, &r) == CTF_ERR)
415                         return (-1); /* errno is set for us */
416                 return (ctf_type_align(fp, r.ctr_contents));

418         case CTF_K_STRUCT:
419         case CTF_K_UNION: {
420                 uint_t n = LCTF_INFO_VLEN(fp, tp->ctt_info);
421                 ssize_t size, increment;
422                 size_t align = 0;
423                 const void *vmp;

425                 (void) ctf_get_ctt_size(fp, tp, &size, &increment);
426                 vmp = (uchar_t *)tp + increment;

428                 if (LCTF_INFO_KIND(fp, tp->ctt_info) == CTF_K_STRUCT)
429                         n = MIN(n, 1); /* only use first member for structs */

431                 if (fp->ctf_version == CTF_VERSION_1 ||
432                     size < CTF_LSTRUCT_THRESH) {
433                         const ctf_member_t *mp = vmp;
434                         for (; n != 0; n--, mp++) {
435                                 ssize_t am = ctf_type_align(fp, mp->ctm_type);
436                                 align = MAX(align, am);
437                         }
438                 } else {
439                         const ctf_lmember_t *lmp = vmp;
440                         for (; n != 0; n--, lmp++) {
441                                 ssize_t am = ctf_type_align(fp, lmp->ctlm_type);
442                                 align = MAX(align, am);
443                         }
444                 }

446                 return (align);
447         }

449         case CTF_K_ENUM:
450                 return (fp->ctf_dmodel->ctd_int);

452         default:
453                 return (ctf_get_ctt_size(fp, tp, NULL, NULL));
454         }
```



```
455 }

457 /*
458  * Return the kind (CTF_K_* constant) for the specified type ID.
459  */
460 int
461 ctf_type_kind(ctf_file_t *fp, ctf_id_t type)
462 {
463         const ctf_type_t *tp;

465         if ((tp = ctf_lookup_by_id(&fp, type)) == NULL)
466                 return (CTF_ERR); /* errno is set for us */

468         return (LCTF_INFO_KIND(fp, tp->ctt_info));
469 }

471 /*
472  * If the type is one that directly references another type (such as POINTER),
473  * then return the ID of the type to which it refers.
474  */
475 ctf_id_t
476 ctf_type_reference(ctf_file_t *fp, ctf_id_t type)
477 {
478         ctf_file_t *ofp = fp;
479         const ctf_type_t *tp;

481         if ((tp = ctf_lookup_by_id(&fp, type)) == NULL)
482                 return (CTF_ERR); /* errno is set for us */

484         switch (LCTF_INFO_KIND(fp, tp->ctt_info)) {
485         case CTF_K_POINTER:
486         case CTF_K_TYPEDEF:
487         case CTF_K_VOLATILE:
488         case CTF_K_CONST:
489         case CTF_K_RESTRICT:
490                 return (tp->ctt_type);
491         default:
492                 return (ctf_set_errno(ofp, ECTF_NOTREF));
493         }
494 }

496 /*
497  * Find a pointer to type by looking in fp->ctf_ptrtab.  If we can't find a
498  * pointer to the given type, see if we can compute a pointer to the type
499  * resulting from resolving the type down to its base type and use that
500  * instead.  This helps with cases where the CTF data includes "struct foo *"
501  * but not "foo_t *" and the user accesses "foo_t *" in the debugger.
502  */
503 ctf_id_t
504 ctf_type_pointer(ctf_file_t *fp, ctf_id_t type)
505 {
506         ctf_file_t *ofp = fp;
507         ctf_id_t ntype;

509         if (ctf_lookup_by_id(&fp, type) == NULL)
510                 return (CTF_ERR); /* errno is set for us */

512         if ((ntype = fp->ctf_ptrtab[CTF_TYPE_TO_INDEX(type)]) != 0)
513                 return (CTF_INDEX_TO_TYPE(ntype, (fp->ctf_flags & LCTF_CHILD)));

515         if ((type = ctf_type_resolve(fp, type)) == CTF_ERR)
516                 return (ctf_set_errno(ofp, ECTF_NOTYPE));

518         if (ctf_lookup_by_id(&fp, type) == NULL)
519                 return (ctf_set_errno(ofp, ECTF_NOTYPE));
```

```
521         if ((ntype = fp->ctf_ptrtab[CTF_TYPE_TO_INDEX(type)]) != 0)
522                 return (CTF_INDEX_TO_TYPE(ntype, (fp->ctf_flags & LCTF_CHILD)));

524         return (ctf_set_errno(ofp, ECTF_NOTYPE));
525 }

527 /*
528  * Return the encoding for the specified INTEGER or FLOAT.
529  */
530 int
531 ctf_type_encoding(ctf_file_t *fp, ctf_id_t type, ctf_encoding_t *ep)
532 {
533         ctf_file_t *ofp = fp;
534         const ctf_type_t *tp;
535         ssize_t increment;
536         uint_t data;

538         if ((tp = ctf_lookup_by_id(&fp, type)) == NULL)
539                 return (CTF_ERR); /* errno is set for us */

541         (void) ctf_get_ctt_size(fp, tp, NULL, &increment);

543         switch (LCTF_INFO_KIND(fp, tp->ctt_info)) {
544         case CTF_K_INTEGER:
545                 data = *(const uint_t *)((uintptr_t)tp + increment);
546                 ep->cte_format = CTF_INT_ENCODING(data);
547                 ep->cte_offset = CTF_INT_OFFSET(data);
548                 ep->cte_bits = CTF_INT_BITS(data);
549                 break;
550         case CTF_K_FLOAT:
551                 data = *(const uint_t *)((uintptr_t)tp + increment);
552                 ep->cte_format = CTF_FP_ENCODING(data);
553                 ep->cte_offset = CTF_FP_OFFSET(data);
554                 ep->cte_bits = CTF_FP_BITS(data);
555                 break;
556         default:
557                 return (ctf_set_errno(ofp, ECTF_NOTINTFP));
558         }

560         return (0);
561 }

563 int
564 ctf_type_cmp(ctf_file_t *lfp, ctf_id_t ltype, ctf_file_t *rfp, ctf_id_t rtype)
565 {
566         int rval;

568         if (ltype < rtype)
569                 rval = -1;
570         else if (ltype > rtype)
571                 rval = 1;
572         else
573                 rval = 0;

575         if (lfp == rfp)
576                 return (rval);

578         if (CTF_TYPE_ISPARENT(ltype) && lfp->ctf_parent != NULL)
579                 lfp = lfp->ctf_parent;

581         if (CTF_TYPE_ISPARENT(rtype) && rfp->ctf_parent != NULL)
582                 rfp = rfp->ctf_parent;

584         if (lfp < rfp)
585                 return (-1);
```



```
587         if (lfp > rfp)
588                 return (1);

590         return (rval);
591 }

593 /*
594  * Return a boolean value indicating if two types are compatible integers or
595  * floating-pointer values.  This function returns true if the two types are
596  * the same, or if they have the same ASCII name and encoding properties.
597  * This function could be extended to test for compatibility for other kinds.
598  */
599 int
600 ctf_type_compat(ctf_file_t *lfp, ctf_id_t ltype,
601     ctf_file_t *rfp, ctf_id_t rtype)
602 {
603         const ctf_type_t *ltp, *rtp;
604         ctf_encoding_t le, re;
605         ctf_arinfo_t la, ra;
606         uint_t lkind, rkind;

608         if (ctf_type_cmp(lfp, ltype, rfp, rtype) == 0)
609                 return (1);

611         ltype = ctf_type_resolve(lfp, ltype);
612         lkind = ctf_type_kind(lfp, ltype);

614         rtype = ctf_type_resolve(rfp, rtype);
615         rkind = ctf_type_kind(rfp, rtype);

617         if (lkind != rkind ||
618             (ltp = ctf_lookup_by_id(&lfp, ltype)) == NULL ||
619             (rtp = ctf_lookup_by_id(&rfp, rtype)) == NULL ||
620             strcmp(ctf_strptr(lfp, ltp->ctt_name),
621             ctf_strptr(rfp, rtp->ctt_name)) != 0)
622                 return (0);

624         switch (lkind) {
625         case CTF_K_INTEGER:
626         case CTF_K_FLOAT:
627                 return (ctf_type_encoding(lfp, ltype, &le) == 0 &&
628                     ctf_type_encoding(rfp, rtype, &re) == 0 &&
629                     bcmp(&le, &re, sizeof (ctf_encoding_t)) == 0);
630         case CTF_K_POINTER:
631                 return (ctf_type_compat(lfp, ctf_type_reference(lfp, ltype),
632                     rfp, ctf_type_reference(rfp, rtype)));
633         case CTF_K_ARRAY:
634                 return (ctf_array_info(lfp, ltype, &la) == 0 &&
635                     ctf_array_info(rfp, rtype, &ra) == 0 &&
636                     la.ctr_nelems == ra.ctr_nelems && ctf_type_compat(
637                     lfp, la.ctr_contents, rfp, ra.ctr_contents) &&
638                     ctf_type_compat(lfp, la.ctr_index, rfp, ra.ctr_index));
639         case CTF_K_STRUCT:
640         case CTF_K_UNION:
641                 return (ctf_type_size(lfp, ltype) == ctf_type_size(rfp, rtype));
642         case CTF_K_ENUM:
643         case CTF_K_FORWARD:
644                 return (1); /* no other checks required for these type kinds */
645         default:
646                 return (0); /* should not get here since we did a resolve */
647         }
648 }

650 /*
651  * Return the type and offset for a given member of a STRUCT or UNION.
652  */
```

```
653 int
654 ctf_member_info(ctf_file_t *fp, ctf_id_t type, const char *name,
655     ctf_membinfo_t *mip)
656 {
657         ctf_file_t *ofp = fp;
658         const ctf_type_t *tp;
659         ssize_t size, increment;
660         uint_t kind, n;
661 
662         if ((type = ctf_type_resolve(fp, type)) == CTF_ERR)
663                 return (CTF_ERR); /* errno is set for us */
664 
665         if ((tp = ctf_lookup_by_id(&fp, type)) == NULL)
666                 return (CTF_ERR); /* errno is set for us */
667 
668         (void) ctf_get_ctt_size(fp, tp, &size, &increment);
669         kind = LCTF_INFO_KIND(fp, tp->ctt_info);
670 
671         if (kind != CTF_K_STRUCT && kind != CTF_K_UNION)
672                 return (ctf_set_errno(ofp, ECTF_NOTSOU));
673 
674         if (fp->ctf_version == CTF_VERSION_1 || size < CTF_LSTRUCT_THRESH) {
675                 const ctf_member_t *mp = (const ctf_member_t *)
676                     ((uintptr_t)tp + increment);
677 
678                 for (n = LCTF_INFO_VLEN(fp, tp->ctt_info); n != 0; n--, mp++) {
679                         if (strcmp(ctf_strptr(fp, mp->ctm_name), name) == 0) {
680                                 mip->ctm_type = mp->ctm_type;
681                                 mip->ctm_offset = mp->ctm_offset;
682                                 return (0);
683                         }
684                 }
685         } else {
686                 const ctf_lmember_t *lmp = (const ctf_lmember_t *)
687                     ((uintptr_t)tp + increment);
688 
689                 for (n = LCTF_INFO_VLEN(fp, tp->ctt_info); n != 0; n--, lmp++) {
690                         if (strcmp(ctf_strptr(fp, lmp->ctlm_name), name) == 0) {
691                                 mip->ctm_type = lmp->ctlm_type;
692                                 mip->ctm_offset = (ulong_t)CTF_LMEM_OFFSET(lmp);
693                                 return (0);
694                         }
695                 }
696         }
697 
698         return (ctf_set_errno(ofp, ECTF_NOMEMBNAM));
699 }
700 
701 /*
702  * Return the array type, index, and size information for the specified ARRAY.
703  */
704 int
705 ctf_array_info(ctf_file_t *fp, ctf_id_t type, ctf_arinfo_t *arp)
706 {
707         ctf_file_t *ofp = fp;
708         const ctf_type_t *tp;
709         const ctf_array_t *ap;
710         ssize_t increment;
711 
712         if ((tp = ctf_lookup_by_id(&fp, type)) == NULL)
713                 return (CTF_ERR); /* errno is set for us */
714 
715         if (LCTF_INFO_KIND(fp, tp->ctt_info) != CTF_K_ARRAY)
716                 return (ctf_set_errno(ofp, ECTF_NOTARRAY));
717 
718         (void) ctf_get_ctt_size(fp, tp, NULL, &increment);
```



```
720         ap = (const ctf_array_t *)((uintptr_t)tp + increment);
721         arp->ctr_contents = ap->cta_contents;
722         arp->ctr_index = ap->cta_index;
723         arp->ctr_nelems = ap->cta_nelems;
724 
725         return (0);
726 }
727 
728 /*
729  * Convert the specified value to the corresponding enum member name, if a
730  * matching name can be found.  Otherwise NULL is returned.
731  */
732 const char *
733 ctf_enum_name(ctf_file_t *fp, ctf_id_t type, int value)
734 {
735         ctf_file_t *ofp = fp;
736         const ctf_type_t *tp;
737         const ctf_enum_t *ep;
738         ssize_t increment;
739         uint_t n;
740 
741         if ((type = ctf_type_resolve(fp, type)) == CTF_ERR)
742                 return (NULL); /* errno is set for us */
743 
744         if ((tp = ctf_lookup_by_id(&fp, type)) == NULL)
745                 return (NULL); /* errno is set for us */
746 
747         if (LCTF_INFO_KIND(fp, tp->ctt_info) != CTF_K_ENUM) {
748                 (void) ctf_set_errno(ofp, ECTF_NOTENUM);
749                 return (NULL);
750         }
751 
752         (void) ctf_get_ctt_size(fp, tp, NULL, &increment);
753 
754         ep = (const ctf_enum_t *)((uintptr_t)tp + increment);
755 
756         for (n = LCTF_INFO_VLEN(fp, tp->ctt_info); n != 0; n--, ep++) {
757                 if (ep->cte_value == value)
758                         return (ctf_strptr(fp, ep->cte_name));
759         }
760 
761         (void) ctf_set_errno(ofp, ECTF_NOENUMNAM);
762         return (NULL);
763 }
764 
765 /*
766  * Convert the specified enum tag name to the corresponding value, if a
767  * matching name can be found.  Otherwise CTF_ERR is returned.
768  */
769 int
770 ctf_enum_value(ctf_file_t *fp, ctf_id_t type, const char *name, int *valp)
771 {
772         ctf_file_t *ofp = fp;
773         const ctf_type_t *tp;
774         const ctf_enum_t *ep;
775         ssize_t size, increment;
776         uint_t n;
777 
778         if ((type = ctf_type_resolve(fp, type)) == CTF_ERR)
779                 return (CTF_ERR); /* errno is set for us */
780 
781         if ((tp = ctf_lookup_by_id(&fp, type)) == NULL)
782                 return (CTF_ERR); /* errno is set for us */
783 
784         if (LCTF_INFO_KIND(fp, tp->ctt_info) != CTF_K_ENUM) {
```

```
785                 (void) ctf_set_errno(ofp, ECTF_NOTENUM);
786                 return (CTF_ERR);
787         }

789         (void) ctf_get_ctt_size(fp, tp, &size, &increment);

791         ep = (const ctf_enum_t *)((uintptr_t)tp + increment);

793         for (n = LCTF_INFO_VLEN(fp, tp->ctt_info); n != 0; n--, ep++) {
794                 if (strcmp(ctf_strptr(fp, ep->cte_name), name) == 0) {
795                         if (valp != NULL)
796                                 *valp = ep->cte_value;
797                         return (0);
798                 }
799         }

801         (void) ctf_set_errno(ofp, ECTF_NOENUMNAM);
802         return (CTF_ERR);
803 }

805 /*
806  * Recursively visit the members of any type.  This function is used as the
807  * engine for ctf_type_visit, below.  We resolve the input type, recursively
808  * invoke ourself for each type member if the type is a struct or union, and
809  * then invoke the callback function on the current type.  If any callback
810  * returns non-zero, we abort and percolate the error code back up to the top.
811  */
812 static int
813 ctf_type_rvisit(ctf_file_t *fp, ctf_id_t type, ctf_visit_f *func, void *arg,
814     const char *name, ulong_t offset, int depth)
815 {
816         ctf_id_t otype = type;
817         const ctf_type_t *tp;
818         ssize_t size, increment;
819         uint_t kind, n;
820         int rc;

822         if ((type = ctf_type_resolve(fp, type)) == CTF_ERR)
823                 return (CTF_ERR); /* errno is set for us */

825         if ((tp = ctf_lookup_by_id(&fp, type)) == NULL)
826                 return (CTF_ERR); /* errno is set for us */

828         if ((rc = func(name, otype, offset, depth, arg)) != 0)
829                 return (rc);

831         kind = LCTF_INFO_KIND(fp, tp->ctt_info);

833         if (kind != CTF_K_STRUCT && kind != CTF_K_UNION)
834                 return (0);

836         (void) ctf_get_ctt_size(fp, tp, &size, &increment);

838         if (fp->ctf_version == CTF_VERSION_1 || size < CTF_LSTRUCT_THRESH) {
839                 const ctf_member_t *mp = (const ctf_member_t *)
840                     ((uintptr_t)tp + increment);

842                 for (n = LCTF_INFO_VLEN(fp, tp->ctt_info); n != 0; n--, mp++) {
843                         if ((rc = ctf_type_rvisit(fp, mp->ctm_type,
844                             func, arg, ctf_strptr(fp, mp->ctm_name),
845                             offset + mp->ctm_offset, depth + 1)) != 0)
846                                 return (rc);
847                 }

849         } else {
850                 const ctf_lmember_t *lmp = (const ctf_lmember_t *)
```



```
851                     ((uintptr_t)tp + increment);

853                 for (n = LCTF_INFO_VLEN(fp, tp->ctt_info); n != 0; n--, lmp++) {
854                         if ((rc = ctf_type_rvisit(fp, lmp->ctlm_type,
855                             func, arg, ctf_strptr(fp, lmp->ctlm_name),
856                             offset + (ulong_t)CTF_LMEM_OFFSET(lmp),
857                             depth + 1)) != 0)
858                                 return (rc);
859                 }
860         }

862         return (0);
863 }

865 /*
866  * Recursively visit the members of any type.  We pass the name, member
867  * type, and offset of each member to the specified callback function.
868  */
869 int
870 ctf_type_visit(ctf_file_t *fp, ctf_id_t type, ctf_visit_f *func, void *arg)
871 {
872         return (ctf_type_rvisit(fp, type, func, arg, "", 0, 0));
873 }
```

```
**********************************************************
    2634 Tue Jan 14 16:48:50 2014
new/usr/src/lib/libctf/common/mapfile-vers
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 (c) 2013, Joyent, Inc.  All rights reserved.
#

#
#endif /* ! codereview */
# 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

# There really should be only one SUNWprivate version.
# Don't add any more.  Add new private symbols to SUNWprivate_1.2

SYMBOL_VERSION SUNWprivate_1.2 {
    global:
        ctf_add_array;
        ctf_add_const;
        ctf_add_enum;
        ctf_add_enumerator;
        ctf_add_float;
        ctf_add_forward;
```



```
        ctf_add_function;
        ctf_add_integer;
        ctf_add_member;
        ctf_add_pointer;
        ctf_add_restrict;
        ctf_add_struct;
        ctf_add_type;
        ctf_add_typedef;
        ctf_add_union;
        ctf_add_volatile;
        ctf_create;
        ctf_delete_type;
        ctf_discard;
        ctf_dup;
#endif /* ! codereview */
        ctf_enum_value;
        ctf_label_info;
        ctf_label_iter;
        ctf_label_topmost;
        ctf_member_info;
        ctf_parent_file;
        ctf_parent_name;
        ctf_set_array;
        ctf_type_align;
        ctf_type_cmp;
        ctf_type_compat;
        ctf_type_pointer;
        ctf_update;
        ctf_write;
} SUNWprivate_1.1;

SYMBOL_VERSION SUNWprivate_1.1 {
    global:
        ctf_array_info;
        ctf_bufopen;
        ctf_close;
        ctf_enum_iter;
        ctf_enum_name;
        ctf_errmsg;
        ctf_errno;
        ctf_fdopen;
        ctf_func_args;
        ctf_func_info;
        ctf_getmodel;
        ctf_getspecific;
        ctf_import;
        ctf_lookup_by_name;
        ctf_lookup_by_symbol;
        ctf_member_iter;
        ctf_open;
        ctf_setmodel;
        ctf_setspecific;
        ctf_type_encoding;
        ctf_type_iter;
        ctf_type_kind;
        ctf_type_lname;
        ctf_type_name;
        ctf_type_qname;
#endif /* ! codereview */
        ctf_type_reference;
        ctf_type_resolve;
        ctf_type_size;
        ctf_type_visit;
        ctf_version;
        _libctf_debug;
    local:
```

```
123         *;
124 };
```

```
**********************************************************
   13814 Tue Jan 14 16:48:50 2014
new/usr/src/lib/libdtrace/common/dt_as.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 * Copyright (c) 2013 Joyent, Inc. All rights reserved.
 */

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

#include <sys/types.h>
#include <strings.h>
#include <stdlib.h>
#include <assert.h>

#include <dt_impl.h>
#include <dt_parser.h>
#include <dt_as.h>

void
dt_irlist_create(dt_irlist_t *dlp)
{
        bzero(dlp, sizeof (dt_irlist_t));
        dlp->dl_label = 1;
}
_____unchanged_portion_omitted_

/*ARGSUSED*/
static int
dt_copyvar(dt_idhash_t *dhp, dt_ident_t *idp, void *data)
{
        dt_pcb_t *pcb = data;
        dtrace_difv_t *dvp;
        ssize_t stroff;
```



```
        dt_node_t dn;

        if (!(idp->di_flags & (DT_IDFLG_DIFR | DT_IDFLG_DIFW)))
                return (0); /* omit variable from vartab */

        dvp = &pcb->pcb_difo->dtdo_vartab[pcb->pcb_asvidx++];
        stroff = dt_strtab_insert(pcb->pcb_strtab, idp->di_name);

        if (stroff == -1L)
                longjmp(pcb->pcb_jmpbuf, EDT_NOMEM);
        if (stroff > DIF_STROFF_MAX)
                longjmp(pcb->pcb_jmpbuf, EDT_STR2BIG);

        dvp->dtdv_name = (uint_t)stroff;
        dvp->dtdv_id = idp->di_id;
        dvp->dtdv_flags = 0;

        dvp->dtdv_kind = (idp->di_kind == DT_IDENT_ARRAY) ?
            DIFV_KIND_ARRAY : DIFV_KIND_SCALAR;

        if (idp->di_flags & DT_IDFLG_LOCAL)
                dvp->dtdv_scope = DIFV_SCOPE_LOCAL;
        else if (idp->di_flags & DT_IDFLG_TLS)
                dvp->dtdv_scope = DIFV_SCOPE_THREAD;
        else
                dvp->dtdv_scope = DIFV_SCOPE_GLOBAL;

        if (idp->di_flags & DT_IDFLG_DIFR)
                dvp->dtdv_flags |= DIFV_F_REF;
        if (idp->di_flags & DT_IDFLG_DIFW)
                dvp->dtdv_flags |= DIFV_F_MOD;

        bzero(&dn, sizeof (dn));
        dt_node_type_assign(&dn, idp->di_ctfp, idp->di_type, B_FALSE);
        dt_node_type_assign(&dn, idp->di_ctfp, idp->di_type);
        dt_node_diftype(pcb->pcb_hdl, &dn, &dvp->dtdv_type);

        idp->di_flags &= ~(DT_IDFLG_DIFR | DT_IDFLG_DIFW);
        return (0);
}
_____unchanged_portion_omitted_
```

```
**********************************************************
   70859 Tue Jan 14 16:48:51 2014
new/usr/src/lib/libdtrace/common/dt_cc.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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, Joyent Inc. All rights reserved.
 * Copyright (c) 2011, Joyent Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

/*
 * DTrace D Language Compiler
 *
 * The code in this source file implements the main engine for the D language
 * compiler.  The driver routine for the compiler is dt_compile(), below.  The
 * compiler operates on either stdio FILEs or in-memory strings as its input
 * and can produce either dtrace_prog_t structures from a D program or a single
 * dtrace_difo_t structure from a D expression.  Multiple entry points are
 * provided as wrappers around dt_compile() for the various input/output pairs.
 * The compiler itself is implemented across the following source files:
 *
 * dt_lex.l - lex scanner
 * dt_grammar.y - yacc grammar
 * dt_parser.c - parse tree creation and semantic checking
 * dt_decl.c - declaration stack processing
 * dt_xlator.c - D translator lookup and creation
 * dt_ident.c - identifier and symbol table routines
 * dt_pragma.c - #pragma processing and D pragmas
 * dt_printf.c - D printf() and printa() argument checking and processing
 * dt_cc.c - compiler driver and dtrace_prog_t construction
 * dt_cg.c - DIF code generator
 * dt_as.c - DIF assembler
 * dt_dof.c - dtrace_prog_t -> DOF conversion
 *
 * Several other source files provide collections of utility routines used by
 * these major files.  The compiler itself is implemented in multiple passes:
 *
 * (1) The input program is scanned and parsed by dt_lex.l and dt_grammar.y
```



```
 *     and parse tree nodes are constructed using the routines in dt_parser.c.
 *     This node construction pass is described further in dt_parser.c.
 *
 * (2) The parse tree is "cooked" by assigning each clause a context (see the
 *     routine dt_setcontext(), below) based on its probe description and then
 *     recursively descending the tree performing semantic checking.  The cook
 *     routines are also implemented in dt_parser.c and described there.
 *
 * (3) For actions that are DIF expression statements, the DIF code generator
 *     and assembler are invoked to create a finished DIFO for the statement.
 *
 * (4) The dtrace_prog_t data structures for the program clauses and actions
 *     are built, containing pointers to any DIFOs created in step (3).
 *
 * (5) The caller invokes a routine in dt_dof.c to convert the finished program
 *     into DOF format for use in anonymous tracing or enabling in the kernel.
 *
 * In the implementation, steps 2-4 are intertwined in that they are performed
 * in order for each clause as part of a loop that executes over the clauses.
 *
 * The D compiler currently implements nearly no optimization.  The compiler
 * implements integer constant folding as part of pass (1), and a set of very
 * simple peephole optimizations as part of pass (3).  As with any C compiler,
 * a large number of optimizations are possible on both the intermediate data
 * structures and the generated DIF code.  These possibilities should be
 * investigated in the context of whether they will have any substantive effect
 * on the overall DTrace probe effect before they are undertaken.
 */

#include <sys/types.h>
#include <sys/wait.h>
#include <sys/sysmacros.h>

#include <assert.h>
#include <strings.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <ucontext.h>
#include <limits.h>
#include <ctype.h>
#include <dirent.h>
#include <dt_module.h>
#include <dt_program.h>
#include <dt_provider.h>
#include <dt_printf.h>
#include <dt_pid.h>
#include <dt_grammar.h>
#include <dt_ident.h>
#include <dt_string.h>
#include <dt_impl.h>

static const dtrace_diftype_t dt_void_rtype = {
        DIF_TYPE_CTF, CTF_K_INTEGER, 0, 0, 0
};
```

*unchanged_portion_omitted_*

```
static void
dt_action_trace(dtrace_hdl_t *dtp, dt_node_t *dnp, dtrace_stmtdesc_t *sdp)
{
        int ctflib;

#endif /* ! codereview */
        dtrace_actdesc_t *ap = dt_stmt_action(dtp, sdp);
```

```
669         boolean_t istrace = (dnp->dn_ident->di_id == DT_ACT_TRACE);
670         const char *act = istrace ?  "trace" : "print";

672         if (dt_node_is_void(dnp->dn_args)) {
673                 dnerror(dnp->dn_args, istrace ? D_TRACE_VOID : D_PRINT_VOID,
674                     "%s( ) may not be applied to a void expression\n", act);
675         }

677         if (dt_node_resolve(dnp->dn_args, DT_IDENT_XLPTR) != NULL) {
678                 dnerror(dnp->dn_args, istrace ? D_TRACE_DYN : D_PRINT_DYN,
679                     "%s( ) may not be applied to a translated pointer\n", act);
680         }

682         if (dnp->dn_args->dn_kind == DT_NODE_AGG) {
683                 dnerror(dnp->dn_args, istrace ? D_TRACE_AGG : D_PRINT_AGG,
684                     "%s( ) may not be applied to an aggregation%s\n", act,
685                     istrace ? "" : " -- did you mean printa()?");
686         }

688         dt_cg(yypcb, dnp->dn_args);

690         /*
691          * The print() action behaves identically to trace(), except that it
692          * stores the CTF type of the argument (if present) within the DOF for
693          * the DIFEXPR action.  To do this, we set the 'dtsd_strdata' to point
694          * to the fully-qualified CTF type ID for the result of the DIF
695          * action.  We use the ID instead of the name to handles complex types
696          * like arrays and function pointers that can't be resolved by
697          * ctf_type_lookup().  This is later processed by dtrace_dof_create()
698          * and turned into a reference into the string table so that we can
699          * get the type information when we process the data after the fact.  In
700          * the case where we are referring to userland CTF data, we also need to
701          * to identify which ctf container in question we care about and encode
702          * that within the name.
665          * get the type information when we process the data after the fact.
703          */
704         if (dnp->dn_ident->di_id == DT_ACT_PRINT) {
705                 dt_node_t *dret;
706                 size_t n;
707                 dt_module_t *dmp;

709                 dret = yypcb->pcb_dret;
710                 dmp = dt_module_lookup_by_ctf(dtp, dret->dn_ctfp);

712                 if (dmp->dm_pid != 0) {
713                         ctflib = dt_module_getlibid(dtp, dmp, dret->dn_ctfp);
714                         assert(ctflib >= 0);
715                         n = snprintf(NULL, 0, "%s`%d`%d", dmp->dm_name,
716                             ctflib, dret->dn_type) + 1;
717                 } else {
718                         n = snprintf(NULL, 0, "%s`%d", dmp->dm_name,
719                             dret->dn_type) + 1;
720                 }
675                 n = snprintf(NULL, 0, "%s`%d", dmp->dm_name, dret->dn_type) + 1;
721                 sdp->dtsd_strdata = dt_alloc(dtp, n);
722                 if (sdp->dtsd_strdata == NULL)
723                         longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
724                 if (dmp->dm_pid != 0) {
725                         (void) snprintf(sdp->dtsd_strdata, n, "%s`%d`%d",
726                             dmp->dm_name, ctflib, dret->dn_type);
727                 } else {
728                         (void) snprintf(sdp->dtsd_strdata, n, "%s`%d",
729                             dmp->dm_name, dret->dn_type);
730                 }
679                 (void) snprintf(sdp->dtsd_strdata, n, "%s`%d", dmp->dm_name,
680                     dret->dn_type);
```



```
731         }

733         ap->dtad_difo = dt_as(yypcb);
734         ap->dtad_kind = DTRACEACT_DIFEXPR;
735 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   31338 Tue Jan 14 16:48:51 2014
new/usr/src/lib/libdtrace/common/dt_decl.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 * Copyright (c) 2013 Joyent, Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

#include <strings.h>
#include <stdlib.h>
#include <limits.h>
#include <alloca.h>
#include <assert.h>

#include <dt_decl.h>
#include <dt_parser.h>
#include <dt_module.h>
#include <dt_impl.h>

static dt_decl_t *
dt_decl_check(dt_decl_t *ddp)
{
        if (ddp->dd_kind == CTF_K_UNKNOWN)
                return (ddp); /* nothing to check if the type is not yet set */

        if (ddp->dd_name != NULL && strcmp(ddp->dd_name, "char") == 0 &&
            (ddp->dd_attr & (DT_DA_SHORT | DT_DA_LONG | DT_DA_LONGLONG))) {
                xyerror(D_DECL_CHARATTR, "invalid type declaration: short and "
                    "long may not be used with char type\n");
        }

        if (ddp->dd_name != NULL && strcmp(ddp->dd_name, "void") == 0 &&
            (ddp->dd_attr & (DT_DA_SHORT | DT_DA_LONG | DT_DA_LONGLONG |
            (DT_DA_SIGNED | DT_DA_UNSIGNED)))) {
                xyerror(D_DECL_VOIDATTR, "invalid type declaration: attributes "
                    "may not be used with void type\n");
        }
```



```
        if (ddp->dd_kind != CTF_K_INTEGER &&
            (ddp->dd_attr & (DT_DA_SIGNED | DT_DA_UNSIGNED))) {
                xyerror(D_DECL_SIGNINT, "invalid type declaration: signed and "
                    "unsigned may only be used with integer type\n");
        }

        if (ddp->dd_kind != CTF_K_INTEGER && ddp->dd_kind != CTF_K_FLOAT &&
            (ddp->dd_attr & (DT_DA_LONG | DT_DA_LONGLONG))) {
                xyerror(D_DECL_LONGINT, "invalid type declaration: long and "
                    "long long may only be used with integer or "
                    "floating-point type\n");
        }

        return (ddp);
}
```

*unchanged_portion_omitted_*

```
void
dt_decl_enumerator(char *s, dt_node_t *dnp)
{
        dt_scope_t *dsp = yypcb->pcb_dstack.ds_next;
        dtrace_hdl_t *dtp = yypcb->pcb_hdl;

        dt_idnode_t *inp;
        dt_ident_t *idp;
        char *name;
        int value;

        name = strdupa(s);
        free(s);

        if (dsp == NULL)
                longjmp(yypcb->pcb_jmpbuf, EDT_NOSCOPE);

        assert(dsp->ds_decl->dd_kind == CTF_K_ENUM);
        value = dsp->ds_enumval + 1; /* default is previous value plus one */

        if (strchr(name, '`') != NULL) {
                xyerror(D_DECL_SCOPE, "D scoping operator may not be used in "
                    "an enumerator name (%s)\n", name);
        }

        /*
         * If the enumerator is being assigned a value, cook and check the node
         * and then free it after we get the value.  We also permit references
         * to identifiers which are previously defined enumerators in the type.
         */
        if (dnp != NULL) {
                if (dnp->dn_kind != DT_NODE_IDENT || ctf_enum_value(
                    dsp->ds_ctfp, dsp->ds_type, dnp->dn_string, &value) != 0) {
                        dnp = dt_node_cook(dnp, DT_IDFLG_REF);

                        if (dnp->dn_kind != DT_NODE_INT) {
                                xyerror(D_DECL_ENCONST, "enumerator '%s' must "
                                    "be assigned to an integral constant "
                                    "expression\n", name);
                        }

                        if ((intmax_t)dnp->dn_value > INT_MAX ||
                            (intmax_t)dnp->dn_value < INT_MIN) {
                                xyerror(D_DECL_ENOFLOW, "enumerator '%s' value "
                                    "overflows INT_MAX (%d)\n", name, INT_MAX);
                        }
```

```
736                         value = (int)dnp->dn_value;
737                 }
738                 dt_node_free(dnp);
739         }

741         if (ctf_add_enumerator(dsp->ds_ctfp, dsp->ds_type,
742             name, value) == CTF_ERR || ctf_update(dsp->ds_ctfp) == CTF_ERR) {
743                 xyerror(D_UNKNOWN, "failed to define enumerator '%s': %s\n",
744                     name, ctf_errmsg(ctf_errno(dsp->ds_ctfp)));
745         }

747         dsp->ds_enumval = value; /* save most recent value */

749         /*
750          * If the enumerator name matches an identifier in the global scope,
751          * flag this as an error.  We only do this for "D" enumerators to
752          * prevent "C" header file enumerators from conflicting with the ever-
753          * growing list of D built-in global variables and inlines.  If a "C"
754          * enumerator conflicts with a global identifier, we add the enumerator
755          * but do not insert a corresponding inline (i.e. the D variable wins).
756          */
757         if (dt_idstack_lookup(&yypcb->pcb_globals, name) != NULL) {
758                 if (dsp->ds_ctfp == dtp->dt_ddefs->dm_ctfp) {
759                         xyerror(D_DECL_IDRED,
760                             "identifier redeclared: %s\n", name);
761                 } else
762                         return;
763         }

765         dt_dprintf("add global enumerator %s = %d\n", name, value);

767         idp = dt_idhash_insert(dtp->dt_globals, name, DT_IDENT_ENUM,
768             DT_IDFLG_INLINE | DT_IDFLG_REF, 0, _dtrace_defattr, 0,
769             &dt_idops_inline, NULL, dtp->dt_gen);

771         if (idp == NULL)
772                 longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

774         yyintprefix = 0;
775         yyintsuffix[0] = '\0';
776         yyintdecimal = 0;

778         dnp = dt_node_int(value);
779         dt_node_type_assign(dnp, dsp->ds_ctfp, dsp->ds_type, B_FALSE);
778         dt_node_type_assign(dnp, dsp->ds_ctfp, dsp->ds_type);

781         if ((inp = malloc(sizeof (dt_idnode_t))) == NULL)
782                 longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

784         /*
785          * Remove the INT node from the node allocation list and store it in
786          * din_list and din_root so it persists with and is freed by the ident.
787          */
788         assert(yypcb->pcb_list == dnp);
789         yypcb->pcb_list = dnp->dn_link;
790         dnp->dn_link = NULL;

792         bzero(inp, sizeof (dt_idnode_t));
793         inp->din_list = dnp;
794         inp->din_root = dnp;

796         idp->di_iarg = inp;
797         idp->di_ctfp = dsp->ds_ctfp;
798         idp->di_type = dsp->ds_type;
799 }
```



```
801 /*
802  * Look up the type corresponding to the specified decl stack.  The scoping of
803  * the underlying type names is handled by dt_type_lookup().  We build up the
804  * name from the specified string and prefixes and then lookup the type.  If
805  * we fail, an errmsg is saved and the caller must abort with EDT_COMPILER.
806  */
807 int
808 dt_decl_type(dt_decl_t *ddp, dtrace_typeinfo_t *tip)
809 {
810         dtrace_hdl_t *dtp = yypcb->pcb_hdl;

812         dt_module_t *dmp;
813         ctf_arinfo_t r;
814         ctf_id_t type;

816         char n[DT_TYPE_NAMELEN];
817         uint_t flag;
818         char *name;
819         int rv;

821         tip->dtt_flags = 0;

823 #endif /* ! codereview */
824         /*
825          * Based on our current #include depth and decl stack depth, determine
826          * which dynamic CTF module and scope to use when adding any new types.
827          */
828         dmp = yypcb->pcb_idepth ? dtp->dt_cdefs : dtp->dt_ddefs;
829         flag = yypcb->pcb_dstack.ds_next ? CTF_ADD_NONROOT : CTF_ADD_ROOT;

831         if (ddp->dd_attr & DT_DA_USER)
832                 tip->dtt_flags = DTT_FL_USER;

834 #endif /* ! codereview */
835         /*
836          * If we have already cached a CTF type for this decl, then we just
837          * return the type information for the cached type.
838          */
839         if (ddp->dd_ctfp != NULL &&
840             (dmp = dt_module_lookup_by_ctf(dtp, ddp->dd_ctfp)) != NULL) {
841                 tip->dtt_object = dmp->dm_name;
842                 tip->dtt_ctfp = ddp->dd_ctfp;
843                 tip->dtt_type = ddp->dd_type;
844                 return (0);
845         }

847         /*
848          * Currently CTF treats all function pointers identically.  We cache a
849          * representative ID of kind CTF_K_FUNCTION and just return that type.
850          * If we want to support full function declarations, dd_next refers to
851          * the declaration of the function return type, and the parameter list
852          * should be parsed and hung off a new pointer inside of this decl.
853          */
854         if (ddp->dd_kind == CTF_K_FUNCTION) {
855                 tip->dtt_object = dtp->dt_ddefs->dm_name;
856                 tip->dtt_ctfp = DT_FUNC_CTFP(dtp);
857                 tip->dtt_type = DT_FUNC_TYPE(dtp);
858                 return (0);
859         }

861         /*
862          * If the decl is a pointer, resolve the rest of the stack by calling
863          * dt_decl_type() recursively and then compute a pointer to the result.
864          * Similar to the code above, we return a cached id for function ptrs.
865          */
866         if (ddp->dd_kind == CTF_K_POINTER) {
```

```
867                 if (ddp->dd_next->dd_kind == CTF_K_FUNCTION) {
868                         tip->dtt_object = dtp->dt_ddefs->dm_name;
869                         tip->dtt_ctfp = DT_FPTR_CTFP(dtp);
870                         tip->dtt_type = DT_FPTR_TYPE(dtp);
871                         return (0);
872                 }

874                 if ((rv = dt_decl_type(ddp->dd_next, tip)) == 0 &&
875                     (rv = dt_type_pointer(tip)) != 0) {
876                         xywarn(D_UNKNOWN, "cannot find type: %s*: %s\n",
877                             dt_type_name(tip->dtt_ctfp, tip->dtt_type,
878                             n, sizeof (n)), ctf_errmsg(dtp->dt_ctferr));
879                 }

881                 return (rv);
882         }

884         /*
885          * If the decl is an array, we must find the base type and then call
886          * dt_decl_type() recursively and then build an array of the result.
887          * The C and D multi-dimensional array syntax requires that consecutive
888          * array declarations be processed from right-to-left (i.e. top-down
889          * from the perspective of the declaration stack).  For example, an
890          * array declaration such as int x[3][5] is stored on the stack as:
891          *
892          * (bottom) NULL <- ( INT "int" ) <- ( ARR [3] ) <- ( ARR [5] ) (top)
893          *
894          * but means that x is declared to be an array of 3 objects each of
895          * which is an array of 5 integers, or in CTF representation:
896          *
897          * type T1:( content=int, nelems=5 ) type T2:( content=T1, nelems=3 )
898          *
899          * For more details, refer to K&R[5.7] and ISO C 6.5.2.1.  Rather than
900          * overcomplicate the implementation of dt_decl_type(), we push array
901          * declarations down into the stack in dt_decl_array(), above, so that
902          * by the time dt_decl_type() is called, the decl stack looks like:
903          *
904          * (bottom) NULL <- ( INT "int" ) <- ( ARR [5] ) <- ( ARR [3] ) (top)
905          *
906          * which permits a straightforward recursive descent of the decl stack
907          * to build the corresponding CTF type tree in the appropriate order.
908          */
909         if (ddp->dd_kind == CTF_K_ARRAY) {
910                 /*
911                  * If the array decl has a parameter list associated with it,
912                  * this is an associative array declaration: return <DYN>.
913                  */
914                 if (ddp->dd_node != NULL &&
915                     ddp->dd_node->dn_kind == DT_NODE_TYPE) {
916                         tip->dtt_object = dtp->dt_ddefs->dm_name;
917                         tip->dtt_ctfp = DT_DYN_CTFP(dtp);
918                         tip->dtt_type = DT_DYN_TYPE(dtp);
919                         return (0);
920                 }

922                 if ((rv = dt_decl_type(ddp->dd_next, tip)) != 0)
923                         return (rv);

925                 /*
926                  * If the array base type is not defined in the target
927                  * container or its parent, copy the type to the target
928                  * container and reset dtt_ctfp and dtt_type to the copy.
929                  */
930                 if (tip->dtt_ctfp != dmp->dm_ctfp &&
931                     tip->dtt_ctfp != ctf_parent_file(dmp->dm_ctfp)) {
```



```
933                         tip->dtt_type = ctf_add_type(dmp->dm_ctfp,
934                             tip->dtt_ctfp, tip->dtt_type);
935                         tip->dtt_ctfp = dmp->dm_ctfp;

937                         if (tip->dtt_type == CTF_ERR ||
938                             ctf_update(tip->dtt_ctfp) == CTF_ERR) {
939                                 xywarn(D_UNKNOWN, "failed to copy type: %s\n",
940                                     ctf_errmsg(ctf_errno(tip->dtt_ctfp)));
941                                 return (-1);
942                         }
943                 }

945                 /*
946                  * The array index type is irrelevant in C and D: just set it
947                  * to "long" for all array types that we create on-the-fly.
948                  */
949                 r.ctr_contents = tip->dtt_type;
950                 r.ctr_index = ctf_lookup_by_name(tip->dtt_ctfp, "long");
951                 r.ctr_nelems = ddp->dd_node ?
952                     (uint_t)ddp->dd_node->dn_value : 0;

954                 tip->dtt_object = dmp->dm_name;
955                 tip->dtt_ctfp = dmp->dm_ctfp;
956                 tip->dtt_type = ctf_add_array(dmp->dm_ctfp, CTF_ADD_ROOT, &r);

958                 if (tip->dtt_type == CTF_ERR ||
959                     ctf_update(tip->dtt_ctfp) == CTF_ERR) {
960                         xywarn(D_UNKNOWN, "failed to create array type: %s\n",
961                             ctf_errmsg(ctf_errno(tip->dtt_ctfp)));
962                         return (-1);
963                 }

965                 return (0);
966         }

968         /*
969          * Allocate space for the type name and enough space for the maximum
970          * additional text ("unsigned long long \0" requires 20 more bytes).
971          */
972         name = alloca(ddp->dd_name ? strlen(ddp->dd_name) + 20 : 20);
973         name[0] = '\0';

975         switch (ddp->dd_kind) {
976         case CTF_K_INTEGER:
977         case CTF_K_FLOAT:
978                 if (ddp->dd_attr & DT_DA_SIGNED)
979                         (void) strcat(name, "signed ");
980                 if (ddp->dd_attr & DT_DA_UNSIGNED)
981                         (void) strcat(name, "unsigned ");
982                 if (ddp->dd_attr & DT_DA_SHORT)
983                         (void) strcat(name, "short ");
984                 if (ddp->dd_attr & DT_DA_LONG)
985                         (void) strcat(name, "long ");
986                 if (ddp->dd_attr & DT_DA_LONGLONG)
987                         (void) strcat(name, "long long ");
988                 if (ddp->dd_attr == 0 && ddp->dd_name == NULL)
989                         (void) strcat(name, "int");
990                 break;
991         case CTF_K_STRUCT:
992                 (void) strcpy(name, "struct ");
993                 break;
994         case CTF_K_UNION:
995                 (void) strcpy(name, "union ");
996                 break;
997         case CTF_K_ENUM:
998                 (void) strcpy(name, "enum ");
```

```
 999                 break;
1000         case CTF_K_TYPEDEF:
1001                 break;
1002         default:
1003                 xywarn(D_UNKNOWN, "internal error -- "
1004                     "bad decl kind %u\n", ddp->dd_kind);
1005                 return (-1);
1006         }

1008         /*
1009          * Add dd_name unless a short, long, or long long is explicitly
1010          * suffixed by int.  We use the C/CTF canonical names for integers.
1011          */
1012         if (ddp->dd_name != NULL && (ddp->dd_kind != CTF_K_INTEGER ||
1013             (ddp->dd_attr & (DT_DA_SHORT | DT_DA_LONG | DT_DA_LONGLONG)) == 0))
1014                 (void) strcat(name, ddp->dd_name);

1016         /*
1017          * Lookup the type.  If we find it, we're done.  Otherwise create a
1018          * forward tag for the type if it is a struct, union, or enum.  If
1019          * we can't find it and we can't create a tag, return failure.
1020          */
1021         if ((rv = dt_type_lookup(name, tip)) == 0)
1022                 return (rv);

1024         switch (ddp->dd_kind) {
1025         case CTF_K_STRUCT:
1026         case CTF_K_UNION:
1027         case CTF_K_ENUM:
1028                 type = ctf_add_forward(dmp->dm_ctfp, flag,
1029                     ddp->dd_name, ddp->dd_kind);
1030                 break;
1031         default:
1032                 xywarn(D_UNKNOWN, "failed to resolve type %s: %s\n", name,
1033                     dtrace_errmsg(dtp, dtrace_errno(dtp)));
1034                 return (rv);
1035         }

1037         if (type == CTF_ERR || ctf_update(dmp->dm_ctfp) == CTF_ERR) {
1038                 xywarn(D_UNKNOWN, "failed to add forward tag for %s: %s\n",
1039                     name, ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1040                 return (-1);
1041         }

1043         ddp->dd_ctfp = dmp->dm_ctfp;
1044         ddp->dd_type = type;

1046         tip->dtt_object = dmp->dm_name;
1047         tip->dtt_ctfp = dmp->dm_ctfp;
1048         tip->dtt_type = type;

1050         return (0);
1051 }

1053 void
1054 dt_scope_create(dt_scope_t *dsp)
1055 {
1056         dsp->ds_decl = NULL;
1057         dsp->ds_next = NULL;
1058         dsp->ds_ident = NULL;
1059         dsp->ds_ctfp = NULL;
1060         dsp->ds_type = CTF_ERR;
1061         dsp->ds_class = DT_DC_DEFAULT;
1062         dsp->ds_enumval = -1;
1063 }
```



```
1065 void
1066 dt_scope_destroy(dt_scope_t *dsp)
1067 {
1068         dt_scope_t *nsp;

1070         for (; dsp != NULL; dsp = nsp) {
1071                 dt_decl_free(dsp->ds_decl);
1072                 free(dsp->ds_ident);
1073                 nsp = dsp->ds_next;
1074                 if (dsp != &yypcb->pcb_dstack)
1075                         free(dsp);
1076         }
1077 }

1079 void
1080 dt_scope_push(ctf_file_t *ctfp, ctf_id_t type)
1081 {
1082         dt_scope_t *rsp = &yypcb->pcb_dstack;
1083         dt_scope_t *dsp = malloc(sizeof (dt_scope_t));

1085         if (dsp == NULL)
1086                 longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

1088         dsp->ds_decl = rsp->ds_decl;
1089         dsp->ds_next = rsp->ds_next;
1090         dsp->ds_ident = rsp->ds_ident;
1091         dsp->ds_ctfp = ctfp;
1092         dsp->ds_type = type;
1093         dsp->ds_class = rsp->ds_class;
1094         dsp->ds_enumval = rsp->ds_enumval;

1096         dt_scope_create(rsp);
1097         rsp->ds_next = dsp;
1098 }

1100 dt_decl_t *
1101 dt_scope_pop(void)
1102 {
1103         dt_scope_t *rsp = &yypcb->pcb_dstack;
1104         dt_scope_t *dsp = rsp->ds_next;

1106         if (dsp == NULL)
1107                 longjmp(yypcb->pcb_jmpbuf, EDT_NOSCOPE);

1109         if (dsp->ds_ctfp != NULL && ctf_update(dsp->ds_ctfp) == CTF_ERR) {
1110                 xyerror(D_UNKNOWN, "failed to update type definitions: %s\n",
1111                     ctf_errmsg(ctf_errno(dsp->ds_ctfp)));
1112         }

1114         dt_decl_free(rsp->ds_decl);
1115         free(rsp->ds_ident);

1117         rsp->ds_decl = dsp->ds_decl;
1118         rsp->ds_next = dsp->ds_next;
1119         rsp->ds_ident = dsp->ds_ident;
1120         rsp->ds_ctfp = dsp->ds_ctfp;
1121         rsp->ds_type = dsp->ds_type;
1122         rsp->ds_class = dsp->ds_class;
1123         rsp->ds_enumval = dsp->ds_enumval;

1125         free(dsp);
1126         return (rsp->ds_decl);
1127 }
```

```
**********************************************************
    4559 Tue Jan 14 16:48:52 2014
new/usr/src/lib/libdtrace/common/dt_decl.h
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 * Copyright (c) 2013 Joyent, Inc. All rights reserved.
 */
#endif /* ! codereview */

#ifndef _DT_DECL_H
#define _DT_DECL_H

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

#include <sys/types.h>
#include <libctf.h>
#include <dtrace.h>
#include <stdio.h>

#ifdef __cplusplus
extern "C" {
#endif

struct dt_node;                         /* forward declaration of dt_node_t */

typedef struct dt_decl {
        ushort_t dd_kind;               /* declaration kind (CTF_K_* kind) */
        ushort_t dd_attr;               /* attributes (DT_DA_* flags) */
        ctf_file_t *dd_ctfp;            /* CTF container for decl's type */
        ctf_id_t dd_type;               /* CTF identifier for decl's type */
        char *dd_name;                  /* string name of this decl (or NULL) */
        struct dt_node *dd_node;        /* node for array size or parm list */
        struct dt_decl *dd_next;        /* next declaration in list */
} dt_decl_t;
```



```
#define DT_DA_SIGNED    0x0001          /* signed integer value */
#define DT_DA_UNSIGNED  0x0002          /* unsigned integer value */
#define DT_DA_SHORT     0x0004          /* short integer value */
#define DT_DA_LONG      0x0008          /* long integer or double */
#define DT_DA_LONGLONG  0x0010          /* long long integer value */
#define DT_DA_CONST     0x0020          /* qualify type as const */
#define DT_DA_RESTRICT  0x0040          /* qualify type as restrict */
#define DT_DA_VOLATILE  0x0080          /* qualify type as volatile */
#define DT_DA_PAREN     0x0100          /* parenthesis tag */
#define DT_DA_USER      0x0200          /* user-land type specifier */
#endif /* ! codereview */

typedef enum dt_dclass {
        DT_DC_DEFAULT,                  /* no storage class specified */
        DT_DC_AUTO,                     /* automatic storage */
        DT_DC_REGISTER,                 /* register storage */
        DT_DC_STATIC,                   /* static storage */
        DT_DC_EXTERN,                   /* extern storage */
        DT_DC_TYPEDEF,                  /* type definition */
        DT_DC_SELF,                     /* thread-local storage */
        DT_DC_THIS                      /* clause-local storage */
} dt_dclass_t;

typedef struct dt_scope {
        dt_decl_t *ds_decl;             /* pointer to top of decl stack */
        struct dt_scope *ds_next;       /* pointer to next scope */
        char *ds_ident;                 /* identifier for this scope (if any) */
        ctf_file_t *ds_ctfp;            /* CTF container for this scope */
        ctf_id_t ds_type;               /* CTF id of enclosing type */
        dt_dclass_t ds_class;           /* declaration class for this scope */
        int ds_enumval;                 /* most recent enumerator value */
} dt_scope_t;

extern dt_decl_t *dt_decl_alloc(ushort_t, char *);
extern void dt_decl_free(dt_decl_t *);
extern void dt_decl_reset(void);
extern dt_decl_t *dt_decl_push(dt_decl_t *);
extern dt_decl_t *dt_decl_pop(void);
extern dt_decl_t *dt_decl_pop_param(char **);
extern dt_decl_t *dt_decl_top(void);

extern dt_decl_t *dt_decl_ident(char *);
extern void dt_decl_class(dt_dclass_t);

#define DT_DP_VARARGS   0x1             /* permit varargs in prototype */
#define DT_DP_DYNAMIC   0x2             /* permit dynamic type in prototype */
#define DT_DP_VOID      0x4             /* permit void type in prototype */
#define DT_DP_ANON      0x8             /* permit anonymous parameters */

extern int dt_decl_prototype(struct dt_node *, struct dt_node *,
    const char *, uint_t);

extern dt_decl_t *dt_decl_spec(ushort_t, char *);
extern dt_decl_t *dt_decl_attr(ushort_t);
extern dt_decl_t *dt_decl_array(struct dt_node *);
extern dt_decl_t *dt_decl_func(dt_decl_t *, struct dt_node *);
extern dt_decl_t *dt_decl_ptr(void);

extern dt_decl_t *dt_decl_sou(uint_t, char *);
extern void dt_decl_member(struct dt_node *);

extern dt_decl_t *dt_decl_enum(char *);
extern void dt_decl_enumerator(char *, struct dt_node *);

extern int dt_decl_type(dt_decl_t *, dtrace_typeinfo_t *);
```

```
122 extern void dt_scope_create(dt_scope_t *);
123 extern void dt_scope_destroy(dt_scope_t *);
124 extern void dt_scope_push(ctf_file_t *, ctf_id_t);
125 extern dt_decl_t *dt_scope_pop(void);

127 #ifdef  __cplusplus
128 }
129 #endif

131 #endif  /* _DT_DECL_H */
```

```
**********************************************************
   15006 Tue Jan 14 16:48:52 2014
new/usr/src/lib/libdtrace/common/dt_dis.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 * Copyright (c) 2013 Joyent, Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

#include <strings.h>
#include <stdio.h>

#include <dt_impl.h>
#include <dt_ident.h>

/*ARGSUSED*/
static void
dt_dis_log(const dtrace_difo_t *dp, const char *name, dif_instr_t in, FILE *fp)
{
        (void) fprintf(fp, "%-4s %%r%u, %%r%u, %%r%u", name,
            DIF_INSTR_R1(in), DIF_INSTR_R2(in), DIF_INSTR_RD(in));
}
```

*unchanged_portion_omitted_*

```
static char *
dt_dis_typestr(const dtrace_diftype_t *t, char *buf, size_t len)
{
        char kind[16], ckind[16];

        switch (t->dtdt_kind) {
        case DIF_TYPE_CTF:
                (void) strcpy(kind, "D type");
```



```
                break;
        case DIF_TYPE_STRING:
                (void) strcpy(kind, "string");
                break;
        default:
                (void) snprintf(kind, sizeof (kind), "0x%x", t->dtdt_kind);
        }

        switch (t->dtdt_ckind) {
        case CTF_K_UNKNOWN:
                (void) strcpy(ckind, "unknown");
                break;
        case CTF_K_INTEGER:
                (void) strcpy(ckind, "integer");
                break;
        case CTF_K_FLOAT:
                (void) strcpy(ckind, "float");
                break;
        case CTF_K_POINTER:
                (void) strcpy(ckind, "pointer");
                break;
        case CTF_K_ARRAY:
                (void) strcpy(ckind, "array");
                break;
        case CTF_K_FUNCTION:
                (void) strcpy(ckind, "function");
                break;
        case CTF_K_STRUCT:
                (void) strcpy(ckind, "struct");
                break;
        case CTF_K_UNION:
                (void) strcpy(ckind, "union");
                break;
        case CTF_K_ENUM:
                (void) strcpy(ckind, "enum");
                break;
        case CTF_K_FORWARD:
                (void) strcpy(ckind, "forward");
                break;
        case CTF_K_TYPEDEF:
                (void) strcpy(ckind, "typedef");
                break;
        case CTF_K_VOLATILE:
                (void) strcpy(ckind, "volatile");
                break;
        case CTF_K_CONST:
                (void) strcpy(ckind, "const");
                break;
        case CTF_K_RESTRICT:
                (void) strcpy(ckind, "restrict");
                break;
        default:
                (void) snprintf(ckind, sizeof (ckind), "0x%x", t->dtdt_ckind);
        }

        if (t->dtdt_flags & (DIF_TF_BYREF | DIF_TF_BYUREF)) {
                (void) snprintf(buf, len, "%s (%s) by %sref (size %lu)",
                    kind, ckind, (t->dtdt_flags & DIF_TF_BYUREF) ? "user " : "",
                    (ulong_t)t->dtdt_size);
        if (t->dtdt_flags & DIF_TF_BYREF) {
                (void) snprintf(buf, len, "%s (%s) by ref (size %lu)",
                    kind, ckind, (ulong_t)t->dtdt_size);
        } else {
                (void) snprintf(buf, len, "%s (%s) (size %lu)",
                    kind, ckind, (ulong_t)t->dtdt_size);
        }
```

```
 325         return (buf);
 326 }
______unchanged_portion_omitted_
```

```
**********************************************************
    7980 Tue Jan 14 16:48:53 2014
new/usr/src/lib/libdtrace/common/dt_error.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
______unchanged_portion_omitted_
```

```
  39         { EDT_VERSION,  "Client requested version newer than library" },
  40         { EDT_VERSINVAL, "Version is not properly formatted or is too large" },
  41         { EDT_VERSUNDEF, "Requested version is not supported by compiler" },
  42         { EDT_VERSREDUCED, "Requested version conflicts with earlier setting" },
  43         { EDT_CTF,      "Unexpected libctf error" },
  44         { EDT_COMPILER, "Error in D program compilation" },
  45         { EDT_NOTUPREG, "Insufficient tuple registers to generate code" },
  46         { EDT_NOMEM,    "Memory allocation failure" },
  47         { EDT_INT2BIG,  "Integer constant table limit exceeded" },
  48         { EDT_STR2BIG,  "String constant table limit exceeded" },
  49         { EDT_NOMOD,    "Unknown module name" },
  50         { EDT_NOPROV,   "Unknown provider name" },
  51         { EDT_NOPROBE,  "No probe matches description" },
  52         { EDT_NOSYM,    "Unknown symbol name" },
  53         { EDT_NOSYMADDR, "No symbol corresponds to address" },
  54         { EDT_NOTYPE,   "Unknown type name" },
  55         { EDT_NOVAR,    "Unknown variable name" },
  56         { EDT_NOAGG,    "Unknown aggregation name" },
  57         { EDT_BADSCOPE, "Improper use of scoping operator in type name" },
  58         { EDT_BADSPEC,  "Overspecified probe description" },
  59         { EDT_BADSPCV,  "Undefined macro variable in probe description" },
  60         { EDT_BADID,    "Unknown probe identifier" },
  61         { EDT_NOTLOADED, "Module is no longer loaded" },
  62         { EDT_NOCTF,    "Module does not contain any CTF data" },
  63         { EDT_DATAMODEL, "Module and program data models do not match" },
  64         { EDT_DIFVERS,  "Library uses newer DIF version than kernel" },
  65         { EDT_BADAGG,   "Unknown aggregating action" },
  66         { EDT_FIO,      "Error occurred while reading from input stream" },
  67         { EDT_DIFINVAL, "DIF program content is invalid" },
  68         { EDT_DIFSIZE,  "DIF program exceeds maximum program size" },
  69         { EDT_DIFFAULT, "DIF program contains invalid pointer" },
  70         { EDT_BADPROBE, "Invalid probe specification" },
  71         { EDT_BADPGLOB, "Probe description has too many globbing characters" },
  72         { EDT_NOSCOPE,  "Declaration scope stack underflow" },
  73         { EDT_NODECL,   "Declaration stack underflow" },
  74         { EDT_DMISMATCH, "Data record list does not match statement" },
  75         { EDT_DOFFSET,  "Data record offset exceeds buffer boundary" },
  76         { EDT_DALIGN,   "Data record has inappropriate alignment" },
  77         { EDT_BADOPTNAME, "Invalid option name" },
  78         { EDT_BADOPTVAL, "Invalid value for specified option" },
  79         { EDT_BADOPTCTX, "Option cannot be used from within a D program" },
  80         { EDT_CPPFORK,  "Failed to fork preprocessor" },
  81         { EDT_CPPEXEC,  "Failed to exec preprocessor" },
  82         { EDT_CPPENT,   "Preprocessor not found" },
  83         { EDT_CPPERR,   "Preprocessor failed to process input program" },
  84         { EDT_SYMOFLOW, "Symbol table identifier space exhausted" },
  85         { EDT_ACTIVE,   "Operation illegal when tracing is active" },
  86         { EDT_DESTRUCTIVE, "Destructive actions not allowed" },
  87         { EDT_NOANON,   "No anonymous tracing state" },
  88         { EDT_ISANON,   "Can't claim anonymous state and enable probes" },
  89         { EDT_ENDTOOBIG, "END enablings exceed size of principal buffer" },
  90         { EDT_NOCONV,   "Failed to load type for printf conversion" },
  91         { EDT_BADCONV,  "Incomplete printf conversion" },
  92         { EDT_BADERROR, "Invalid library ERROR action" },
  93         { EDT_ERRABORT, "Abort due to error" },
```



```
  94         { EDT_DROPABORT, "Abort due to drop" },
  95         { EDT_DIRABORT, "Abort explicitly directed" },
  96         { EDT_BADRVAL,  "Invalid return value from callback" },
  97         { EDT_BADNORMAL, "Invalid normalization" },
  98         { EDT_BUFTOOSMALL, "Enabling exceeds size of buffer" },
  99         { EDT_BADTRUNC, "Invalid truncation" },
 100         { EDT_BUSY, "DTrace cannot be used when kernel debugger is active" },
 101         { EDT_ACCESS, "DTrace requires additional privileges" },
 102         { EDT_NOENT, "DTrace device not available on system" },
 103         { EDT_BRICKED, "Abort due to systemic unresponsiveness" },
 104         { EDT_HARDWIRE, "Failed to load language definitions" },
 105         { EDT_ELFVERSION, "libelf is out-of-date with respect to libdtrace" },
 106         { EDT_NOBUFFERED, "Attempt to buffer output without handler" },
 107         { EDT_UNSTABLE, "Description matched an unstable set of probes" },
 108         { EDT_BADSETOPT, "Invalid setopt() library action" },
 109         { EDT_BADSTACKPC, "Invalid stack program counter size" },
 110         { EDT_BADAGGVAR, "Invalid aggregation variable identifier" },
 111         { EDT_OVERSION, "Client requested deprecated version of library" },
 112         { EDT_ENABLING_ERR, "Failed to enable probe" },
 113         { EDT_NOPROBES, "No probe sites found for declared provider" },
 114         { EDT_CANTLOAD, "Failed to load module" },
 113         { EDT_NOPROBES, "No probe sites found for declared provider" }
 115 };
______unchanged_portion_omitted_
```

```
**********************************************************
   22377 Tue Jan 14 16:48:53 2014
new/usr/src/lib/libdtrace/common/dt_grammar.y
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 */

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

#include <dt_impl.h>

#define OP1(op, c)      dt_node_op1(op, c)
#define OP2(op, l, r)   dt_node_op2(op, l, r)
#define OP3(x, y, z)    dt_node_op3(x, y, z)
#define LINK(l, r)      dt_node_link(l, r)
#define DUP(s)          strdup(s)

%}

%union {
        dt_node_t *l_node;
        dt_decl_t *l_decl;
        char *l_str;
        uintmax_t l_int;
        int l_tok;
}

%token  DT_TOK_COMMA DT_TOK_ELLIPSIS
%token  DT_TOK_ASGN DT_TOK_ADD_EQ DT_TOK_SUB_EQ DT_TOK_MUL_EQ
%token  DT_TOK_DIV_EQ DT_TOK_MOD_EQ DT_TOK_AND_EQ DT_TOK_XOR_EQ DT_TOK_OR_EQ
%token  DT_TOK_LSH_EQ DT_TOK_RSH_EQ DT_TOK_QUESTION DT_TOK_COLON
%token  DT_TOK_LOR DT_TOK_LXOR DT_TOK_LAND
%token  DT_TOK_BOR DT_TOK_XOR DT_TOK_BAND DT_TOK_EQU DT_TOK_NEQ
```



```
%token  DT_TOK_LT DT_TOK_LE DT_TOK_GT DT_TOK_GE DT_TOK_LSH DT_TOK_RSH
%token  DT_TOK_ADD DT_TOK_SUB DT_TOK_MUL DT_TOK_DIV DT_TOK_MOD
%token  DT_TOK_LNEG DT_TOK_BNEG DT_TOK_ADDADD DT_TOK_SUBSUB
%token  DT_TOK_PREINC DT_TOK_POSTINC DT_TOK_PREDEC DT_TOK_POSTDEC
%token  DT_TOK_IPOS DT_TOK_INEG DT_TOK_DEREF DT_TOK_ADDROF
%token  DT_TOK_OFFSETOF DT_TOK_SIZEOF DT_TOK_STRINGOF DT_TOK_XLATE
%token  DT_TOK_LPAR DT_TOK_RPAR DT_TOK_LBRAC DT_TOK_RBRAC DT_TOK_PTR DT_TOK_DOT

%token <l_str>  DT_TOK_STRING
%token <l_str>  DT_TOK_IDENT
%token <l_str>  DT_TOK_PSPEC
%token <l_str>  DT_TOK_AGG
%token <l_str>  DT_TOK_TNAME
%token <l_int>  DT_TOK_INT

%token  DT_KEY_AUTO
%token  DT_KEY_BREAK
%token  DT_KEY_CASE
%token  DT_KEY_CHAR
%token  DT_KEY_CONST
%token  DT_KEY_CONTINUE
%token  DT_KEY_COUNTER
%token  DT_KEY_DEFAULT
%token  DT_KEY_DO
%token  DT_KEY_DOUBLE
%token  DT_KEY_ELSE
%token  DT_KEY_ENUM
%token  DT_KEY_EXTERN
%token  DT_KEY_FLOAT
%token  DT_KEY_FOR
%token  DT_KEY_GOTO
%token  DT_KEY_IF
%token  DT_KEY_IMPORT
%token  DT_KEY_INLINE
%token  DT_KEY_INT
%token  DT_KEY_LONG
%token  DT_KEY_PROBE
%token  DT_KEY_PROVIDER
%token  DT_KEY_REGISTER
%token  DT_KEY_RESTRICT
%token  DT_KEY_RETURN
%token  DT_KEY_SELF
%token  DT_KEY_SHORT
%token  DT_KEY_SIGNED
%token  DT_KEY_STATIC
%token  DT_KEY_STRING
%token  DT_KEY_STRUCT
%token  DT_KEY_SWITCH
%token  DT_KEY_THIS
%token  DT_KEY_TYPEDEF
%token  DT_KEY_UNION
%token  DT_KEY_UNSIGNED
%token  DT_KEY_USERLAND
#endif /* ! codereview */
%token  DT_KEY_VOID
%token  DT_KEY_VOLATILE
%token  DT_KEY_WHILE
%token  DT_KEY_XLATOR

%token  DT_TOK_EPRED
%token  DT_CTX_DEXPR
%token  DT_CTX_DPROG
%token  DT_CTX_DTYPE
%token  DT_TOK_EOF      0

%left   DT_TOK_COMMA
```

```
121 %right  DT_TOK_ASGN DT_TOK_ADD_EQ DT_TOK_SUB_EQ DT_TOK_MUL_EQ DT_TOK_DIV_EQ
122         DT_TOK_MOD_EQ DT_TOK_AND_EQ DT_TOK_XOR_EQ DT_TOK_OR_EQ DT_TOK_LSH_EQ
123         DT_TOK_RSH_EQ
124 %left   DT_TOK_QUESTION DT_TOK_COLON
125 %left   DT_TOK_LOR
126 %left   DT_TOK_LXOR
127 %left   DT_TOK_LAND
128 %left   DT_TOK_BOR
129 %left   DT_TOK_XOR
130 %left   DT_TOK_BAND
131 %left   DT_TOK_EQU DT_TOK_NEQ
132 %left   DT_TOK_LT DT_TOK_LE DT_TOK_GT DT_TOK_GE
133 %left   DT_TOK_LSH DT_TOK_RSH
134 %left   DT_TOK_ADD DT_TOK_SUB
135 %left   DT_TOK_MUL DT_TOK_DIV DT_TOK_MOD
136 %right  DT_TOK_LNEG DT_TOK_BNEG DT_TOK_ADDADD DT_TOK_SUBSUB
137         DT_TOK_IPOS DT_TOK_INEG
138 %right  DT_TOK_DEREF DT_TOK_ADDROF DT_TOK_SIZEOF DT_TOK_STRINGOF DT_TOK_XLATE
139 %left   DT_TOK_LPAR DT_TOK_RPAR DT_TOK_LBRAC DT_TOK_RBRAC DT_TOK_PTR DT_TOK_DOT

141 %type   <l_node>        d_expression
142 %type   <l_node>        d_program
143 %type   <l_node>        d_type

145 %type   <l_node>        translation_unit
146 %type   <l_node>        external_declaration
147 %type   <l_node>        inline_definition
148 %type   <l_node>        translator_definition
149 %type   <l_node>        translator_member_list
150 %type   <l_node>        translator_member
151 %type   <l_node>        provider_definition
152 %type   <l_node>        provider_probe_list
153 %type   <l_node>        provider_probe
154 %type   <l_node>        probe_definition
155 %type   <l_node>        probe_specifiers
156 %type   <l_node>        probe_specifier_list
157 %type   <l_node>        probe_specifier
158 %type   <l_node>        statement_list
159 %type   <l_node>        statement
160 %type   <l_node>        declaration
161 %type   <l_node>        init_declarator_list
162 %type   <l_node>        init_declarator

164 %type   <l_decl>        type_specifier
165 %type   <l_decl>        type_qualifier
166 %type   <l_decl>        struct_or_union_specifier
167 %type   <l_decl>        specifier_qualifier_list
168 %type   <l_decl>        enum_specifier
169 %type   <l_decl>        declarator
170 %type   <l_decl>        direct_declarator
171 %type   <l_decl>        pointer
172 %type   <l_decl>        type_qualifier_list
173 %type   <l_decl>        type_name
174 %type   <l_decl>        abstract_declarator
175 %type   <l_decl>        direct_abstract_declarator

177 %type   <l_node>        parameter_type_list
178 %type   <l_node>        parameter_list
179 %type   <l_node>        parameter_declaration

181 %type   <l_node>        array
182 %type   <l_node>        array_parameters
183 %type   <l_node>        function
184 %type   <l_node>        function_parameters

186 %type   <l_node>        expression
```



```
187 %type   <l_node>        assignment_expression
188 %type   <l_node>        conditional_expression
189 %type   <l_node>        constant_expression
190 %type   <l_node>        logical_or_expression
191 %type   <l_node>        logical_xor_expression
192 %type   <l_node>        logical_and_expression
193 %type   <l_node>        inclusive_or_expression
194 %type   <l_node>        exclusive_or_expression
195 %type   <l_node>        and_expression
196 %type   <l_node>        equality_expression
197 %type   <l_node>        relational_expression
198 %type   <l_node>        shift_expression
199 %type   <l_node>        additive_expression
200 %type   <l_node>        multiplicative_expression
201 %type   <l_node>        cast_expression
202 %type   <l_node>        unary_expression
203 %type   <l_node>        postfix_expression
204 %type   <l_node>        primary_expression
205 %type   <l_node>        argument_expression_list

207 %type   <l_tok>         assignment_operator
208 %type   <l_tok>         unary_operator
209 %type   <l_tok>         struct_or_union

211 %%

213 dtrace_program: d_expression DT_TOK_EOF { return (dt_node_root($1)); }
214         |       d_program DT_TOK_EOF { return (dt_node_root($1)); }
215         |       d_type DT_TOK_EOF { return (dt_node_root($1)); }
216         ;

218 d_expression:   DT_CTX_DEXPR { $$ = NULL; }
219         |       DT_CTX_DEXPR expression { $$ = $2; }
220         ;

222 d_program:      DT_CTX_DPROG { $$ = dt_node_program(NULL); }
223         |       DT_CTX_DPROG translation_unit { $$ = dt_node_program($2); }
224         ;

226 d_type:         DT_CTX_DTYPE { $$ = NULL; }
227         |       DT_CTX_DTYPE type_name { $$ = (dt_node_t *)$2; }
228         ;

230 translation_unit:
231                 external_declaration
232         |       translation_unit external_declaration { $$ = LINK($1, $2); }
233         ;

235 external_declaration:
236                 inline_definition
237         |       translator_definition
238         |       provider_definition
239         |       probe_definition
240         |       declaration
241         ;

243 inline_definition:
244                 DT_KEY_INLINE declaration_specifiers declarator
245                     { dt_scope_push(NULL, CTF_ERR); } DT_TOK_ASGN
246                     assignment_expression ';' {
247                         /*
248                          * We push a new declaration scope before shifting the
249                          * assignment_expression in order to preserve ds_class
250                          * and ds_ident for use in dt_node_inline().  Once the
251                          * entire inline_definition rule is matched, pop the
252                          * scope and construct the inline using the saved decl.
```

```
253                          */
254                         dt_scope_pop();
255                         $$ = dt_node_inline($6);
256                 }
257         ;

259 translator_definition:
260                 DT_KEY_XLATOR type_name DT_TOK_LT type_name
261                     DT_TOK_IDENT DT_TOK_GT '{' translator_member_list '}' ';' {
262                         $$ = dt_node_xlator($2, $4, $5, $8);
263                 }
264         |       DT_KEY_XLATOR type_name DT_TOK_LT type_name
265                     DT_TOK_IDENT DT_TOK_GT '{' '}' ';' {
266                         $$ = dt_node_xlator($2, $4, $5, NULL);
267                 }
268         ;

270 translator_member_list:
271                 translator_member
272         |       translator_member_list translator_member { $$ = LINK($1,$2); }
273         ;

275 translator_member:
276                 DT_TOK_IDENT DT_TOK_ASGN assignment_expression ';' {
277                         $$ = dt_node_member(NULL, $1, $3);
278                 }
279         ;

281 provider_definition:
282                 DT_KEY_PROVIDER DT_TOK_IDENT '{' provider_probe_list '}' ';' {
283                         $$ = dt_node_provider($2, $4);
284                 }
285         |       DT_KEY_PROVIDER DT_TOK_IDENT '{' '}' ';' {
286                         $$ = dt_node_provider($2, NULL);
287                 }
288         ;

290 provider_probe_list:
291                 provider_probe
292         |       provider_probe_list provider_probe { $$ = LINK($1, $2); }
293         ;

295 provider_probe:
296                 DT_KEY_PROBE DT_TOK_IDENT function DT_TOK_COLON function ';' {
297                         $$ = dt_node_probe($2, 2, $3, $5);
298                 }
299         |       DT_KEY_PROBE DT_TOK_IDENT function ';' {
300                         $$ = dt_node_probe($2, 1, $3, NULL);
301                 }
302         ;
303

305 probe_definition:
306                 probe_specifiers {
307                         /*
308                          * If the input stream is a file, do not permit a probe
309                          * specification without / <pred> / or { <act> } after
310                          * it.  This can only occur if the next token is EOF or
311                          * an ambiguous predicate was slurped up as a comment.
312                          * We cannot perform this check if input() is a string
313                          * because dtrace(1M) [-fmnP] also use the compiler and
314                          * things like dtrace -n BEGIN have to be accepted.
315                          */
316                         if (yypcb->pcb_fileptr != NULL) {
317                                 dnerror($1, D_SYNTAX, "expected predicate and/"
318                                     "or actions following probe description\n");
```



```
319                         }
320                         $$ = dt_node_clause($1, NULL, NULL);
321                 }
322         |       probe_specifiers '{' statement_list '}' {
323                         $$ = dt_node_clause($1, NULL, $3);
324                 }
325         |       probe_specifiers DT_TOK_DIV expression DT_TOK_EPRED {
326                         dnerror($3, D_SYNTAX, "expected actions { } following "
327                             "probe description and predicate\n");
328                 }
329         |       probe_specifiers DT_TOK_DIV expression DT_TOK_EPRED
330                     '{' statement_list '}' {
331                         $$ = dt_node_clause($1, $3, $6);
332                 }
333         ;

335 probe_specifiers:
336                 probe_specifier_list { yybegin(YYS_EXPR); $$ = $1; }
337         ;

339 probe_specifier_list:
340                 probe_specifier
341         |       probe_specifier_list DT_TOK_COMMA probe_specifier {
342                         $$ = LINK($1, $3);
343                 }
344         ;

346 probe_specifier:
347                 DT_TOK_PSPEC { $$ = dt_node_pdesc_by_name($1); }
348         |       DT_TOK_INT   { $$ = dt_node_pdesc_by_id($1); }
349         ;

351 statement_list: statement { $$ = $1; }
352         |       statement_list ';' statement { $$ = LINK($1, $3); }
353         ;

355 statement:     /* empty */ { $$ = NULL; }
356         |       expression { $$ = dt_node_statement($1); }
357         ;

359 argument_expression_list:
360                 assignment_expression
361         |       argument_expression_list DT_TOK_COMMA assignment_expression {
362                         $$ = LINK($1, $3);
363                 }
364         ;

366 primary_expression:
367                 DT_TOK_IDENT { $$ = dt_node_ident($1); }
368         |       DT_TOK_AGG { $$ = dt_node_ident($1); }
369         |       DT_TOK_INT { $$ = dt_node_int($1); }
370         |       DT_TOK_STRING { $$ = dt_node_string($1); }
371         |       DT_KEY_SELF { $$ = dt_node_ident(DUP("self")); }
372         |       DT_KEY_THIS { $$ = dt_node_ident(DUP("this")); }
373         |       DT_TOK_LPAR expression DT_TOK_RPAR { $$ = $2; }
374         ;

376 postfix_expression:
377                 primary_expression
378         |       postfix_expression
379                     DT_TOK_LBRAC argument_expression_list DT_TOK_RBRAC {
380                         $$ = OP2(DT_TOK_LBRAC, $1, $3);
381                 }
382         |       postfix_expression DT_TOK_LPAR DT_TOK_RPAR {
383                         $$ = dt_node_func($1, NULL);
384                 }
```

```
385         |       postfix_expression
386                     DT_TOK_LPAR argument_expression_list DT_TOK_RPAR {
387                         $$ = dt_node_func($1, $3);
388                 }
389         |       postfix_expression DT_TOK_DOT DT_TOK_IDENT {
390                         $$ = OP2(DT_TOK_DOT, $1, dt_node_ident($3));
391                 }
392         |       postfix_expression DT_TOK_DOT DT_TOK_TNAME {
393                         $$ = OP2(DT_TOK_DOT, $1, dt_node_ident($3));
394                 }
395         |       postfix_expression DT_TOK_PTR DT_TOK_IDENT {
396                         $$ = OP2(DT_TOK_PTR, $1, dt_node_ident($3));
397                 }
398         |       postfix_expression DT_TOK_PTR DT_TOK_TNAME {
399                         $$ = OP2(DT_TOK_PTR, $1, dt_node_ident($3));
400                 }
401         |       postfix_expression DT_TOK_ADDADD {
402                         $$ = OP1(DT_TOK_POSTINC, $1);
403                 }
404         |       postfix_expression DT_TOK_SUBSUB {
405                         $$ = OP1(DT_TOK_POSTDEC, $1);
406                 }
407         |       DT_TOK_OFFSETOF DT_TOK_LPAR type_name DT_TOK_COMMA
408                     DT_TOK_IDENT DT_TOK_RPAR {
409                         $$ = dt_node_offsetof($3, $5);
410                 }
411         |       DT_TOK_OFFSETOF DT_TOK_LPAR type_name DT_TOK_COMMA
412                     DT_TOK_TNAME DT_TOK_RPAR {
413                         $$ = dt_node_offsetof($3, $5);
414                 }
415         |       DT_TOK_XLATE DT_TOK_LT type_name DT_TOK_GT
416                     DT_TOK_LPAR expression DT_TOK_RPAR {
417                         $$ = OP2(DT_TOK_XLATE, dt_node_type($3), $6);
418                 }
419         ;

421 unary_expression:
422                 postfix_expression
423         |       DT_TOK_ADDADD unary_expression { $$ = OP1(DT_TOK_PREINC, $2); }
424         |       DT_TOK_SUBSUB unary_expression { $$ = OP1(DT_TOK_PREDEC, $2); }
425         |       unary_operator cast_expression { $$ = OP1($1, $2); }
426         |       DT_TOK_SIZEOF unary_expression { $$ = OP1(DT_TOK_SIZEOF, $2); }
427         |       DT_TOK_SIZEOF DT_TOK_LPAR type_name DT_TOK_RPAR {
428                         $$ = OP1(DT_TOK_SIZEOF, dt_node_type($3));
429                 }
430         |       DT_TOK_STRINGOF unary_expression {
431                         $$ = OP1(DT_TOK_STRINGOF, $2);
432                 }
433         ;

435 unary_operator: DT_TOK_BAND { $$ = DT_TOK_ADDROF; }
436         |       DT_TOK_MUL { $$ = DT_TOK_DEREF; }
437         |       DT_TOK_ADD { $$ = DT_TOK_IPOS; }
438         |       DT_TOK_SUB { $$ = DT_TOK_INEG; }
439         |       DT_TOK_BNEG { $$ = DT_TOK_BNEG; }
440         |       DT_TOK_LNEG { $$ = DT_TOK_LNEG; }
441         ;

443 cast_expression:
444                 unary_expression
445         |       DT_TOK_LPAR type_name DT_TOK_RPAR cast_expression {
446                         $$ = OP2(DT_TOK_LPAR, dt_node_type($2), $4);
447                 }
448         ;

450 multiplicative_expression:
```



```
451                 cast_expression
452         |       multiplicative_expression DT_TOK_MUL cast_expression {
453                         $$ = OP2(DT_TOK_MUL, $1, $3);
454                 }
455         |       multiplicative_expression DT_TOK_DIV cast_expression {
456                         $$ = OP2(DT_TOK_DIV, $1, $3);
457                 }
458         |       multiplicative_expression DT_TOK_MOD cast_expression {
459                         $$ = OP2(DT_TOK_MOD, $1, $3);
460                 }
461         ;

463 additive_expression:
464                 multiplicative_expression
465         |       additive_expression DT_TOK_ADD multiplicative_expression {
466                         $$ = OP2(DT_TOK_ADD, $1, $3);
467                 }
468         |       additive_expression DT_TOK_SUB multiplicative_expression {
469                         $$ = OP2(DT_TOK_SUB, $1, $3);
470                 }
471         ;

473 shift_expression:
474                 additive_expression
475         |       shift_expression DT_TOK_LSH additive_expression {
476                         $$ = OP2(DT_TOK_LSH, $1, $3);
477                 }
478         |       shift_expression DT_TOK_RSH additive_expression {
479                         $$ = OP2(DT_TOK_RSH, $1, $3);
480                 }
481         ;

483 relational_expression:
484                 shift_expression
485         |       relational_expression DT_TOK_LT shift_expression {
486                         $$ = OP2(DT_TOK_LT, $1, $3);
487                 }
488         |       relational_expression DT_TOK_GT shift_expression {
489                         $$ = OP2(DT_TOK_GT, $1, $3);
490                 }
491         |       relational_expression DT_TOK_LE shift_expression {
492                         $$ = OP2(DT_TOK_LE, $1, $3);
493                 }
494         |       relational_expression DT_TOK_GE shift_expression {
495                         $$ = OP2(DT_TOK_GE, $1, $3);
496                 }
497         ;

499 equality_expression:
500                 relational_expression
501         |       equality_expression DT_TOK_EQU relational_expression {
502                         $$ = OP2(DT_TOK_EQU, $1, $3);
503                 }
504         |       equality_expression DT_TOK_NEQ relational_expression {
505                         $$ = OP2(DT_TOK_NEQ, $1, $3);
506                 }
507         ;

509 and_expression:
510                 equality_expression
511         |       and_expression DT_TOK_BAND equality_expression {
512                         $$ = OP2(DT_TOK_BAND, $1, $3);
513                 }
514         ;

516 exclusive_or_expression:
```

```
517                 and_expression
518         |       exclusive_or_expression DT_TOK_XOR and_expression {
519                         $$ = OP2(DT_TOK_XOR, $1, $3);
520                 }
521         ;

523 inclusive_or_expression:
524                 exclusive_or_expression
525         |       inclusive_or_expression DT_TOK_BOR exclusive_or_expression {
526                         $$ = OP2(DT_TOK_BOR, $1, $3);
527                 }
528         ;

530 logical_and_expression:
531                 inclusive_or_expression
532         |       logical_and_expression DT_TOK_LAND inclusive_or_expression {
533                         $$ = OP2(DT_TOK_LAND, $1, $3);
534                 }
535         ;

537 logical_xor_expression:
538                 logical_and_expression
539         |       logical_xor_expression DT_TOK_LXOR logical_and_expression {
540                         $$ = OP2(DT_TOK_LXOR, $1, $3);
541                 }
542         ;

544 logical_or_expression:
545                 logical_xor_expression
546         |       logical_or_expression DT_TOK_LOR logical_xor_expression {
547                         $$ = OP2(DT_TOK_LOR, $1, $3);
548                 }
549         ;

551 constant_expression: conditional_expression
552         ;

554 conditional_expression:
555                 logical_or_expression
556         |       logical_or_expression DT_TOK_QUESTION expression DT_TOK_COLON
557                     conditional_expression { $$ = OP3($1, $3, $5); }
558         ;

560 assignment_expression:
561                 conditional_expression
562         |       unary_expression assignment_operator assignment_expression {
563                         $$ = OP2($2, $1, $3);
564                 }
565         ;

567 assignment_operator:
568                 DT_TOK_ASGN   { $$ = DT_TOK_ASGN; }
569         |       DT_TOK_MUL_EQ { $$ = DT_TOK_MUL_EQ; }
570         |       DT_TOK_DIV_EQ { $$ = DT_TOK_DIV_EQ; }
571         |       DT_TOK_MOD_EQ { $$ = DT_TOK_MOD_EQ; }
572         |       DT_TOK_ADD_EQ { $$ = DT_TOK_ADD_EQ; }
573         |       DT_TOK_SUB_EQ { $$ = DT_TOK_SUB_EQ; }
574         |       DT_TOK_LSH_EQ { $$ = DT_TOK_LSH_EQ; }
575         |       DT_TOK_RSH_EQ { $$ = DT_TOK_RSH_EQ; }
576         |       DT_TOK_AND_EQ { $$ = DT_TOK_AND_EQ; }
577         |       DT_TOK_XOR_EQ { $$ = DT_TOK_XOR_EQ; }
578         |       DT_TOK_OR_EQ  { $$ = DT_TOK_OR_EQ; }
579         ;

581 expression:    assignment_expression
582         |       expression DT_TOK_COMMA assignment_expression {
```



```
583                         $$ = OP2(DT_TOK_COMMA, $1, $3);
584                 }
585         ;

587 declaration:   declaration_specifiers ';' {
588                         $$ = dt_node_decl();
589                         dt_decl_free(dt_decl_pop());
590                         yybegin(YYS_CLAUSE);
591                 }
592         |       declaration_specifiers init_declarator_list ';' {
593                         $$ = $2;
594                         dt_decl_free(dt_decl_pop());
595                         yybegin(YYS_CLAUSE);
596                 }
597         ;

599 declaration_specifiers:
600                 d_storage_class_specifier
601         |       d_storage_class_specifier declaration_specifiers
602         |       type_specifier
603         |       type_specifier declaration_specifiers
604         |       type_qualifier
605         |       type_qualifier declaration_specifiers
606         ;

608 parameter_declaration_specifiers:
609                 storage_class_specifier
610         |       storage_class_specifier declaration_specifiers
611         |       type_specifier
612         |       type_specifier declaration_specifiers
613         |       type_qualifier
614         |       type_qualifier declaration_specifiers
615         ;

617 storage_class_specifier:
618                 DT_KEY_AUTO { dt_decl_class(DT_DC_AUTO); }
619         |       DT_KEY_REGISTER { dt_decl_class(DT_DC_REGISTER); }
620         |       DT_KEY_STATIC { dt_decl_class(DT_DC_STATIC); }
621         |       DT_KEY_EXTERN { dt_decl_class(DT_DC_EXTERN); }
622         |       DT_KEY_TYPEDEF { dt_decl_class(DT_DC_TYPEDEF); }
623         ;

625 d_storage_class_specifier:
626                 storage_class_specifier
627         |       DT_KEY_SELF { dt_decl_class(DT_DC_SELF); }
628         |       DT_KEY_THIS { dt_decl_class(DT_DC_THIS); }
629         ;

631 type_specifier: DT_KEY_VOID { $$ = dt_decl_spec(CTF_K_INTEGER, DUP("void")); }
632         |       DT_KEY_CHAR { $$ = dt_decl_spec(CTF_K_INTEGER, DUP("char")); }
633         |       DT_KEY_SHORT { $$ = dt_decl_attr(DT_DA_SHORT); }
634         |       DT_KEY_INT { $$ = dt_decl_spec(CTF_K_INTEGER, DUP("int")); }
635         |       DT_KEY_LONG { $$ = dt_decl_attr(DT_DA_LONG); }
636         |       DT_KEY_FLOAT { $$ = dt_decl_spec(CTF_K_FLOAT, DUP("float")); }
637         |       DT_KEY_DOUBLE { $$ = dt_decl_spec(CTF_K_FLOAT, DUP("double")); }
638         |       DT_KEY_SIGNED { $$ = dt_decl_attr(DT_DA_SIGNED); }
639         |       DT_KEY_UNSIGNED { $$ = dt_decl_attr(DT_DA_UNSIGNED); }
640         |       DT_KEY_USERLAND { $$ = dt_decl_attr(DT_DA_USER); }
641 #endif /* ! codereview */
642         |       DT_KEY_STRING {
643                         $$ = dt_decl_spec(CTF_K_TYPEDEF, DUP("string"));
644                 }
645         |       DT_TOK_TNAME { $$ = dt_decl_spec(CTF_K_TYPEDEF, $1); }
646         |       struct_or_union_specifier
647         |       enum_specifier
648         ;
```

```
650 type_qualifier: DT_KEY_CONST { $$ = dt_decl_attr(DT_DA_CONST); }
651         |       DT_KEY_RESTRICT { $$ = dt_decl_attr(DT_DA_RESTRICT); }
652         |       DT_KEY_VOLATILE { $$ = dt_decl_attr(DT_DA_VOLATILE); }
653         ;

655 struct_or_union_specifier:
656                 struct_or_union_definition struct_declaration_list '}' {
657                         $$ = dt_scope_pop();
658                 }
659         |       struct_or_union DT_TOK_IDENT { $$ = dt_decl_spec($1, $2); }
660         |       struct_or_union DT_TOK_TNAME { $$ = dt_decl_spec($1, $2); }
661         ;

663 struct_or_union_definition:
664                 struct_or_union '{' { dt_decl_sou($1, NULL); }
665         |       struct_or_union DT_TOK_IDENT '{' { dt_decl_sou($1, $2); }
666         |       struct_or_union DT_TOK_TNAME '{' { dt_decl_sou($1, $2); }
667         ;

669 struct_or_union:
670                 DT_KEY_STRUCT { $$ = CTF_K_STRUCT; }
671         |       DT_KEY_UNION { $$ = CTF_K_UNION; }
672         ;

674 struct_declaration_list:
675                 struct_declaration
676         |       struct_declaration_list struct_declaration
677         ;

679 init_declarator_list:
680                 init_declarator
681         |       init_declarator_list DT_TOK_COMMA init_declarator {
682                         $$ = LINK($1, $3);
683                 }
684         ;

686 init_declarator:
687                 declarator {
688                         $$ = dt_node_decl();
689                         dt_decl_reset();
690                 }
691         ;

693 struct_declaration:
694                 specifier_qualifier_list struct_declarator_list ';' {
695                         dt_decl_free(dt_decl_pop());
696                 }
697         ;

699 specifier_qualifier_list:
700                 type_specifier
701         |       type_specifier specifier_qualifier_list { $$ = $2; }
702         |       type_qualifier
703         |       type_qualifier specifier_qualifier_list { $$ = $2; }
704         ;

706 struct_declarator_list:
707                 struct_declarator
708         |       struct_declarator_list DT_TOK_COMMA struct_declarator
709         ;

711 struct_declarator:
712                 declarator { dt_decl_member(NULL); }
713         |       DT_TOK_COLON constant_expression { dt_decl_member($2); }
714         |       declarator DT_TOK_COLON constant_expression {
```



```
715                         dt_decl_member($3);
716                 }
717         ;

719 enum_specifier:
720                 enum_definition enumerator_list '}' { $$ = dt_scope_pop(); }
721         |       DT_KEY_ENUM DT_TOK_IDENT { $$ = dt_decl_spec(CTF_K_ENUM, $2); }
722         |       DT_KEY_ENUM DT_TOK_TNAME { $$ = dt_decl_spec(CTF_K_ENUM, $2); }
723         ;

725 enum_definition:
726                 DT_KEY_ENUM '{' { dt_decl_enum(NULL); }
727         |       DT_KEY_ENUM DT_TOK_IDENT '{' { dt_decl_enum($2); }
728         |       DT_KEY_ENUM DT_TOK_TNAME '{' { dt_decl_enum($2); }
729         ;

731 enumerator_list:
732                 enumerator
733         |       enumerator_list DT_TOK_COMMA enumerator
734         ;

736 enumerator:    DT_TOK_IDENT { dt_decl_enumerator($1, NULL); }
737         |       DT_TOK_IDENT DT_TOK_ASGN expression {
738                         dt_decl_enumerator($1, $3);
739                 }
740         ;

742 declarator:    direct_declarator
743         |       pointer direct_declarator
744         ;

746 direct_declarator:
747                 DT_TOK_IDENT { $$ = dt_decl_ident($1); }
748         |       lparen declarator DT_TOK_RPAR { $$ = $2; }
749         |       direct_declarator array { dt_decl_array($2); }
750         |       direct_declarator function { dt_decl_func($1, $2); }
751         ;

753 lparen:        DT_TOK_LPAR { dt_decl_top()->dd_attr |= DT_DA_PAREN; }
754         ;

756 pointer:       DT_TOK_MUL { $$ = dt_decl_ptr(); }
757         |       DT_TOK_MUL type_qualifier_list { $$ = dt_decl_ptr(); }
758         |       DT_TOK_MUL pointer { $$ = dt_decl_ptr(); }
759         |       DT_TOK_MUL type_qualifier_list pointer { $$ = dt_decl_ptr(); }
760         ;

762 type_qualifier_list:
763                 type_qualifier
764         |       type_qualifier_list type_qualifier { $$ = $2; }
765         ;

767 parameter_type_list:
768                 parameter_list
769         |       DT_TOK_ELLIPSIS { $$ = dt_node_vatype(); }
770         |       parameter_list DT_TOK_COMMA DT_TOK_ELLIPSIS {
771                         $$ = LINK($1, dt_node_vatype());
772                 }
773         ;

775 parameter_list: parameter_declaration
776         |       parameter_list DT_TOK_COMMA parameter_declaration {
777                         $$ = LINK($1, $3);
778                 }
779         ;
```

```
781 parameter_declaration:
782                 parameter_declaration_specifiers {
783                         $$ = dt_node_type(NULL);
784                 }
785         |       parameter_declaration_specifiers declarator {
786                         $$ = dt_node_type(NULL);
787                 }
788         |       parameter_declaration_specifiers abstract_declarator {
789                         $$ = dt_node_type(NULL);
790                 }
791         ;

793 type_name:      specifier_qualifier_list {
794                         $$ = dt_decl_pop();
795                 }
796         |       specifier_qualifier_list abstract_declarator {
797                         $$ = dt_decl_pop();
798                 }
799         ;

801 abstract_declarator:
802                 pointer
803         |       direct_abstract_declarator
804         |       pointer direct_abstract_declarator
805         ;

807 direct_abstract_declarator:
808                 lparen abstract_declarator DT_TOK_RPAR { $$ = $2; }
809         |       direct_abstract_declarator array { dt_decl_array($2); }
810         |       array { dt_decl_array($1); $$ = NULL; }
811         |       direct_abstract_declarator function { dt_decl_func($1, $2); }
812         |       function { dt_decl_func(NULL, $1); }
813         ;

815 array:          DT_TOK_LBRAC { dt_scope_push(NULL, CTF_ERR); }
816                     array_parameters DT_TOK_RBRAC {
817                         dt_scope_pop();
818                         $$ = $3;
819                 }
820         ;

822 array_parameters:
823                 /* empty */             { $$ = NULL; }
824         |       constant_expression     { $$ = $1; }
825         |       parameter_type_list     { $$ = $1; }
826         ;

828 function:       DT_TOK_LPAR { dt_scope_push(NULL, CTF_ERR); }
829                     function_parameters DT_TOK_RPAR {
830                         dt_scope_pop();
831                         $$ = $3;
832                 }
833         ;

835 function_parameters:
836                 /* empty */             { $$ = NULL; }
837         |       parameter_type_list     { $$ = $1; }
838         ;

840 %%
```

```
**********************************************************
   26807 Tue Jan 14 16:48:54 2014
new/usr/src/lib/libdtrace/common/dt_ident.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 * Copyright (c) 2013 Joyent, Inc. All rights reserved.
#endif /* ! codereview */
 */

#include <sys/sysmacros.h>
#include <strings.h>
#include <stdlib.h>
#include <alloca.h>
#include <assert.h>
#include <errno.h>
#include <ctype.h>
#include <sys/procfs_isa.h>
#include <limits.h>

#include <dt_ident.h>
#include <dt_parser.h>
#include <dt_provider.h>
#include <dt_strtab.h>
#include <dt_impl.h>

/*
 * Common code for cooking an identifier that uses a typed signature list (we
 * use this for associative arrays and functions).  If the argument list is
 * of the same length and types, then return the return type.  Otherwise
 * print an appropriate compiler error message and abort the compile.
 */
static void
dt_idcook_sign(dt_node_t *dnp, dt_ident_t *idp,
    int argc, dt_node_t *args, const char *prefix, const char *suffix)
{
        dt_idsig_t *isp = idp->di_data;
        int i, compat, mismatch, arglimit, iskey;
```



```
        char n1[DT_TYPE_NAMELEN];
        char n2[DT_TYPE_NAMELEN];

        iskey = idp->di_kind == DT_IDENT_ARRAY || idp->di_kind == DT_IDENT_AGG;

        if (isp->dis_varargs >= 0) {
                mismatch = argc < isp->dis_varargs;
                arglimit = isp->dis_varargs;
        } else if (isp->dis_optargs >= 0) {
                mismatch = (argc < isp->dis_optargs || argc > isp->dis_argc);
                arglimit = argc;
        } else {
                mismatch = argc != isp->dis_argc;
                arglimit = isp->dis_argc;
        }

        if (mismatch) {
                xyerror(D_PROTO_LEN, "%s%s%s prototype mismatch: %d %s%s"
                    "passed, %s%d expected\n", prefix, idp->di_name, suffix,
                    argc, iskey ? "key" : "arg", argc == 1 ? " " : "s ",
                    isp->dis_optargs >= 0 ? "at least " : "",
                    isp->dis_optargs >= 0 ? isp->dis_optargs : arglimit);
        }

        for (i = 0; i < arglimit; i++, args = args->dn_list) {
                if (isp->dis_args[i].dn_ctfp != NULL)
                        compat = dt_node_is_argcompat(&isp->dis_args[i], args);
                else
                        compat = 1; /* "@" matches any type */

                if (!compat) {
                        xyerror(D_PROTO_ARG,
                            "%s%s%s %s #%d is incompatible with "
                            "prototype:\n\tprototype: %s\n\t%9s: %s\n",
                            prefix, idp->di_name, suffix,
                            iskey ? "key" : "argument", i + 1,
                            dt_node_type_name(&isp->dis_args[i], n1,
                            sizeof (n1)),
                            iskey ? "key" : "argument",
                            dt_node_type_name(args, n2, sizeof (n2)));
                }
        }

        dt_node_type_assign(dnp, idp->di_ctfp, idp->di_type, B_FALSE);
        dt_node_type_assign(dnp, idp->di_ctfp, idp->di_type);
}

/*
 * Cook an associative array identifier.  If this is the first time we are
 * cooking this array, create its signature based on the argument list.
 * Otherwise validate the argument list against the existing signature.
 */
static void
dt_idcook_assc(dt_node_t *dnp, dt_ident_t *idp, int argc, dt_node_t *args)
{
        if (idp->di_data == NULL) {
                dt_idsig_t *isp = idp->di_data = malloc(sizeof (dt_idsig_t));
                char n[DT_TYPE_NAMELEN];
                int i;

                if (isp == NULL)
                        longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

                isp->dis_varargs = -1;
                isp->dis_optargs = -1;
```

```
122                 isp->dis_argc = argc;
123                 isp->dis_args = NULL;
124                 isp->dis_auxinfo = 0;

126                 if (argc != 0 && (isp->dis_args = calloc(argc,
127                     sizeof (dt_node_t))) == NULL) {
128                         idp->di_data = NULL;
129                         free(isp);
130                         longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
131                 }

133                 /*
134                  * If this identifier has not been explicitly declared earlier,
135                  * set the identifier's base type to be our special type <DYN>.
136                  * If this ident is an aggregation, it will remain as is.  If
137                  * this ident is an associative array, it will be reassigned
138                  * based on the result type of the first assignment statement.
139                  */
140                 if (!(idp->di_flags & DT_IDFLG_DECL)) {
141                         idp->di_ctfp = DT_DYN_CTFP(yypcb->pcb_hdl);
142                         idp->di_type = DT_DYN_TYPE(yypcb->pcb_hdl);
143                 }

145                 for (i = 0; i < argc; i++, args = args->dn_list) {
146                         if (dt_node_is_dynamic(args) || dt_node_is_void(args)) {
147                                 xyerror(D_KEY_TYPE, "%s expression may not be "
148                                     "used as %s index: key #%d\n",
149                                     dt_node_type_name(args, n, sizeof (n)),
150                                     dt_idkind_name(idp->di_kind), i + 1);
151                         }

153                         dt_node_type_propagate(args, &isp->dis_args[i]);
154                         isp->dis_args[i].dn_list = &isp->dis_args[i + 1];
155                 }

157                 if (argc != 0)
158                         isp->dis_args[argc - 1].dn_list = NULL;

160                 dt_node_type_assign(dnp, idp->di_ctfp, idp->di_type, B_FALSE);
 83                 dt_node_type_assign(dnp, idp->di_ctfp, idp->di_type);

162         } else {
163                 dt_idcook_sign(dnp, idp, argc, args,
164                     idp->di_kind == DT_IDENT_AGG ? "@" : "", "[ ]");
165         }
166 }

168 /*
169  * Cook a function call.  If this is the first time we are cooking this
170  * identifier, create its type signature based on predefined prototype stored
171  * in di_iarg.  We then validate the argument list against this signature.
172  */
173 static void
174 dt_idcook_func(dt_node_t *dnp, dt_ident_t *idp, int argc, dt_node_t *args)
175 {
176         if (idp->di_data == NULL) {
177                 dtrace_hdl_t *dtp = yypcb->pcb_hdl;
178                 dtrace_typeinfo_t dtt;
179                 dt_idsig_t *isp;
180                 char *s, *p1, *p2;
181                 int i = 0;

183                 assert(idp->di_iarg != NULL);
184                 s = strdupa(idp->di_iarg);

186                 if ((p2 = strrchr(s, ')')) != NULL)
```



```
187                         *p2 = '\0'; /* mark end of parameter list string */

189                 if ((p1 = strchr(s, '(')) != NULL)
190                         *p1++ = '\0'; /* mark end of return type string */

192                 if (p1 == NULL || p2 == NULL) {
193                         xyerror(D_UNKNOWN, "internal error: malformed entry "
194                             "for built-in function %s\n", idp->di_name);
195                 }

197                 for (p2 = p1; *p2 != '\0'; p2++) {
198                         if (!isspace(*p2)) {
199                                 i++;
200                                 break;
201                         }
202                 }

204                 for (p2 = strchr(p2, ','); p2++ != NULL; i++)
205                         p2 = strchr(p2, ',');

207                 /*
208                  * We first allocate a new ident signature structure with the
209                  * appropriate number of argument entries, and then look up
210                  * the return type and store its CTF data in di_ctfp/type.
211                  */
212                 if ((isp = idp->di_data = malloc(sizeof (dt_idsig_t))) == NULL)
213                         longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

215                 isp->dis_varargs = -1;
216                 isp->dis_optargs = -1;
217                 isp->dis_argc = i;
218                 isp->dis_args = NULL;
219                 isp->dis_auxinfo = 0;

221                 if (i != 0 && (isp->dis_args = calloc(i,
222                     sizeof (dt_node_t))) == NULL) {
223                         idp->di_data = NULL;
224                         free(isp);
225                         longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
226                 }

228                 if (dt_type_lookup(s, &dtt) == -1) {
229                         xyerror(D_UNKNOWN, "failed to resolve type of %s (%s):"
230                             " %s\n", idp->di_name, s,
231                             dtrace_errmsg(dtp, dtrace_errno(dtp)));
232                 }

234                 if (idp->di_kind == DT_IDENT_AGGFUNC) {
235                         idp->di_ctfp = DT_DYN_CTFP(dtp);
236                         idp->di_type = DT_DYN_TYPE(dtp);
237                 } else {
238                         idp->di_ctfp = dtt.dtt_ctfp;
239                         idp->di_type = dtt.dtt_type;
240                 }

242                 /*
243                  * For each comma-delimited parameter in the prototype string,
244                  * we look up the corresponding type and store its CTF data in
245                  * the corresponding location in dis_args[].  We also recognize
246                  * the special type string "@" to indicate that the specified
247                  * parameter may be a D expression of *any* type (represented
248                  * as a dis_args[] element with ctfp = NULL, type == CTF_ERR).
249                  * If a varargs "..." is present, we record the argument index
250                  * in dis_varargs for the benefit of dt_idcook_sign(), above.
251                  * If the type of an argument is enclosed in square brackets
252                  * (e.g. "[int]"), the argument is considered optional:  the
```

```
253                  * argument may be absent, but if it is present, it must be of
254                  * the specified type.  Note that varargs may not optional,
255                  * optional arguments may not follow varargs, and non-optional
256                  * arguments may not follow optional arguments.
257                  */
258                 for (i = 0; i < isp->dis_argc; i++, p1 = p2) {
259                         while (isspace(*p1))
260                                 p1++; /* skip leading whitespace */
261
262                         if ((p2 = strchr(p1, ',')) == NULL)
263                                 p2 = p1 + strlen(p1);
264                         else
265                                 *p2++ = '\0';
266
267                         if (strcmp(p1, "@") == 0 || strcmp(p1, "...") == 0) {
268                                 isp->dis_args[i].dn_ctfp = NULL;
269                                 isp->dis_args[i].dn_type = CTF_ERR;
270                                 if (*p1 == '.')
271                                         isp->dis_varargs = i;
272                                 continue;
273                         }
274
275                         if (*p1 == '[' && p1[strlen(p1) - 1] == ']') {
276                                 if (isp->dis_varargs != -1) {
277                                         xyerror(D_UNKNOWN, "optional arg#%d "
278                                             "may not follow variable arg#%d\n",
279                                             i + 1, isp->dis_varargs + 1);
280                                 }
281
282                                 if (isp->dis_optargs == -1)
283                                         isp->dis_optargs = i;
284
285                                 p1[strlen(p1) - 1] = '\0';
286                                 p1++;
287                         } else if (isp->dis_optargs != -1) {
288                                 xyerror(D_UNKNOWN, "required arg#%d may not "
289                                     "follow optional arg#%d\n", i + 1,
290                                     isp->dis_optargs + 1);
291                         }
292
293                         if (dt_type_lookup(p1, &dtt) == -1) {
294                                 xyerror(D_UNKNOWN, "failed to resolve type of "
295                                     "%s arg#%d (%s): %s\n", idp->di_name, i + 1,
296                                     p1, dtrace_errmsg(dtp, dtrace_errno(dtp)));
297                         }
298
299                         dt_node_type_assign(&isp->dis_args[i],
300                             dtt.dtt_ctfp, dtt.dtt_type, B_FALSE);
223                             dtt.dtt_ctfp, dtt.dtt_type);
301                 }
302         }
303
304         dt_idcook_sign(dnp, idp, argc, args, "", "( )");
305 }
306
307 /*
308  * Cook a reference to the dynamically typed args[] array.  We verify that the
309  * reference is using a single integer constant, and then construct a new ident
310  * representing the appropriate type or translation specifically for this node.
311  */
312 static void
313 dt_idcook_args(dt_node_t *dnp, dt_ident_t *idp, int argc, dt_node_t *ap)
314 {
315         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
316         dt_probe_t *prp = yypcb->pcb_probe;
```



```
317
318         dt_node_t tag, *nnp, *xnp;
319         dt_xlator_t *dxp;
320         dt_ident_t *xidp;
321
322         char n1[DT_TYPE_NAMELEN];
323         char n2[DT_TYPE_NAMELEN];
324
325         if (argc != 1) {
326                 xyerror(D_PROTO_LEN, "%s[ ] prototype mismatch: %d arg%s"
327                     "passed, 1 expected\n", idp->di_name, argc,
328                     argc == 1 ? " " : "s ");
329         }
330
331         if (ap->dn_kind != DT_NODE_INT) {
332                 xyerror(D_PROTO_ARG, "%s[ ] argument #1 is incompatible with "
333                     "prototype:\n\tprototype: %s\n\t argument: %s\n",
334                     idp->di_name, "integer constant",
335                     dt_type_name(ap->dn_ctfp, ap->dn_type, n1, sizeof (n1)));
336         }
337
338         if (yypcb->pcb_pdesc == NULL) {
339                 xyerror(D_ARGS_NONE, "%s[ ] may not be referenced outside "
340                     "of a probe clause\n", idp->di_name);
341         }
342
343         if (prp == NULL) {
344                 xyerror(D_ARGS_MULTI,
345                     "%s[ ] may not be referenced because probe description %s "
346                     "matches an unstable set of probes\n", idp->di_name,
347                     dtrace_desc2str(yypcb->pcb_pdesc, n1, sizeof (n1)));
348         }
349
350         if (ap->dn_value >= prp->pr_argc) {
351                 xyerror(D_ARGS_IDX, "index %lld is out of range for %s %s[ ]\n",
352                     (longlong_t)ap->dn_value, dtrace_desc2str(yypcb->pcb_pdesc,
353                     n1, sizeof (n1)), idp->di_name);
354         }
355
356         /*
357          * Look up the native and translated argument types for the probe.
358          * If no translation is needed, these will be the same underlying node.
359          * If translation is needed, look up the appropriate translator.  Once
360          * we have the appropriate node, create a new dt_ident_t for this node,
361          * assign it the appropriate attributes, and set the type of 'dnp'.
362          */
363         xnp = prp->pr_xargv[ap->dn_value];
364         nnp = prp->pr_nargv[prp->pr_mapping[ap->dn_value]];
365
366         if (xnp->dn_type == CTF_ERR) {
367                 xyerror(D_ARGS_TYPE, "failed to resolve translated type for "
368                     "%s[%lld]\n", idp->di_name, (longlong_t)ap->dn_value);
369         }
370
371         if (nnp->dn_type == CTF_ERR) {
372                 xyerror(D_ARGS_TYPE, "failed to resolve native type for "
373                     "%s[%lld]\n", idp->di_name, (longlong_t)ap->dn_value);
374         }
375
376         if (dtp->dt_xlatemode == DT_XL_STATIC && (
377             nnp == xnp || dt_node_is_argcompat(nnp, xnp))) {
378                 dnp->dn_ident = dt_ident_create(idp->di_name, idp->di_kind,
379                     idp->di_flags | DT_IDFLG_ORPHAN, idp->di_id, idp->di_attr,
380                     idp->di_vers, idp->di_ops, idp->di_iarg, idp->di_gen);
381
382                 if (dnp->dn_ident == NULL)
383                         longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
```

```
385                 dt_node_type_assign(dnp,
386                     prp->pr_argv[ap->dn_value].dtt_ctfp,
387                     prp->pr_argv[ap->dn_value].dtt_type,
388                     prp->pr_argv[ap->dn_value].dtt_flags & DTT_FL_USER ?
389                     B_TRUE : B_FALSE);
310                     prp->pr_argv[ap->dn_value].dtt_type);

391         } else if ((dxp = dt_xlator_lookup(dtp,
392             nnp, xnp, DT_XLATE_FUZZY)) != NULL || (
393             dxp = dt_xlator_lookup(dtp, dt_probe_tag(prp, ap->dn_value, &tag),
394             xnp, DT_XLATE_EXACT | DT_XLATE_EXTERN)) != NULL) {

396                 xidp = dt_xlator_ident(dxp, xnp->dn_ctfp, xnp->dn_type);

398                 dnp->dn_ident = dt_ident_create(idp->di_name, xidp->di_kind,
399                     xidp->di_flags | DT_IDFLG_ORPHAN, idp->di_id, idp->di_attr,
400                     idp->di_vers, idp->di_ops, idp->di_iarg, idp->di_gen);

402                 if (dnp->dn_ident == NULL)
403                         longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

405                 if (dt_xlator_dynamic(dxp))
406                         dxp->dx_arg = (int)ap->dn_value;

408                 /*
409                  * Propagate relevant members from the translator's internal
410                  * dt_ident_t.  This code must be kept in sync with the state
411                  * that is initialized for idents in dt_xlator_create().
412                  */
413                 dnp->dn_ident->di_data = xidp->di_data;
414                 dnp->dn_ident->di_ctfp = xidp->di_ctfp;
415                 dnp->dn_ident->di_type = xidp->di_type;

417                 dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp),
418                     B_FALSE);
338                 dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp));

420         } else {
421                 xyerror(D_ARGS_XLATOR, "translator for %s[%lld] from %s to %s "
422                     "is not defined\n", idp->di_name, (longlong_t)ap->dn_value,
423                     dt_node_type_name(nnp, n1, sizeof (n1)),
424                     dt_node_type_name(xnp, n2, sizeof (n2)));
425         }

427         assert(dnp->dn_ident->di_flags & DT_IDFLG_ORPHAN);
428         assert(dnp->dn_ident->di_id == idp->di_id);
429 }

431 static void
432 dt_idcook_regs(dt_node_t *dnp, dt_ident_t *idp, int argc, dt_node_t *ap)
433 {
434         dtrace_typeinfo_t dtt;
435         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
436         char n[DT_TYPE_NAMELEN];

438         if (argc != 1) {
439                 xyerror(D_PROTO_LEN, "%s[ ] prototype mismatch: %d arg%s"
440                     "passed, 1 expected\n", idp->di_name,
441                     argc, argc == 1 ? " " : "s ");
442         }

444         if (ap->dn_kind != DT_NODE_INT) {
445                 xyerror(D_PROTO_ARG, "%s[ ] argument #1 is incompatible with "
446                     "prototype:\n\tprototype: %s\n\t argument: %s\n",
447                     idp->di_name, "integer constant",
```



```
448                     dt_type_name(ap->dn_ctfp, ap->dn_type, n, sizeof (n)));
449         }

451         if ((ap->dn_flags & DT_NF_SIGNED) && (int64_t)ap->dn_value < 0) {
452                 xyerror(D_REGS_IDX, "index %lld is out of range for array %s\n",
453                     (longlong_t)ap->dn_value, idp->di_name);
454         }

456         if (dt_type_lookup("uint64_t", &dtt) == -1) {
457                 xyerror(D_UNKNOWN, "failed to resolve type of %s: %s\n",
458                     idp->di_name, dtrace_errmsg(dtp, dtrace_errno(dtp)));
459         }

461         idp->di_ctfp = dtt.dtt_ctfp;
462         idp->di_type = dtt.dtt_type;

464         dt_node_type_assign(dnp, idp->di_ctfp, idp->di_type, B_FALSE);
384         dt_node_type_assign(dnp, idp->di_ctfp, idp->di_type);
465 }

467 /*ARGSUSED*/
468 static void
469 dt_idcook_type(dt_node_t *dnp, dt_ident_t *idp, int argc, dt_node_t *args)
470 {
471         if (idp->di_type == CTF_ERR) {
472                 dtrace_hdl_t *dtp = yypcb->pcb_hdl;
473                 dtrace_typeinfo_t dtt;

475                 if (dt_type_lookup(idp->di_iarg, &dtt) == -1) {
476                         xyerror(D_UNKNOWN,
477                             "failed to resolve type %s for identifier %s: %s\n",
478                             (const char *)idp->di_iarg, idp->di_name,
479                             dtrace_errmsg(dtp, dtrace_errno(dtp)));
480                 }

482                 idp->di_ctfp = dtt.dtt_ctfp;
483                 idp->di_type = dtt.dtt_type;
484         }

486         dt_node_type_assign(dnp, idp->di_ctfp, idp->di_type, B_FALSE);
406         dt_node_type_assign(dnp, idp->di_ctfp, idp->di_type);
487 }

489 /*ARGSUSED*/
490 static void
491 dt_idcook_thaw(dt_node_t *dnp, dt_ident_t *idp, int argc, dt_node_t *args)
492 {
493         if (idp->di_ctfp != NULL && idp->di_type != CTF_ERR)
494                 dt_node_type_assign(dnp, idp->di_ctfp, idp->di_type, B_FALSE);
414                 dt_node_type_assign(dnp, idp->di_ctfp, idp->di_type);
495 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   30764 Tue Jan 14 16:48:54 2014
new/usr/src/lib/libdtrace/common/dt_impl.h
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
______unchanged_portion_omitted_

typedef struct dt_module {
        dt_list_t dm_list;      /* list forward/back pointers */
        char dm_name[DTRACE_MODNAMELEN]; /* string name of module */
        char dm_file[MAXPATHLEN]; /* file path of module (if any) */
        struct dt_module *dm_next; /* pointer to next module in hash chain */
        const dt_modops_t *dm_ops; /* pointer to data model's ops vector */
        Elf *dm_elf;            /* libelf handle for module object */
        objfs_info_t dm_info;   /* object filesystem private info */
        ctf_sect_t dm_symtab;   /* symbol table for module */
        ctf_sect_t dm_strtab;   /* string table for module */
        ctf_sect_t dm_ctdata;   /* CTF data for module */
        ctf_file_t *dm_ctfp;    /* CTF container handle */
        uint_t *dm_symbuckets;  /* symbol table hash buckets (chain indices) */
        dt_sym_t *dm_symchains; /* symbol table hash chains buffer */
        void *dm_asmap;         /* symbol pointers sorted by value */
        uint_t dm_symfree;      /* index of next free hash element */
        uint_t dm_nsymbuckets;  /* number of elements in bucket array */
        uint_t dm_nsymelems;    /* number of elements in hash table */
        uint_t dm_asrsv;        /* actual reserved size of dm_asmap */
        uint_t dm_aslen;        /* number of entries in dm_asmap */
        uint_t dm_flags;        /* module flags (see below) */
        int dm_modid;           /* modinfo(1M) module identifier */
        GElf_Addr dm_text_va;   /* virtual address of text section */
        GElf_Xword dm_text_size; /* size in bytes of text section */
        GElf_Addr dm_data_va;   /* virtual address of data section */
        GElf_Xword dm_data_size; /* size in bytes of data section */
        GElf_Addr dm_bss_va;    /* virtual address of BSS */
        GElf_Xword dm_bss_size; /* size in bytes of BSS */
        dt_idhash_t *dm_extern; /* external symbol definitions */
        pid_t dm_pid;           /* pid for this module */
        uint_t dm_nctflibs;     /* number of ctf children libraries */
        ctf_file_t **dm_libctfp; /* process library ctf pointers */
        char **dm_libctfn;      /* names of process ctf containers */
#endif /* ! codereview */
} dt_module_t;

#define DT_DM_LOADED    0x1     /* module symbol and type data is loaded */
#define DT_DM_KERNEL    0x2     /* module is associated with a kernel object */
#define DT_DM_PRIMARY   0x4     /* module is a krtld primary kernel object */

typedef struct dt_provmod {
        char *dp_name;                          /* name of provider module */
        struct dt_provmod *dp_next;             /* next module */
} dt_provmod_t;

typedef struct dt_ahashent {
        struct dt_ahashent *dtahe_prev;         /* prev on hash chain */
        struct dt_ahashent *dtahe_next;         /* next on hash chain */
        struct dt_ahashent *dtahe_prevall;      /* prev on list of all */
        struct dt_ahashent *dtahe_nextall;      /* next on list of all */
        uint64_t dtahe_hashval;                 /* hash value */
        size_t dtahe_size;                      /* size of data */
        dtrace_aggdata_t dtahe_data;            /* data */
        void (*dtahe_aggregate)(int64_t *, int64_t *, size_t); /* function */
```



```
} dt_ahashent_t;

typedef struct dt_ahash {
        dt_ahashent_t   **dtah_hash;            /* hash table */
        dt_ahashent_t   *dtah_all;              /* list of all elements */
        size_t          dtah_size;              /* size of hash table */
} dt_ahash_t;

typedef struct dt_aggregate {
        dtrace_bufdesc_t dtat_buf;      /* buf aggregation snapshot */
        int dtat_flags;                 /* aggregate flags */
        processorid_t dtat_ncpus;       /* number of CPUs in aggregate */
        processorid_t *dtat_cpus;       /* CPUs in aggregate */
        processorid_t dtat_ncpu;        /* size of dtat_cpus array */
        processorid_t dtat_maxcpu;      /* maximum number of CPUs */
        dt_ahash_t dtat_hash;           /* aggregate hash table */
} dt_aggregate_t;

typedef struct dt_print_aggdata {
        dtrace_hdl_t *dtpa_dtp;         /* pointer to libdtrace handle */
        dtrace_aggvarid_t dtpa_id;      /* aggregation variable of interest */
        FILE *dtpa_fp;                  /* file pointer */
        int dtpa_allunprint;            /* print only unprinted aggregations */
} dt_print_aggdata_t;

typedef struct dt_dirpath {
        dt_list_t dir_list;             /* linked-list forward/back pointers */
        char *dir_path;                 /* directory pathname */
} dt_dirpath_t;

typedef struct dt_lib_depend {
        dt_list_t dtld_deplist;         /* linked-list forward/back pointers */
        char *dtld_library;             /* library name */
        char *dtld_libpath;             /* library pathname */
        uint_t dtld_finish;             /* completion time in tsort for lib */
        uint_t dtld_start;              /* starting time in tsort for lib */
        uint_t dtld_loaded;             /* boolean: is this library loaded */
        dt_list_t dtld_dependencies;    /* linked-list of lib dependencies */
        dt_list_t dtld_dependents;      /* linked-list of lib dependents */
} dt_lib_depend_t;

typedef uint32_t dt_version_t;          /* encoded version (see below) */

struct dtrace_hdl {
        const dtrace_vector_t *dt_vector; /* library vector, if vectored open */
        void *dt_varg;  /* vector argument, if vectored open */
        dtrace_conf_t dt_conf;  /* DTrace driver configuration profile */
        char dt_errmsg[BUFSIZ]; /* buffer for formatted syntax error msgs */
        const char *dt_errtag;  /* tag used with last call to dt_set_errmsg() */
        dt_pcb_t *dt_pcb;       /* pointer to current parsing control block */
        ulong_t dt_gen;         /* compiler generation number */
        dt_list_t dt_programs;  /* linked list of dtrace_prog_t's */
        dt_list_t dt_xlators;   /* linked list of dt_xlator_t's */
        struct dt_xlator **dt_xlatormap; /* dt_xlator_t's indexed by dx_id */
        id_t dt_xlatorid;       /* next dt_xlator_t id to assign */
        dt_ident_t *dt_externs; /* linked list of external symbol identifiers */
        dt_idhash_t *dt_macros; /* hash table of macro variable identifiers */
        dt_idhash_t *dt_aggs;   /* hash table of aggregation identifiers */
        dt_idhash_t *dt_globals; /* hash table of global identifiers */
        dt_idhash_t *dt_tls;    /* hash table of thread-local identifiers */
        dt_list_t dt_modlist;   /* linked list of dt_module_t's */
        dt_module_t **dt_mods;  /* hash table of dt_module_t's */
        uint_t dt_modbuckets;   /* number of module hash buckets */
        uint_t dt_nmods;        /* number of modules in hash and list */
        dt_provmod_t *dt_provmod; /* linked list of provider modules */
        dt_module_t *dt_exec;   /* pointer to executable module */
```

```
227         dt_module_t *dt_rtld;   /* pointer to run-time linker module */
228         dt_module_t *dt_cdefs;  /* pointer to C dynamic type module */
229         dt_module_t *dt_ddefs;  /* pointer to D dynamic type module */
230         dt_list_t dt_provlist;  /* linked list of dt_provider_t's */
231         struct dt_provider **dt_provs; /* hash table of dt_provider_t's */
232         uint_t dt_provbuckets;  /* number of provider hash buckets */
233         uint_t dt_nprovs;       /* number of providers in hash and list */
234         dt_proc_hash_t *dt_procs; /* hash table of grabbed process handles */
235         char **dt_proc_env;     /* additional environment variables */
236         dt_intdesc_t dt_ints[6]; /* cached integer type descriptions */
237         ctf_id_t dt_type_func;  /* cached CTF identifier for function type */
238         ctf_id_t dt_type_fptr;  /* cached CTF identifier for function pointer */
239         ctf_id_t dt_type_str;   /* cached CTF identifier for string type */
240         ctf_id_t dt_type_dyn;   /* cached CTF identifier for <DYN> type */
241         ctf_id_t dt_type_stack; /* cached CTF identifier for stack type */
242         ctf_id_t dt_type_symaddr; /* cached CTF identifier for _symaddr type */
243         ctf_id_t dt_type_usymaddr; /* cached CTF ident. for _usymaddr type */
244         size_t dt_maxprobe;     /* max enabled probe ID */
245         dtrace_eprobedesc_t **dt_edesc; /* enabled probe descriptions */
246         dtrace_probedesc_t **dt_pdesc; /* probe descriptions for enabled prbs */
247         size_t dt_maxagg;       /* max aggregation ID */
248         dtrace_aggdesc_t **dt_aggdesc; /* aggregation descriptions */
249         int dt_maxformat;       /* max format ID */
250         void **dt_formats;      /* pointer to format array */
251         int dt_maxstrdata;      /* max strdata ID */
252         char **dt_strdata;      /* pointer to strdata array */
253         dt_aggregate_t dt_aggregate; /* aggregate */
254         dt_pq_t *dt_bufq;       /* CPU-specific data queue */
255         struct dt_pfdict *dt_pfdict; /* dictionary of printf conversions */
256         dt_version_t dt_vmax;   /* optional ceiling on program API binding */
257         dtrace_attribute_t dt_amin; /* optional floor on program attributes */
258         char *dt_cpp_path;      /* pathname of cpp(1) to invoke if needed */
259         char **dt_cpp_argv;     /* argument vector for exec'ing cpp(1) */
260         int dt_cpp_argc;        /* count of initialized cpp(1) arguments */
261         int dt_cpp_args;        /* size of dt_cpp_argv[] array */
262         char *dt_ld_path;       /* pathname of ld(1) to invoke if needed */
263         dt_list_t dt_lib_path;  /* linked-list forming library search path */
264         uint_t dt_lazyload;     /* boolean:  set via -xlazyload */
265         uint_t dt_droptags;     /* boolean:  set via -xdroptags */
266         uint_t dt_active;       /* boolean:  set once tracing is active */
267         uint_t dt_stopped;      /* boolean:  set once tracing is stopped */
268         processorid_t dt_beganon; /* CPU that executed BEGIN probe (if any) */
269         processorid_t dt_endedon; /* CPU that executed END probe (if any) */
270         uint_t dt_oflags;       /* dtrace open-time options (see dtrace.h) */
271         uint_t dt_cflags;       /* dtrace compile-time options (see dtrace.h) */
272         uint_t dt_dflags;       /* dtrace link-time options (see dtrace.h) */
273         uint_t dt_prcmode;      /* dtrace process create mode (see dt_proc.h) */
274         uint_t dt_linkmode;     /* dtrace symbol linking mode (see below) */
275         uint_t dt_linktype;     /* dtrace link output file type (see below) */
276         uint_t dt_xlatemode;    /* dtrace translator linking mode (see below) */
277         uint_t dt_stdcmode;     /* dtrace stdc compatibility mode (see below) */
278         uint_t dt_treedump;     /* dtrace tree debug bitmap (see below) */
279         uint64_t dt_options[DTRACEOPT_MAX]; /* dtrace run-time options */
280         int dt_version;         /* library version requested by client */
281         int dt_ctferr;          /* error resulting from last CTF failure */
282         int dt_errno;           /* error resulting from last failed operation */
283         int dt_fd;              /* file descriptor for dtrace pseudo-device */
284         int dt_ftfd;            /* file descriptor for fasttrap pseudo-device */
285         int dt_fterr;           /* saved errno from failed open of dt_ftfd */
286         int dt_cdefs_fd;        /* file descriptor for C CTF debugging cache */
287         int dt_ddefs_fd;        /* file descriptor for D CTF debugging cache */
288         int dt_stdout_fd;       /* file descriptor for saved stdout */
289         dtrace_handle_err_f *dt_errhdlr; /* error handler, if any */
290         void *dt_errarg;        /* error handler argument */
291         dtrace_prog_t *dt_errprog; /* error handler program, if any */
292         dtrace_handle_drop_f *dt_drophdlr; /* drop handler, if any */
```



```
293         void *dt_droparg;       /* drop handler argument */
294         dtrace_handle_proc_f *dt_prochdlr; /* proc handler, if any */
295         void *dt_procarg;       /* proc handler argument */
296         dtrace_handle_setopt_f *dt_setopthdlr; /* setopt handler, if any */
297         void *dt_setoptarg;     /* setopt handler argument */
298         dtrace_status_t dt_status[2]; /* status cache */
299         int dt_statusgen;       /* current status generation */
300         hrtime_t dt_laststatus; /* last status */
301         hrtime_t dt_lastswitch; /* last switch of buffer data */
302         hrtime_t dt_lastagg;    /* last snapshot of aggregation data */
303         char *dt_sprintf_buf;   /* buffer for dtrace_sprintf() */
304         int dt_sprintf_buflen;  /* length of dtrace_sprintf() buffer */
305         const char *dt_filetag; /* default filetag for dt_set_errmsg() */
306         char *dt_buffered_buf;  /* buffer for buffered output */
307         size_t dt_buffered_offs; /* current offset into buffered buffer */
308         size_t dt_buffered_size; /* size of buffered buffer */
309         dtrace_handle_buffered_f *dt_bufhdlr; /* buffered handler, if any */
310         void *dt_bufarg;        /* buffered handler argument */
311         dt_dof_t dt_dof;        /* DOF generation buffers (see dt_dof.c) */
312         struct utsname dt_uts;  /* uname(2) information for system */
313         dt_list_t dt_lib_dep;   /* scratch linked-list of lib dependencies */
314         dt_list_t dt_lib_dep_sorted;    /* dependency sorted library list */
315         dtrace_flowkind_t dt_flow;      /* flow kind */
316         const char *dt_prefix;  /* recommended flow prefix */
317         int dt_indent;          /* recommended flow indent */
318         dtrace_epid_t dt_last_epid;     /* most recently consumed EPID */
319         uint64_t dt_last_timestamp;     /* most recently consumed timestamp */
320 };

321 /*
322  * Values for the user arg of the ECB.
323  */
324 #define DT_ECB_DEFAULT          0
325 #define DT_ECB_ERROR            1

326 /*
327  * Values for the dt_linkmode property, which is used by the assembler when
328  * processing external symbol references.  User can set using -xlink=<mode>.
329  */
330 #define DT_LINK_KERNEL  0       /* kernel syms static, user syms dynamic */
331 #define DT_LINK_PRIMARY 1       /* primary kernel syms static, others dynamic */
332 #define DT_LINK_DYNAMIC 2       /* all symbols dynamic */
333 #define DT_LINK_STATIC  3       /* all symbols static */

334 /*
335  * Values for the dt_linktype property, which is used by dtrace_program_link()
336  * to determine the type of output file that is desired by the client.
337  */
338 #define DT_LTYP_ELF     0       /* produce ELF containing DOF */
339 #define DT_LTYP_DOF     1       /* produce stand-alone DOF */

340 /*
341  * Values for the dt_xlatemode property, which is used to determine whether
342  * references to dynamic translators are permitted.  Set using -xlate=<mode>.
343  */
344 #define DT_XL_STATIC    0       /* require xlators to be statically defined */
345 #define DT_XL_DYNAMIC   1       /* produce references to dynamic translators */

346 /*
347  * Values for the dt_stdcmode property, which is used by the compiler when
348  * running cpp to determine the presence and setting of the __STDC__ macro.
349  */
350 #define DT_STDC_XA      0       /* ISO C + K&R C compat w/o ISO: __STDC__=0 */
351 #define DT_STDC_XC      1       /* Strict ISO C: __STDC__=1 */
352 #define DT_STDC_XS      2       /* K&R C: __STDC__ not defined */
353 #define DT_STDC_XT      3       /* ISO C + K&R C compat with ISO: __STDC__=0 */
```

```
/*
 * Macro to test whether a given pass bit is set in the dt_treedump bit-vector.
 * If the bit for pass 'p' is set, the D compiler displays the parse tree for
 * the program by printing it to stderr at the end of compiler pass 'p'.
 */
#define DT_TREEDUMP_PASS(dtp, p)        ((dtp)->dt_treedump & (1 << ((p) - 1)))

/*
 * Macros for accessing the cached CTF container and type ID for the common
 * types "int", "string", and <DYN>, which we need to use frequently in the D
 * compiler.  The DT_INT_* macro relies upon "int" being at index 0 in the
 * _dtrace_ints_* tables in dt_open.c; the others are also set up there.
 */
#define DT_INT_CTFP(dtp)        ((dtp)->dt_ints[0].did_ctfp)
#define DT_INT_TYPE(dtp)        ((dtp)->dt_ints[0].did_type)

#define DT_FUNC_CTFP(dtp)       ((dtp)->dt_ddefs->dm_ctfp)
#define DT_FUNC_TYPE(dtp)       ((dtp)->dt_type_func)

#define DT_FPTR_CTFP(dtp)       ((dtp)->dt_ddefs->dm_ctfp)
#define DT_FPTR_TYPE(dtp)       ((dtp)->dt_type_fptr)

#define DT_STR_CTFP(dtp)        ((dtp)->dt_ddefs->dm_ctfp)
#define DT_STR_TYPE(dtp)        ((dtp)->dt_type_str)

#define DT_DYN_CTFP(dtp)        ((dtp)->dt_ddefs->dm_ctfp)
#define DT_DYN_TYPE(dtp)        ((dtp)->dt_type_dyn)

#define DT_STACK_CTFP(dtp)      ((dtp)->dt_ddefs->dm_ctfp)
#define DT_STACK_TYPE(dtp)      ((dtp)->dt_type_stack)

#define DT_SYMADDR_CTFP(dtp)    ((dtp)->dt_ddefs->dm_ctfp)
#define DT_SYMADDR_TYPE(dtp)    ((dtp)->dt_type_symaddr)

#define DT_USYMADDR_CTFP(dtp)   ((dtp)->dt_ddefs->dm_ctfp)
#define DT_USYMADDR_TYPE(dtp)   ((dtp)->dt_type_usymaddr)

/*
 * Actions and subroutines are both DT_NODE_FUNC nodes; to avoid confusing
 * an action for a subroutine (or vice versa), we assure that the DT_ACT_*
 * constants and the DIF_SUBR_* constants occupy non-overlapping ranges by
 * starting the DT_ACT_* constants at DIF_SUBR_MAX + 1.
 */
#define DT_ACT_BASE             DIF_SUBR_MAX + 1
#define DT_ACT(n)               (DT_ACT_BASE + (n))

#define DT_ACT_PRINTF           DT_ACT(0)       /* printf() action */
#define DT_ACT_TRACE            DT_ACT(1)       /* trace() action */
#define DT_ACT_TRACEMEM         DT_ACT(2)       /* tracemem() action */
#define DT_ACT_STACK            DT_ACT(3)       /* stack() action */
#define DT_ACT_STOP             DT_ACT(4)       /* stop() action */
#define DT_ACT_BREAKPOINT       DT_ACT(5)       /* breakpoint() action */
#define DT_ACT_PANIC            DT_ACT(6)       /* panic() action */
#define DT_ACT_SPECULATE        DT_ACT(7)       /* speculate() action */
#define DT_ACT_COMMIT           DT_ACT(8)       /* commit() action */
#define DT_ACT_DISCARD          DT_ACT(9)       /* discard() action */
#define DT_ACT_CHILL            DT_ACT(10)      /* chill() action */
#define DT_ACT_EXIT             DT_ACT(11)      /* exit() action */
#define DT_ACT_USTACK           DT_ACT(12)      /* ustack() action */
#define DT_ACT_PRINTA           DT_ACT(13)      /* printa() action */
#define DT_ACT_RAISE            DT_ACT(14)      /* raise() action */
#define DT_ACT_CLEAR            DT_ACT(15)      /* clear() action */
#define DT_ACT_NORMALIZE        DT_ACT(16)      /* normalize() action */
#define DT_ACT_DENORMALIZE      DT_ACT(17)      /* denormalize() action */
#define DT_ACT_TRUNC            DT_ACT(18)      /* trunc() action */
```



```
#define DT_ACT_SYSTEM           DT_ACT(19)      /* system() action */
#define DT_ACT_JSTACK           DT_ACT(20)      /* jstack() action */
#define DT_ACT_FTRUNCATE        DT_ACT(21)      /* ftruncate() action */
#define DT_ACT_FREOPEN          DT_ACT(22)      /* freopen() action */
#define DT_ACT_SYM              DT_ACT(23)      /* sym()/func() actions */
#define DT_ACT_MOD              DT_ACT(24)      /* mod() action */
#define DT_ACT_USYM             DT_ACT(25)      /* usym()/ufunc() actions */
#define DT_ACT_UMOD             DT_ACT(26)      /* umod() action */
#define DT_ACT_UADDR            DT_ACT(27)      /* uaddr() action */
#define DT_ACT_SETOPT           DT_ACT(28)      /* setopt() action */
#define DT_ACT_PRINT            DT_ACT(29)      /* print() action */

/*
 * Sentinel to tell freopen() to restore the saved stdout.  This must not
 * be ever valid for opening for write access via freopen(3C), which of
 * course, "." never is.
 */
#define DT_FREOPEN_RESTORE      "."

#define EDT_BASE        1000    /* base value for libdtrace errnos */

enum {
        EDT_VERSION = EDT_BASE, /* client is requesting unsupported version */
        EDT_VERSINVAL,          /* version string is invalid or overflows */
        EDT_VERSUNDEF,          /* requested API version is not defined */
        EDT_VERSREDUCED,        /* requested API version has been reduced */
        EDT_CTF,                /* libctf called failed (dt_ctferr has more) */
        EDT_COMPILER,           /* error in D program compilation */
        EDT_NOTUPREG,           /* tuple register allocation failure */
        EDT_NOMEM,              /* memory allocation failure */
        EDT_INT2BIG,            /* integer limit exceeded */
        EDT_STR2BIG,            /* string limit exceeded */
        EDT_NOMOD,              /* unknown module name */
        EDT_NOPROV,             /* unknown provider name */
        EDT_NOPROBE,            /* unknown probe name */
        EDT_NOSYM,              /* unknown symbol name */
        EDT_NOSYMADDR,          /* no symbol corresponds to address */
        EDT_NOTYPE,             /* unknown type name */
        EDT_NOVAR,              /* unknown variable name */
        EDT_NOAGG,              /* unknown aggregation name */
        EDT_BADSCOPE,           /* improper use of type name scoping operator */
        EDT_BADSPEC,            /* overspecified probe description */
        EDT_BADSPCV,            /* bad macro variable in probe description */
        EDT_BADID,              /* invalid probe identifier */
        EDT_NOTLOADED,          /* module is not currently loaded */
        EDT_NOCTF,              /* module does not contain any CTF data */
        EDT_DATAMODEL,          /* module and program data models don't match */
        EDT_DIFVERS,            /* library has newer DIF version than driver */
        EDT_BADAGG,             /* unrecognized aggregating action */
        EDT_FIO,                /* file i/o error */
        EDT_DIFINVAL,           /* invalid DIF program */
        EDT_DIFSIZE,            /* invalid DIF size */
        EDT_DIFFAULT,           /* failed to copyin DIF program */
        EDT_BADPROBE,           /* bad probe description */
        EDT_BADPGLOB,           /* bad probe description globbing pattern */
        EDT_NOSCOPE,            /* declaration scope stack underflow */
        EDT_NODECL,             /* declaration stack underflow */
        EDT_DMISMATCH,          /* record list does not match statement */
        EDT_DOFFSET,            /* record data offset error */
        EDT_DALIGN,             /* record data alignment error */
        EDT_BADOPTNAME,         /* invalid dtrace_setopt option name */
        EDT_BADOPTVAL,          /* invalid dtrace_setopt option value */
        EDT_BADOPTCTX,          /* invalid dtrace_setopt option context */
        EDT_CPPFORK,            /* failed to fork preprocessor */
        EDT_CPPEXEC,            /* failed to exec preprocessor */
        EDT_CPPENT,             /* preprocessor not found */
```

```
491         EDT_CPPERR,             /* unknown preprocessor error */
492         EDT_SYMOFLOW,           /* external symbol table overflow */
493         EDT_ACTIVE,             /* operation illegal when tracing is active */
494         EDT_DESTRUCTIVE,        /* destructive actions not allowed */
495         EDT_NOANON,             /* no anonymous tracing state */
496         EDT_ISANON,             /* can't claim anon state and enable probes */
497         EDT_ENDTOOBIG,          /* END enablings exceed size of prncpl buffer */
498         EDT_NOCONV,             /* failed to load type for printf conversion */
499         EDT_BADCONV,            /* incomplete printf conversion */
500         EDT_BADERROR,           /* invalid library ERROR action */
501         EDT_ERRABORT,           /* abort due to error */
502         EDT_DROPABORT,          /* abort due to drop */
503         EDT_DIRABORT,           /* abort explicitly directed */
504         EDT_BADRVAL,            /* invalid return value from callback */
505         EDT_BADNORMAL,          /* invalid normalization */
506         EDT_BUFTOOSMALL,        /* enabling exceeds size of buffer */
507         EDT_BADTRUNC,           /* invalid truncation */
508         EDT_BUSY,               /* device busy (active kernel debugger) */
509         EDT_ACCESS,             /* insufficient privileges to use DTrace */
510         EDT_NOENT,              /* dtrace device not available */
511         EDT_BRICKED,            /* abort due to systemic unresponsiveness */
512         EDT_HARDWIRE,           /* failed to load hard-wired definitions */
513         EDT_ELFVERSION,         /* libelf is out-of-date w.r.t libdtrace */
514         EDT_NOBUFFERED,         /* attempt to buffer output without handler */
515         EDT_UNSTABLE,           /* description matched unstable set of probes */
516         EDT_BADSETOPT,          /* invalid setopt library action */
517         EDT_BADSTACKPC,         /* invalid stack program counter size */
518         EDT_BADAGGVAR,          /* invalid aggregation variable identifier */
519         EDT_OVERSION,           /* client is requesting deprecated version */
520         EDT_ENABLING_ERR,       /* failed to enable probe */
521         EDT_NOPROBES,           /* no probes sites for declared provider */
522         EDT_CANTLOAD            /* failed to load a module */
136         EDT_NOPROBES            /* no probes sites for declared provider */
523 };
______unchanged_portion_omitted_
```

```
**********************************************************
   23175 Tue Jan 14 16:48:55 2014
new/usr/src/lib/libdtrace/common/dt_lex.l
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 */
#endif /* ! codereview */

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <ctype.h>
#include <errno.h>

#include <dt_impl.h>
#include <dt_grammar.h>
#include <dt_parser.h>
#include <dt_string.h>

/*
 * We need to undefine lex's input and unput macros so that references to these
 * call the functions provided at the end of this source file.
 */
#undef input
#undef unput

static int id_or_type(const char *);
static int input(void);
static void unput(int);

/*
 * We first define a set of labeled states for use in the D lexer and then a
```



```
 * set of regular expressions to simplify things below. The lexer states are:
 *
 * S0 - D program clause and expression lexing
 * S1 - D comments (i.e. skip everything until end of comment)
 * S2 - D program outer scope (probe specifiers and declarations)
 * S3 - D control line parsing (i.e. after ^# is seen but before \n)
 * S4 - D control line scan (locate control directives only and invoke S3)
 */
%}

%e 1500         /* maximum nodes */
%p 4900         /* maximum positions */
%p 3700         /* maximum positions */
%n 600          /* maximum states */
%a 3000         /* maximum transitions */
#endif /* ! codereview */

%s S0 S1 S2 S3 S4

RGX_AGG         "@"[a-zA-Z_][0-9a-zA-Z_]*
RGX_PSPEC       [-$:a-zA-Z_.?*\\\[\]!][-$:0-9a-zA-Z_.'?*\\\[\]!]*
RGX_ALTIDENT    [a-zA-Z_][0-9a-zA-Z_]*
RGX_LMID        LM[0-9a-fA-F]+'
RGX_MOD_IDENT   [a-zA-Z_'][0-9a-z.A-Z_']*'
#endif /* ! codereview */
RGX_IDENT       [a-zA-Z_'][0-9a-zA-Z_']*
RGX_INT         ([0-9]+|0[xX][0-9A-Fa-f]+)[uU]?[lL]?[lL]?
RGX_FP          ([0-9]+("."?)[0-9]*|"."[0-9]+)((e|E)("+"|-)?[0-9]+)?[fFlL]?
RGX_WS          [\f\n\r\t\v ]
RGX_STR         ([^"\\\n]|\\[^"\n]|\\\")*
RGX_CHR         ([^'\\\n]|\\[^'\n]|\\')*
RGX_INTERP      ^[\f\t\v ]*#!.*
RGX_CTL         ^[\f\t\v ]*#

%%

%{

/*
 * We insert a special prologue into yylex() itself: if the pcb contains a
 * context token, we return that prior to running the normal lexer.  This
 * allows libdtrace to force yacc into one of our three parsing contexts: D
 * expression (DT_CTX_DEXPR), D program (DT_CTX_DPROG) or D type (DT_CTX_DTYPE).
 * Once the token is returned, we clear it so this only happens once.
 */
if (yypcb->pcb_token != 0) {
        int tok = yypcb->pcb_token;
        yypcb->pcb_token = 0;
        return (tok);
}

%}

<S0>auto        return (DT_KEY_AUTO);
<S0>break       return (DT_KEY_BREAK);
<S0>case        return (DT_KEY_CASE);
<S0>char        return (DT_KEY_CHAR);
<S0>const       return (DT_KEY_CONST);
<S0>continue    return (DT_KEY_CONTINUE);
<S0>counter     return (DT_KEY_COUNTER);
<S0>default     return (DT_KEY_DEFAULT);
<S0>do          return (DT_KEY_DO);
<S0>double      return (DT_KEY_DOUBLE);
<S0>else        return (DT_KEY_ELSE);
<S0>enum        return (DT_KEY_ENUM);
<S0>extern      return (DT_KEY_EXTERN);
```

```
<S0>float       return (DT_KEY_FLOAT);
<S0>for         return (DT_KEY_FOR);
<S0>goto        return (DT_KEY_GOTO);
<S0>if          return (DT_KEY_IF);
<S0>import      return (DT_KEY_IMPORT);
<S0>inline      return (DT_KEY_INLINE);
<S0>int         return (DT_KEY_INT);
<S0>long        return (DT_KEY_LONG);
<S0>offsetof    return (DT_TOK_OFFSETOF);
<S0>probe       return (DT_KEY_PROBE);
<S0>provider    return (DT_KEY_PROVIDER);
<S0>register    return (DT_KEY_REGISTER);
<S0>restrict    return (DT_KEY_RESTRICT);
<S0>return      return (DT_KEY_RETURN);
<S0>self        return (DT_KEY_SELF);
<S0>short       return (DT_KEY_SHORT);
<S0>signed      return (DT_KEY_SIGNED);
<S0>sizeof      return (DT_TOK_SIZEOF);
<S0>static      return (DT_KEY_STATIC);
<S0>string      return (DT_KEY_STRING);
<S0>stringof    return (DT_TOK_STRINGOF);
<S0>struct      return (DT_KEY_STRUCT);
<S0>switch      return (DT_KEY_SWITCH);
<S0>this        return (DT_KEY_THIS);
<S0>translator  return (DT_KEY_XLATOR);
<S0>typedef     return (DT_KEY_TYPEDEF);
<S0>union       return (DT_KEY_UNION);
<S0>unsigned    return (DT_KEY_UNSIGNED);
<S0>userland    return (DT_KEY_USERLAND);
#endif /* ! codereview */
<S0>void        return (DT_KEY_VOID);
<S0>volatile    return (DT_KEY_VOLATILE);
<S0>while       return (DT_KEY_WHILE);
<S0>xlate       return (DT_TOK_XLATE);

<S2>auto        { yybegin(YYS_EXPR);    return (DT_KEY_AUTO); }
<S2>char        { yybegin(YYS_EXPR);    return (DT_KEY_CHAR); }
<S2>const       { yybegin(YYS_EXPR);    return (DT_KEY_CONST); }
<S2>counter     { yybegin(YYS_DEFINE);  return (DT_KEY_COUNTER); }
<S2>double      { yybegin(YYS_EXPR);    return (DT_KEY_DOUBLE); }
<S2>enum        { yybegin(YYS_EXPR);    return (DT_KEY_ENUM); }
<S2>extern      { yybegin(YYS_EXPR);    return (DT_KEY_EXTERN); }
<S2>float       { yybegin(YYS_EXPR);    return (DT_KEY_FLOAT); }
<S2>import      { yybegin(YYS_EXPR);    return (DT_KEY_IMPORT); }
<S2>inline      { yybegin(YYS_DEFINE);  return (DT_KEY_INLINE); }
<S2>int         { yybegin(YYS_EXPR);    return (DT_KEY_INT); }
<S2>long        { yybegin(YYS_EXPR);    return (DT_KEY_LONG); }
<S2>provider    { yybegin(YYS_DEFINE);  return (DT_KEY_PROVIDER); }
<S2>register    { yybegin(YYS_EXPR);    return (DT_KEY_REGISTER); }
<S2>restrict    { yybegin(YYS_EXPR);    return (DT_KEY_RESTRICT); }
<S2>self        { yybegin(YYS_EXPR);    return (DT_KEY_SELF); }
<S2>short       { yybegin(YYS_EXPR);    return (DT_KEY_SHORT); }
<S2>signed      { yybegin(YYS_EXPR);    return (DT_KEY_SIGNED); }
<S2>static      { yybegin(YYS_EXPR);    return (DT_KEY_STATIC); }
<S2>string      { yybegin(YYS_EXPR);    return (DT_KEY_STRING); }
<S2>struct      { yybegin(YYS_EXPR);    return (DT_KEY_STRUCT); }
<S2>this        { yybegin(YYS_EXPR);    return (DT_KEY_THIS); }
<S2>translator  { yybegin(YYS_DEFINE);  return (DT_KEY_XLATOR); }
<S2>typedef     { yybegin(YYS_EXPR);    return (DT_KEY_TYPEDEF); }
<S2>union       { yybegin(YYS_EXPR);    return (DT_KEY_UNION); }
<S2>unsigned    { yybegin(YYS_EXPR);    return (DT_KEY_UNSIGNED); }
<S2>void        { yybegin(YYS_EXPR);    return (DT_KEY_VOID); }
<S2>volatile    { yybegin(YYS_EXPR);    return (DT_KEY_VOLATILE); }

<S0>"$$"[0-9]+  {
                        int i = atoi(yytext + 2);
```



```
                        char *v = "";

                        /*
                         * A macro argument reference substitutes the text of
                         * an argument in place of the current token.  When we
                         * see $$<d> we fetch the saved string from pcb_sargv
                         * (or use the default argument if the option has been
                         * set and the argument hasn't been specified) and
                         * return a token corresponding to this string.
                         */
                        if (i < 0 || (i >= yypcb->pcb_sargc &&
                            !(yypcb->pcb_cflags & DTRACE_C_DEFARG))) {
                                xyerror(D_MACRO_UNDEF, "macro argument %s is "
                                    "not defined\n", yytext);
                        }

                        if (i < yypcb->pcb_sargc) {
                                v = yypcb->pcb_sargv[i]; /* get val from pcb */
                                yypcb->pcb_sflagv[i] |= DT_IDFLG_REF;
                        }

                        if ((yylval.l_str = strdup(v)) == NULL)
                                longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

                        (void) stresc2chr(yylval.l_str);
                        return (DT_TOK_STRING);
                }

<S0>"$"[0-9]+   {
                        int i = atoi(yytext + 1);
                        char *p, *v = "0";

                        /*
                         * A macro argument reference substitutes the text of
                         * one identifier or integer pattern for another.  When
                         * we see $<d> we fetch the saved string from pcb_sargv
                         * (or use the default argument if the option has been
                         * set and the argument hasn't been specified) and
                         * return a token corresponding to this string.
                         */
                        if (i < 0 || (i >= yypcb->pcb_sargc &&
                            !(yypcb->pcb_cflags & DTRACE_C_DEFARG))) {
                                xyerror(D_MACRO_UNDEF, "macro argument %s is "
                                    "not defined\n", yytext);
                        }

                        if (i < yypcb->pcb_sargc) {
                                v = yypcb->pcb_sargv[i]; /* get val from pcb */
                                yypcb->pcb_sflagv[i] |= DT_IDFLG_REF;
                        }

                        /*
                         * If the macro text is not a valid integer or ident,
                         * then we treat it as a string.  The string may be
                         * optionally enclosed in quotes, which we strip.
                         */
                        if (strbadidnum(v)) {
                                size_t len = strlen(v);

                                if (len != 1 && *v == '"' && v[len - 1] == '"')
                                        yylval.l_str = strndup(v + 1, len - 2);
                                else
                                        yylval.l_str = strndup(v, len);

                                if (yylval.l_str == NULL)
                                        longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
```

```
255                             (void) stresc2chr(yylval.l_str);
256                             return (DT_TOK_STRING);
257                     }
259                     /*
260                      * If the macro text is not a string an begins with a
261                      * digit or a +/- sign, process it as an integer token.
262                      */
263                     if (isdigit(v[0]) || v[0] == '-' || v[0] == '+') {
264                             if (isdigit(v[0]))
265                                     yyintprefix = 0;
266                             else
267                                     yyintprefix = *v++;
269                             errno = 0;
270                             yylval.l_int = strtoull(v, &p, 0);
271                             (void) strncpy(yyintsuffix, p,
272                                 sizeof (yyintsuffix));
273                             yyintdecimal = *v != '0';
275                             if (errno == ERANGE) {
276                                     xyerror(D_MACRO_OFLOW, "macro argument"
277                                         " %s constant %s results in integer"
278                                         " overflow\n", yytext, v);
279                             }
281                             return (DT_TOK_INT);
282                     }
284                     return (id_or_type(v));
285             }
287 <S0>"$$"{RGX_IDENT} {
288                     dt_ident_t *idp = dt_idhash_lookup(
289                         yypcb->pcb_hdl->dt_macros, yytext + 2);
291                     char s[16]; /* enough for UINT_MAX + \0 */
293                     if (idp == NULL) {
294                             xyerror(D_MACRO_UNDEF, "macro variable %s "
295                                 "is not defined\n", yytext);
296                     }
298                     /*
299                      * For the moment, all current macro variables are of
300                      * type id_t (refer to dtrace_update() for details).
301                      */
302                     (void) snprintf(s, sizeof (s), "%u", idp->di_id);
303                     if ((yylval.l_str = strdup(s)) == NULL)
304                             longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
306                     return (DT_TOK_STRING);
307             }
309 <S0>"$"{RGX_IDENT} {
310                     dt_ident_t *idp = dt_idhash_lookup(
311                         yypcb->pcb_hdl->dt_macros, yytext + 1);
313                     if (idp == NULL) {
314                             xyerror(D_MACRO_UNDEF, "macro variable %s "
315                                 "is not defined\n", yytext);
316                     }
318                     /*
319                      * For the moment, all current macro variables are of
```



```
320                      * type id_t (refer to dtrace_update() for details).
321                      */
322                     yylval.l_int = (intmax_t)(int)idp->di_id;
323                     yyintprefix = 0;
324                     yyintsuffix[0] = '\0';
325                     yyintdecimal = 1;
327                     return (DT_TOK_INT);
328             }
330 <S0>{RGX_IDENT} |
331 <S0>{RGX_MOD_IDENT}{RGX_IDENT} |
332 <S0>{RGX_MOD_IDENT} {
 28 <S0>{RGX_IDENT} {
333                     return (id_or_type(yytext));
334             }
336 <S0>{RGX_AGG}   {
337                     if ((yylval.l_str = strdup(yytext)) == NULL)
338                             longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
339                     return (DT_TOK_AGG);
340             }
342 <S0>"@"         {
343                     if ((yylval.l_str = strdup("@_")) == NULL)
344                             longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
345                     return (DT_TOK_AGG);
346             }
348 <S0>{RGX_INT}   |
349 <S2>{RGX_INT}   |
350 <S3>{RGX_INT}   {
351                     char *p;
353                     errno = 0;
354                     yylval.l_int = strtoull(yytext, &p, 0);
355                     yyintprefix = 0;
356                     (void) strncpy(yyintsuffix, p, sizeof (yyintsuffix));
357                     yyintdecimal = yytext[0] != '0';
359                     if (errno == ERANGE) {
360                             xyerror(D_INT_OFLOW, "constant %s results in "
361                                 "integer overflow\n", yytext);
362                     }
364                     if (*p != '\0' && strchr("uUlL", *p) == NULL) {
365                             xyerror(D_INT_DIGIT, "constant %s contains "
366                                 "invalid digit %c\n", yytext, *p);
367                     }
369                     if ((YYSTATE) != S3)
370                             return (DT_TOK_INT);
372                     yypragma = dt_node_link(yypragma,
373                         dt_node_int(yylval.l_int));
374             }
376 <S0>{RGX_FP}    yyerror("floating-point constants are not permitted\n");
378 <S0>\"{RGX_STR}$ |
379 <S3>\"{RGX_STR}$ xyerror(D_STR_NL, "newline encountered in string literal");
381 <S0>\"{RGX_STR}\" |
382 <S3>\"{RGX_STR}\" {
383                     /*
384                      * Quoted string -- convert C escape sequences and
```

```
385                          * return the string as a token.
386                          */
387                         yylval.l_str = strndup(yytext + 1, yyleng - 2);
388
389                         if (yylval.l_str == NULL)
390                                 longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
391
392                         (void) stresc2chr(yylval.l_str);
393                         if ((YYSTATE) != S3)
394                                 return (DT_TOK_STRING);
395
396                         yypragma = dt_node_link(yypragma,
397                             dt_node_string(yylval.l_str));
398                 }
399
400 <S0>'{RGX_CHR}$ xyerror(D_CHR_NL, "newline encountered in character constant");
401
402 <S0>'{RGX_CHR}' {
403                         char *s, *p, *q;
404                         size_t nbytes;
405
406                         /*
407                          * Character constant -- convert C escape sequences and
408                          * return the character as an integer immediate value.
409                          */
410                         if (yyleng == 2)
411                                 xyerror(D_CHR_NULL, "empty character constant");
412
413                         s = yytext + 1;
414                         yytext[yyleng - 1] = '\0';
415                         nbytes = stresc2chr(s);
416                         yylval.l_int = 0;
417                         yyintprefix = 0;
418                         yyintsuffix[0] = '\0';
419                         yyintdecimal = 1;
420
421                         if (nbytes > sizeof (yylval.l_int)) {
422                                 xyerror(D_CHR_OFLOW, "character constant is "
423                                     "too long");
424                         }
425 #ifdef _LITTLE_ENDIAN
426                         p = ((char *)&yylval.l_int) + nbytes - 1;
427                         for (q = s; nbytes != 0; nbytes--)
428                                 *p-- = *q++;
429 #else
430                         bcopy(s, ((char *)&yylval.l_int) +
431                             sizeof (yylval.l_int) - nbytes, nbytes);
432 #endif
433                         return (DT_TOK_INT);
434                 }
435
436 <S0>"/*"        |
437 <S2>"/*"        {
438                         yypcb->pcb_cstate = (YYSTATE);
439                         BEGIN(S1);
440                 }
441
442 <S0>{RGX_INTERP} |
443 <S2>{RGX_INTERP} ;      /* discard any #! lines */
444
445 <S0>{RGX_CTL}   |
446 <S2>{RGX_CTL}   |
447 <S4>{RGX_CTL}   {
448                         assert(yypragma == NULL);
449                         yypcb->pcb_cstate = (YYSTATE);
450                         BEGIN(S3);
```



```
451                 }
452
453 <S4>.           ;       /* discard */
454 <S4>"\n"        ;       /* discard */
455
456 <S0>"/"         {
457                         int c, tok;
458
459                         /*
460                          * The use of "/" as the predicate delimiter and as the
461                          * integer division symbol requires special lookahead
462                          * to avoid a shift/reduce conflict in the D grammar.
463                          * We look ahead to the next non-whitespace character.
464                          * If we encounter EOF, ";", "{", or "/", then this "/"
465                          * closes the predicate and we return DT_TOK_EPRED.
466                          * If we encounter anything else, it's DT_TOK_DIV.
467                          */
468                         while ((c = input()) != 0) {
469                                 if (strchr("\f\n\r\t\v ", c) == NULL)
470                                         break;
471                         }
472
473                         if (c == 0 || c == ';' || c == '{' || c == '/') {
474                                 if (yypcb->pcb_parens != 0) {
475                                         yyerror("closing ) expected in "
476                                             "predicate before /\n");
477                                 }
478                                 if (yypcb->pcb_brackets != 0) {
479                                         yyerror("closing ] expected in "
480                                             "predicate before /\n");
481                                 }
482                                 tok = DT_TOK_EPRED;
483                         } else
484                                 tok = DT_TOK_DIV;
485
486                         unput(c);
487                         return (tok);
488                 }
489
490 <S0>"("         {
491                         yypcb->pcb_parens++;
492                         return (DT_TOK_LPAR);
493                 }
494
495 <S0>")"         {
496                         if (--yypcb->pcb_parens < 0)
497                                 yyerror("extra ) in input stream\n");
498                         return (DT_TOK_RPAR);
499                 }
500
501 <S0>"["         {
502                         yypcb->pcb_brackets++;
503                         return (DT_TOK_LBRAC);
504                 }
505
506 <S0>"]"         {
507                         if (--yypcb->pcb_brackets < 0)
508                                 yyerror("extra ] in input stream\n");
509                         return (DT_TOK_RBRAC);
510                 }
511
512 <S0>"{"         |
513 <S2>"{"         {
514                         yypcb->pcb_braces++;
515                         return ('{');
516                 }
```

```
518 <S0>"}"         {
519                         if (--yypcb->pcb_braces < 0)
520                                 yyerror("extra } in input stream\n");
521                         return ('}');
522                 }

524 <S0>"|"         return (DT_TOK_BOR);
525 <S0>"^"         return (DT_TOK_XOR);
526 <S0>"&"         return (DT_TOK_BAND);
527 <S0>"&&"        return (DT_TOK_LAND);
528 <S0>"^^"        return (DT_TOK_LXOR);
529 <S0>"||"        return (DT_TOK_LOR);
530 <S0>"=="        return (DT_TOK_EQU);
531 <S0>"!="        return (DT_TOK_NEQ);
532 <S0>"<"         return (DT_TOK_LT);
533 <S0>"<="        return (DT_TOK_LE);
534 <S0>">"         return (DT_TOK_GT);
535 <S0>">="        return (DT_TOK_GE);
536 <S0>"<<"        return (DT_TOK_LSH);
537 <S0>">>"        return (DT_TOK_RSH);
538 <S0>"+"         return (DT_TOK_ADD);
539 <S0>"-"         return (DT_TOK_SUB);
540 <S0>"*"         return (DT_TOK_MUL);
541 <S0>"%"         return (DT_TOK_MOD);
542 <S0>"~"         return (DT_TOK_BNEG);
543 <S0>"!"         return (DT_TOK_LNEG);
544 <S0>"?"         return (DT_TOK_QUESTION);
545 <S0>":"         return (DT_TOK_COLON);
546 <S0>"."         return (DT_TOK_DOT);
547 <S0>"->"        return (DT_TOK_PTR);
548 <S0>"="         return (DT_TOK_ASGN);
549 <S0>"+="        return (DT_TOK_ADD_EQ);
550 <S0>"-="        return (DT_TOK_SUB_EQ);
551 <S0>"*="        return (DT_TOK_MUL_EQ);
552 <S0>"/="        return (DT_TOK_DIV_EQ);
553 <S0>"%="        return (DT_TOK_MOD_EQ);
554 <S0>"&="        return (DT_TOK_AND_EQ);
555 <S0>"^="        return (DT_TOK_XOR_EQ);
556 <S0>"|="        return (DT_TOK_OR_EQ);
557 <S0>"<<="       return (DT_TOK_LSH_EQ);
558 <S0>">>="       return (DT_TOK_RSH_EQ);
559 <S0>"++"        return (DT_TOK_ADDADD);
560 <S0>"--"        return (DT_TOK_SUBSUB);
561 <S0>"..."       return (DT_TOK_ELLIPSIS);
562 <S0>","         return (DT_TOK_COMMA);
563 <S0>";"         return (';');
564 <S0>{RGX_WS}    ; /* discard */
565 <S0>"\\"\n      ; /* discard */
566 <S0>.           yyerror("syntax error near \"%c\"\n", yytext[0]);

568 <S1>"/*"        yyerror("/* encountered inside a comment\n");
569 <S1>"*/"        BEGIN(yypcb->pcb_cstate);
570 <S1>.|\n        ; /* discard */

572 <S2>{RGX_PSPEC} {
573                         /*
574                          * S2 has an ambiguity because RGX_PSPEC includes '*'
575                          * as a glob character and '*' also can be DT_TOK_STAR.
576                          * Since lex always matches the longest token, this
577                          * rule can be matched by an input string like "int*",
578                          * which could begin a global variable declaration such
579                          * as "int*x;" or could begin a RGX_PSPEC with globbing
580                          * such as "int* { trace(timestamp); }".  If C_PSPEC is
581                          * not set, we must resolve the ambiguity in favor of
582                          * the type and perform lexer pushback if the fragment
```



```
583                          * before '*' or entire fragment matches a type name.
584                          * If C_PSPEC is set, we always return a PSPEC token.
585                          * If C_PSPEC is off, the user can avoid ambiguity by
586                          * including a ':' delimiter in the specifier, which
587                          * they should be doing anyway to specify the provider.
588                          */
589                         if (!(yypcb->pcb_cflags & DTRACE_C_PSPEC) &&
590                             strchr(yytext, ':') == NULL) {

592                                 char *p = strchr(yytext, '*');
593                                 char *q = yytext + yyleng - 1;

595                                 if (p != NULL && p > yytext)
596                                         *p = '\0'; /* prune yytext */

598                                 if (dt_type_lookup(yytext, NULL) == 0) {
599                                         yylval.l_str = strdup(yytext);

601                                         if (yylval.l_str == NULL) {
602                                                 longjmp(yypcb->pcb_jmpbuf,
603                                                     EDT_NOMEM);
604                                         }

606                                         if (p != NULL && p > yytext) {
607                                                 for (*p = '*'; q >= p; q--)
608                                                         unput(*q);
609                                         }

611                                         yybegin(YYS_EXPR);
612                                         return (DT_TOK_TNAME);
613                                 }

615                                 if (p != NULL && p > yytext)
616                                         *p = '*'; /* restore yytext */
617                         }

619                         if ((yylval.l_str = strdup(yytext)) == NULL)
620                                 longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

622                         return (DT_TOK_PSPEC);
623                 }

625 <S2>"/"         return (DT_TOK_DIV);
626 <S2>","         return (DT_TOK_COMMA);

628 <S2>{RGX_WS}    ; /* discard */
629 <S2>.           yyerror("syntax error near \"%c\"\n", yytext[0]);

631 <S3>\n          {
632                         dt_pragma(yypragma);
633                         yypragma = NULL;
634                         BEGIN(yypcb->pcb_cstate);
635                 }

637 <S3>[\f\t\v ]+  ; /* discard */

639 <S3>[^\f\n\t\v "]+ {
640                         dt_node_t *dnp;

642                         if ((yylval.l_str = strdup(yytext)) == NULL)
643                                 longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

645                         /*
646                          * We want to call dt_node_ident() here, but we can't
647                          * because it will expand inlined identifiers, which we
648                          * don't want to do from #pragma context in order to
```

```
649                          * support pragmas that apply to the ident itself.  We
650                          * call dt_node_string() and then reset dn_op instead.
651                          */
652                         dnp = dt_node_string(yylval.l_str);
653                         dnp->dn_kind = DT_NODE_IDENT;
654                         dnp->dn_op = DT_TOK_IDENT;
655                         yypragma = dt_node_link(yypragma, dnp);
656                 }

658 <S3>.           yyerror("syntax error near \"%c\"\n", yytext[0]);

660 %%

662 /*
663  * yybegin provides a wrapper for use from C code around the lex BEGIN() macro.
664  * We use two main states for lexing because probe descriptions use a syntax
665  * that is incompatible with the normal D tokens (e.g. names can contain "-").
666  * yybegin also handles the job of switching between two lists of dt_nodes
667  * as we allocate persistent definitions, like inlines, and transient nodes
668  * that will be freed once we are done parsing the current program file.
669  */
670 void
671 yybegin(yystate_t state)
672 {
673 #ifdef  YYDEBUG
674         yydebug = _dtrace_debug;
675 #endif
676         if (yypcb->pcb_yystate == state)
677                 return; /* nothing to do if we're in the state already */

679         if (yypcb->pcb_yystate == YYS_DEFINE) {
680                 yypcb->pcb_list = yypcb->pcb_hold;
681                 yypcb->pcb_hold = NULL;
682         }

684         switch (state) {
685         case YYS_CLAUSE:
686                 BEGIN(S2);
687                 break;
688         case YYS_DEFINE:
689                 assert(yypcb->pcb_hold == NULL);
690                 yypcb->pcb_hold = yypcb->pcb_list;
691                 yypcb->pcb_list = NULL;
692                 /*FALLTHRU*/
693         case YYS_EXPR:
694                 BEGIN(S0);
695                 break;
696         case YYS_DONE:
697                 break;
698         case YYS_CONTROL:
699                 BEGIN(S4);
700                 break;
701         default:
702                 xyerror(D_UNKNOWN, "internal error -- bad yystate %d\n", state);
703         }

705         yypcb->pcb_yystate = state;
706 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   41593 Tue Jan 14 16:48:55 2014
new/usr/src/lib/libdtrace/common/dt_module.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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, Joyent, Inc.  All rights reserved.
 */
#endif /* ! codereview */

#include <sys/types.h>
#include <sys/modctl.h>
#include <sys/kobj.h>
#include <sys/kobj_impl.h>
#include <sys/sysmacros.h>
#include <sys/elf.h>
#include <sys/task.h>

#include <unistd.h>
#include <project.h>
#include <strings.h>
#include <stdlib.h>
#include <libelf.h>
#include <limits.h>
#include <assert.h>
#include <errno.h>
#include <dirent.h>

#include <dt_strtab.h>
#include <dt_module.h>
#include <dt_impl.h>

static const char *dt_module_strtab; /* active strtab for qsort callbacks */

static void
dt_module_symhash_insert(dt_module_t *dmp, const char *name, uint_t id)
{
```



```
        dt_sym_t *dsp = &dmp->dm_symchains[dmp->dm_symfree];
        uint_t h;

        assert(dmp->dm_symfree < dmp->dm_nsymelems + 1);

        dsp->ds_symid = id;
        h = dt_strtab_hash(name, NULL) % dmp->dm_nsymbuckets;
        dsp->ds_next = dmp->dm_symbuckets[h];
        dmp->dm_symbuckets[h] = dmp->dm_symfree++;
}

static uint_t
dt_module_syminit32(dt_module_t *dmp)
{
#if STT_NUM != (STT_TLS + 1)
#error "STT_NUM has grown. update dt_module_syminit32()"
#endif

        const Elf32_Sym *sym = dmp->dm_symtab.cts_data;
        const char *base = dmp->dm_strtab.cts_data;
        size_t ss_size = dmp->dm_strtab.cts_size;
        uint_t i, n = dmp->dm_nsymelems;
        uint_t asrsv = 0;

        for (i = 0; i < n; i++, sym++) {
                const char *name = base + sym->st_name;
                uchar_t type = ELF32_ST_TYPE(sym->st_info);

                if (type >= STT_NUM || type == STT_SECTION)
                        continue; /* skip sections and unknown types */

                if (sym->st_name == 0 || sym->st_name >= ss_size)
                        continue; /* skip null or invalid names */

                if (sym->st_value != 0 &&
                    (ELF32_ST_BIND(sym->st_info) != STB_LOCAL || sym->st_size))
                        asrsv++; /* reserve space in the address map */

                dt_module_symhash_insert(dmp, name, i);
        }

        return (asrsv);
}

static uint_t
dt_module_syminit64(dt_module_t *dmp)
{
#if STT_NUM != (STT_TLS + 1)
#error "STT_NUM has grown. update dt_module_syminit64()"
#endif

        const Elf64_Sym *sym = dmp->dm_symtab.cts_data;
        const char *base = dmp->dm_strtab.cts_data;
        size_t ss_size = dmp->dm_strtab.cts_size;
        uint_t i, n = dmp->dm_nsymelems;
        uint_t asrsv = 0;

        for (i = 0; i < n; i++, sym++) {
                const char *name = base + sym->st_name;
                uchar_t type = ELF64_ST_TYPE(sym->st_info);

                if (type >= STT_NUM || type == STT_SECTION)
                        continue; /* skip sections and unknown types */

                if (sym->st_name == 0 || sym->st_name >= ss_size)
                        continue; /* skip null or invalid names */
```

```
124                 if (sym->st_value != 0 &&
125                     (ELF64_ST_BIND(sym->st_info) != STB_LOCAL || sym->st_size))
126                         asrsv++; /* reserve space in the address map */
127
128                 dt_module_symhash_insert(dmp, name, i);
129         }
130
131         return (asrsv);
132 }
133
134 /*
135  * Sort comparison function for 32-bit symbol address-to-name lookups.  We sort
136  * symbols by value.  If values are equal, we prefer the symbol that is
137  * non-zero sized, typed, not weak, or lexically first, in that order.
138  */
139 static int
140 dt_module_symcomp32(const void *lp, const void *rp)
141 {
142         Elf32_Sym *lhs = *((Elf32_Sym **)lp);
143         Elf32_Sym *rhs = *((Elf32_Sym **)rp);
144
145         if (lhs->st_value != rhs->st_value)
146                 return (lhs->st_value > rhs->st_value ? 1 : -1);
147
148         if ((lhs->st_size == 0) != (rhs->st_size == 0))
149                 return (lhs->st_size == 0 ? 1 : -1);
150
151         if ((ELF32_ST_TYPE(lhs->st_info) == STT_NOTYPE) !=
152             (ELF32_ST_TYPE(rhs->st_info) == STT_NOTYPE))
153                 return (ELF32_ST_TYPE(lhs->st_info) == STT_NOTYPE ? 1 : -1);
154
155         if ((ELF32_ST_BIND(lhs->st_info) == STB_WEAK) !=
156             (ELF32_ST_BIND(rhs->st_info) == STB_WEAK))
157                 return (ELF32_ST_BIND(lhs->st_info) == STB_WEAK ? 1 : -1);
158
159         return (strcmp(dt_module_strtab + lhs->st_name,
160             dt_module_strtab + rhs->st_name));
161 }
162
163 /*
164  * Sort comparison function for 64-bit symbol address-to-name lookups.  We sort
165  * symbols by value.  If values are equal, we prefer the symbol that is
166  * non-zero sized, typed, not weak, or lexically first, in that order.
167  */
168 static int
169 dt_module_symcomp64(const void *lp, const void *rp)
170 {
171         Elf64_Sym *lhs = *((Elf64_Sym **)lp);
172         Elf64_Sym *rhs = *((Elf64_Sym **)rp);
173
174         if (lhs->st_value != rhs->st_value)
175                 return (lhs->st_value > rhs->st_value ? 1 : -1);
176
177         if ((lhs->st_size == 0) != (rhs->st_size == 0))
178                 return (lhs->st_size == 0 ? 1 : -1);
179
180         if ((ELF64_ST_TYPE(lhs->st_info) == STT_NOTYPE) !=
181             (ELF64_ST_TYPE(rhs->st_info) == STT_NOTYPE))
182                 return (ELF64_ST_TYPE(lhs->st_info) == STT_NOTYPE ? 1 : -1);
183
184         if ((ELF64_ST_BIND(lhs->st_info) == STB_WEAK) !=
185             (ELF64_ST_BIND(rhs->st_info) == STB_WEAK))
186                 return (ELF64_ST_BIND(lhs->st_info) == STB_WEAK ? 1 : -1);
187
188         return (strcmp(dt_module_strtab + lhs->st_name,
```



```
189             dt_module_strtab + rhs->st_name));
190 }
191
192 static void
193 dt_module_symsort32(dt_module_t *dmp)
194 {
195         Elf32_Sym *symtab = (Elf32_Sym *)dmp->dm_symtab.cts_data;
196         Elf32_Sym **sympp = (Elf32_Sym **)dmp->dm_asmap;
197         const dt_sym_t *dsp = dmp->dm_symchains + 1;
198         uint_t i, n = dmp->dm_symfree;
199
200         for (i = 1; i < n; i++, dsp++) {
201                 Elf32_Sym *sym = symtab + dsp->ds_symid;
202                 if (sym->st_value != 0 &&
203                     (ELF32_ST_BIND(sym->st_info) != STB_LOCAL || sym->st_size))
204                         *sympp++ = sym;
205         }
206
207         dmp->dm_aslen = (uint_t)(sympp - (Elf32_Sym **)dmp->dm_asmap);
208         assert(dmp->dm_aslen <= dmp->dm_asrsv);
209
210         dt_module_strtab = dmp->dm_strtab.cts_data;
211         qsort(dmp->dm_asmap, dmp->dm_aslen,
212             sizeof (Elf32_Sym *), dt_module_symcomp32);
213         dt_module_strtab = NULL;
214 }
215
216 static void
217 dt_module_symsort64(dt_module_t *dmp)
218 {
219         Elf64_Sym *symtab = (Elf64_Sym *)dmp->dm_symtab.cts_data;
220         Elf64_Sym **sympp = (Elf64_Sym **)dmp->dm_asmap;
221         const dt_sym_t *dsp = dmp->dm_symchains + 1;
222         uint_t i, n = dmp->dm_symfree;
223
224         for (i = 1; i < n; i++, dsp++) {
225                 Elf64_Sym *sym = symtab + dsp->ds_symid;
226                 if (sym->st_value != 0 &&
227                     (ELF64_ST_BIND(sym->st_info) != STB_LOCAL || sym->st_size))
228                         *sympp++ = sym;
229         }
230
231         dmp->dm_aslen = (uint_t)(sympp - (Elf64_Sym **)dmp->dm_asmap);
232         assert(dmp->dm_aslen <= dmp->dm_asrsv);
233
234         dt_module_strtab = dmp->dm_strtab.cts_data;
235         qsort(dmp->dm_asmap, dmp->dm_aslen,
236             sizeof (Elf64_Sym *), dt_module_symcomp64);
237         dt_module_strtab = NULL;
238 }
239
240 static GElf_Sym *
241 dt_module_symgelf32(const Elf32_Sym *src, GElf_Sym *dst)
242 {
243         if (dst != NULL) {
244                 dst->st_name = src->st_name;
245                 dst->st_info = src->st_info;
246                 dst->st_other = src->st_other;
247                 dst->st_shndx = src->st_shndx;
248                 dst->st_value = src->st_value;
249                 dst->st_size = src->st_size;
250         }
251
252         return (dst);
253 }
```

```
255 static GElf_Sym *
256 dt_module_symgelf64(const Elf64_Sym *src, GElf_Sym *dst)
257 {
258         if (dst != NULL)
259                 bcopy(src, dst, sizeof (GElf_Sym));
260
261         return (dst);
262 }
263
264 static GElf_Sym *
265 dt_module_symname32(dt_module_t *dmp, const char *name,
266     GElf_Sym *symp, uint_t *idp)
267 {
268         const Elf32_Sym *symtab = dmp->dm_symtab.cts_data;
269         const char *strtab = dmp->dm_strtab.cts_data;
270
271         const Elf32_Sym *sym;
272         const dt_sym_t *dsp;
273         uint_t i, h;
274
275         if (dmp->dm_nsymelems == 0)
276                 return (NULL);
277
278         h = dt_strtab_hash(name, NULL) % dmp->dm_nsymbuckets;
279
280         for (i = dmp->dm_symbuckets[h]; i != 0; i = dsp->ds_next) {
281                 dsp = &dmp->dm_symchains[i];
282                 sym = symtab + dsp->ds_symid;
283
284                 if (strcmp(name, strtab + sym->st_name) == 0) {
285                         if (idp != NULL)
286                                 *idp = dsp->ds_symid;
287                         return (dt_module_symgelf32(sym, symp));
288                 }
289         }
290
291         return (NULL);
292 }
293
294 static GElf_Sym *
295 dt_module_symname64(dt_module_t *dmp, const char *name,
296     GElf_Sym *symp, uint_t *idp)
297 {
298         const Elf64_Sym *symtab = dmp->dm_symtab.cts_data;
299         const char *strtab = dmp->dm_strtab.cts_data;
300
301         const Elf64_Sym *sym;
302         const dt_sym_t *dsp;
303         uint_t i, h;
304
305         if (dmp->dm_nsymelems == 0)
306                 return (NULL);
307
308         h = dt_strtab_hash(name, NULL) % dmp->dm_nsymbuckets;
309
310         for (i = dmp->dm_symbuckets[h]; i != 0; i = dsp->ds_next) {
311                 dsp = &dmp->dm_symchains[i];
312                 sym = symtab + dsp->ds_symid;
313
314                 if (strcmp(name, strtab + sym->st_name) == 0) {
315                         if (idp != NULL)
316                                 *idp = dsp->ds_symid;
317                         return (dt_module_symgelf64(sym, symp));
318                 }
319         }
```



```
321         return (NULL);
322 }
323
324 static GElf_Sym *
325 dt_module_symaddr32(dt_module_t *dmp, GElf_Addr addr,
326     GElf_Sym *symp, uint_t *idp)
327 {
328         const Elf32_Sym **asmap = (const Elf32_Sym **)dmp->dm_asmap;
329         const Elf32_Sym *symtab = dmp->dm_symtab.cts_data;
330         const Elf32_Sym *sym;
331
332         uint_t i, mid, lo = 0, hi = dmp->dm_aslen - 1;
333         Elf32_Addr v;
334
335         if (dmp->dm_aslen == 0)
336                 return (NULL);
337
338         while (hi - lo > 1) {
339                 mid = (lo + hi) / 2;
340                 if (addr >= asmap[mid]->st_value)
341                         lo = mid;
342                 else
343                         hi = mid;
344         }
345
346         i = addr < asmap[hi]->st_value ? lo : hi;
347         sym = asmap[i];
348         v = sym->st_value;
349
350         /*
351          * If the previous entry has the same value, improve our choice.  The
352          * order of equal-valued symbols is determined by the comparison func.
353          */
354         while (i-- != 0 && asmap[i]->st_value == v)
355                 sym = asmap[i];
356
357         if (addr - sym->st_value < MAX(sym->st_size, 1)) {
358                 if (idp != NULL)
359                         *idp = (uint_t)(sym - symtab);
360                 return (dt_module_symgelf32(sym, symp));
361         }
362
363         return (NULL);
364 }
365
366 static GElf_Sym *
367 dt_module_symaddr64(dt_module_t *dmp, GElf_Addr addr,
368     GElf_Sym *symp, uint_t *idp)
369 {
370         const Elf64_Sym **asmap = (const Elf64_Sym **)dmp->dm_asmap;
371         const Elf64_Sym *symtab = dmp->dm_symtab.cts_data;
372         const Elf64_Sym *sym;
373
374         uint_t i, mid, lo = 0, hi = dmp->dm_aslen - 1;
375         Elf64_Addr v;
376
377         if (dmp->dm_aslen == 0)
378                 return (NULL);
379
380         while (hi - lo > 1) {
381                 mid = (lo + hi) / 2;
382                 if (addr >= asmap[mid]->st_value)
383                         lo = mid;
384                 else
385                         hi = mid;
386         }
```

```
388         i = addr < asmap[hi]->st_value ? lo : hi;
389         sym = asmap[i];
390         v = sym->st_value;

392         /*
393          * If the previous entry has the same value, improve our choice.  The
394          * order of equal-valued symbols is determined by the comparison func.
395          */
396         while (i-- != 0 && asmap[i]->st_value == v)
397                 sym = asmap[i];

399         if (addr - sym->st_value < MAX(sym->st_size, 1)) {
400                 if (idp != NULL)
401                         *idp = (uint_t)(sym - symtab);
402                 return (dt_module_symgelf64(sym, symp));
403         }

405         return (NULL);
406 }

408 static const dt_modops_t dt_modops_32 = {
409         dt_module_syminit32,
410         dt_module_symsort32,
411         dt_module_symname32,
412         dt_module_symaddr32
413 };

415 static const dt_modops_t dt_modops_64 = {
416         dt_module_syminit64,
417         dt_module_symsort64,
418         dt_module_symname64,
419         dt_module_symaddr64
420 };

422 dt_module_t *
423 dt_module_create(dtrace_hdl_t *dtp, const char *name)
424 {
425         long pid;
426         char *eptr;
427         dt_ident_t *idp;
428 #endif /* ! codereview */
429         uint_t h = dt_strtab_hash(name, NULL) % dtp->dt_modbuckets;
430         dt_module_t *dmp;

432         for (dmp = dtp->dt_mods[h]; dmp != NULL; dmp = dmp->dm_next) {
433                 if (strcmp(dmp->dm_name, name) == 0)
434                         return (dmp);
435         }

437         if ((dmp = malloc(sizeof (dt_module_t))) == NULL)
438                 return (NULL); /* caller must handle allocation failure */

440         bzero(dmp, sizeof (dt_module_t));
441         (void) strlcpy(dmp->dm_name, name, sizeof (dmp->dm_name));
442         dt_list_append(&dtp->dt_modlist, dmp);
443         dmp->dm_next = dtp->dt_mods[h];
444         dtp->dt_mods[h] = dmp;
445         dtp->dt_nmods++;

447         if (dtp->dt_conf.dtc_ctfmodel == CTF_MODEL_LP64)
448                 dmp->dm_ops = &dt_modops_64;
449         else
450                 dmp->dm_ops = &dt_modops_32;

452         /*
```



```
453          * Modules for userland processes are special. They always refer to a
454          * specific process and have a copy of their CTF data from a specific
455          * instant in time. Any dt_module_t that begins with 'pid' is a module
456          * for a specific process, much like how any probe description that
457          * begins with 'pid' is special. pid123 refers to process 123. A module
458          * that is just 'pid' refers specifically to pid$target. This is
459          * generally done as D does not currently allow for macros to be
460          * evaluated when working with types.
461          */
462         if (strncmp(dmp->dm_name, "pid", 3) == 0) {
463                 errno = 0;
464                 if (dmp->dm_name[3] == '\0') {
465                         idp = dt_idhash_lookup(dtp->dt_macros, "target");
466                         if (idp != NULL && idp->di_id != 0)
467                                 dmp->dm_pid = idp->di_id;
468                 } else {
469                         pid = strtol(dmp->dm_name + 3, &eptr, 10);
470                         if (errno == 0 && *eptr == '\0')
471                                 dmp->dm_pid = (pid_t)pid;
472                         else
473                                 dt_dprintf("encountered malformed pid "
474                                     "module: %s\n", dmp->dm_name);
475                 }
476         }

478 #endif /* ! codereview */
479         return (dmp);
480 }

482 dt_module_t *
483 dt_module_lookup_by_name(dtrace_hdl_t *dtp, const char *name)
484 {
485         uint_t h = dt_strtab_hash(name, NULL) % dtp->dt_modbuckets;
486         dt_module_t *dmp;

488         for (dmp = dtp->dt_mods[h]; dmp != NULL; dmp = dmp->dm_next) {
489                 if (strcmp(dmp->dm_name, name) == 0)
490                         return (dmp);
491         }

493         return (NULL);
494 }

496 /*ARGSUSED*/
497 dt_module_t *
498 dt_module_lookup_by_ctf(dtrace_hdl_t *dtp, ctf_file_t *ctfp)
499 {
500         return (ctfp ? ctf_getspecific(ctfp) : NULL);
501 }

503 static int
504 dt_module_load_sect(dtrace_hdl_t *dtp, dt_module_t *dmp, ctf_sect_t *ctsp)
505 {
506         const char *s;
507         size_t shstrs;
508         GElf_Shdr sh;
509         Elf_Data *dp;
510         Elf_Scn *sp;

512         if (elf_getshdrstrndx(dmp->dm_elf, &shstrs) == -1)
513                 return (dt_set_errno(dtp, EDT_NOTLOADED));

515         for (sp = NULL; (sp = elf_nextscn(dmp->dm_elf, sp)) != NULL; ) {
516                 if (gelf_getshdr(sp, &sh) == NULL || sh.sh_type == SHT_NULL ||
517                     (s = elf_strptr(dmp->dm_elf, shstrs, sh.sh_name)) == NULL)
518                         continue; /* skip any malformed sections */
```

```
520                 if (sh.sh_type == ctsp->cts_type &&
521                     sh.sh_entsize == ctsp->cts_entsize &&
522                     strcmp(s, ctsp->cts_name) == 0)
523                         break; /* section matches specification */
524         }
525
526         /*
527          * If the section isn't found, return success but leave cts_data set
528          * to NULL and cts_size set to zero for our caller.
529          */
530         if (sp == NULL || (dp = elf_getdata(sp, NULL)) == NULL)
531                 return (0);
532
533         ctsp->cts_data = dp->d_buf;
534         ctsp->cts_size = dp->d_size;
535
536         dt_dprintf("loaded %s [%s] (%lu bytes)\n",
537             dmp->dm_name, ctsp->cts_name, (ulong_t)ctsp->cts_size);
538
539         return (0);
540 }
541
542 typedef struct dt_module_cb_arg {
543         struct ps_prochandle *dpa_proc;
544         dtrace_hdl_t *dpa_dtp;
545         dt_module_t *dpa_dmp;
546         uint_t dpa_count;
547 } dt_module_cb_arg_t;
548
549 /* ARGSUSED */
550 static int
551 dt_module_load_proc_count(void *arg, const prmap_t *prmap, const char *obj)
552 {
553         ctf_file_t *fp;
554         dt_module_cb_arg_t *dcp = arg;
555
556         /* Try to grab a ctf container if it exists */
557         fp = Pname_to_ctf(dcp->dpa_proc, obj);
558         if (fp != NULL)
559                 dcp->dpa_count++;
560         return (0);
561 }
562
563 /* ARGSUSED */
564 static int
565 dt_module_load_proc_build(void *arg, const prmap_t *prmap, const char *obj)
566 {
567         ctf_file_t *fp;
568         char buf[MAXPATHLEN], *p;
569         dt_module_cb_arg_t *dcp = arg;
570         int count = dcp->dpa_count;
571         Lmid_t lmid;
572
573         fp = Pname_to_ctf(dcp->dpa_proc, obj);
574         if (fp == NULL)
575                 return (0);
576         fp = ctf_dup(fp);
577         if (fp == NULL)
578                 return (0);
579         dcp->dpa_dmp->dm_libctfp[count] = fp;
580         /*
581          * While it'd be nice to simply use objname here, because of our prior
582          * actions we'll always get a resolved object name to its on disk file.
583          * Like the pid provider, we need to tell a bit of a lie here. The type
584          * that the user thinks of is in terms of the libraries they requested,
```



```
585          * eg. libc.so.1, they don't care about the fact that it's
586          * libc_hwcap.so.1.
587          */
588         (void) Pobjname(dcp->dpa_proc, prmap->pr_vaddr, buf, sizeof (buf));
589         if ((p = strrchr(buf, '/')) == NULL)
590                 p = buf;
591         else
592                 p++;
593
594         /*
595          * If for some reason we can't find a link map id for this module, which
596          * would be really quite weird. We instead just say the link map id is
597          * zero.
598          */
599         if (Plmid(dcp->dpa_proc, prmap->pr_vaddr, &lmid) != 0)
600                 lmid = 0;
601
602         if (lmid == 0)
603                 dcp->dpa_dmp->dm_libctfn[count] = strdup(p);
604         else
605                 (void) asprintf(&dcp->dpa_dmp->dm_libctfn[count],
606                     "LM%lx`%s", lmid, p);
607         if (dcp->dpa_dmp->dm_libctfn[count] == NULL)
608                 return (1);
609         ctf_setspecific(fp, dcp->dpa_dmp);
610         dcp->dpa_count++;
611         return (0);
612 }
613
614 /*
615  * We've been asked to load data that belongs to another process. As such we're
616  * going to pgrab it at this instant, load everything that we might ever care
617  * about, and then drive on. The reason for this is that the process that we're
618  * interested in might be changing. As long as we have grabbed it, then this
619  * can't be a problem for us.
620  *
621  * For now, we're actually going to punt on most things and just try to get CTF
622  * data, nothing else. Basically this is only useful as a source of type
623  * information, we can't go and do the stacktrace lookups, etc.
624  */
625 static int
626 dt_module_load_proc(dtrace_hdl_t *dtp, dt_module_t *dmp)
627 {
628         struct ps_prochandle *p;
629         dt_module_cb_arg_t arg;
630
631         /*
632          * Note that on success we do not release this hold. We must hold this
633          * for our life time.
634          */
635         p = dt_proc_grab(dtp, dmp->dm_pid, 0, PGRAB_RDONLY | PGRAB_FORCE);
636         if (p == NULL) {
637                 dt_dprintf("failed to grab pid: %d\n", (int)dmp->dm_pid);
638                 return (dt_set_errno(dtp, EDT_CANTLOAD));
639         }
640         dt_proc_lock(dtp, p);
641
642         arg.dpa_proc = p;
643         arg.dpa_dtp = dtp;
644         arg.dpa_dmp = dmp;
645         arg.dpa_count = 0;
646         if (Pobject_iter_resolved(p, dt_module_load_proc_count, &arg) != 0) {
647                 dt_dprintf("failed to iterate objects\n");
648                 dt_proc_release(dtp, p);
649                 return (dt_set_errno(dtp, EDT_CANTLOAD));
650         }
```

```
652         if (arg.dpa_count == 0) {
653                 dt_dprintf("no ctf data present\n");
654                 dt_proc_unlock(dtp, p);
655                 dt_proc_release(dtp, p);
656                 return (dt_set_errno(dtp, EDT_CANTLOAD));
657         }

659         dmp->dm_libctfp = malloc(sizeof (ctf_file_t *) * arg.dpa_count);
660         if (dmp->dm_libctfp == NULL) {
661                 dt_proc_unlock(dtp, p);
662                 dt_proc_release(dtp, p);
663                 return (dt_set_errno(dtp, EDT_NOMEM));
664         }
665         bzero(dmp->dm_libctfp, sizeof (ctf_file_t *) * arg.dpa_count);

667         dmp->dm_libctfn = malloc(sizeof (char *) * arg.dpa_count);
668         if (dmp->dm_libctfn == NULL) {
669                 free(dmp->dm_libctfp);
670                 dt_proc_unlock(dtp, p);
671                 dt_proc_release(dtp, p);
672                 return (dt_set_errno(dtp, EDT_NOMEM));
673         }
674         bzero(dmp->dm_libctfn, sizeof (char *) * arg.dpa_count);

676         dmp->dm_nctflibs = arg.dpa_count;

678         arg.dpa_count = 0;
679         if (Pobject_iter_resolved(p, dt_module_load_proc_build, &arg) != 0) {
680                 dt_proc_unlock(dtp, p);
681                 dt_module_unload(dtp, dmp);
682                 dt_proc_release(dtp, p);
683                 return (dt_set_errno(dtp, EDT_CANTLOAD));
684         }
685         assert(arg.dpa_count == dmp->dm_nctflibs);
686         dt_dprintf("loaded %d ctf modules for pid %d\n", arg.dpa_count,
687             (int)dmp->dm_pid);

689         dt_proc_unlock(dtp, p);
690         dt_proc_release(dtp, p);
691         dmp->dm_flags |= DT_DM_LOADED;

693         return (0);
694 }

696 #endif /* ! codereview */
697 int
698 dt_module_load(dtrace_hdl_t *dtp, dt_module_t *dmp)
699 {
700         if (dmp->dm_flags & DT_DM_LOADED)
701                 return (0); /* module is already loaded */

703         if (dmp->dm_pid != 0)
704                 return (dt_module_load_proc(dtp, dmp));

706 #endif /* ! codereview */
707         dmp->dm_ctdata.cts_name = ".SUNW_ctf";
708         dmp->dm_ctdata.cts_type = SHT_PROGBITS;
709         dmp->dm_ctdata.cts_flags = 0;
710         dmp->dm_ctdata.cts_data = NULL;
711         dmp->dm_ctdata.cts_size = 0;
712         dmp->dm_ctdata.cts_entsize = 0;
713         dmp->dm_ctdata.cts_offset = 0;

715         dmp->dm_symtab.cts_name = ".symtab";
716         dmp->dm_symtab.cts_type = SHT_SYMTAB;
```



```
717         dmp->dm_symtab.cts_flags = 0;
718         dmp->dm_symtab.cts_data = NULL;
719         dmp->dm_symtab.cts_size = 0;
720         dmp->dm_symtab.cts_entsize = dmp->dm_ops == &dt_modops_64 ?
721             sizeof (Elf64_Sym) : sizeof (Elf32_Sym);
722         dmp->dm_symtab.cts_offset = 0;

724         dmp->dm_strtab.cts_name = ".strtab";
725         dmp->dm_strtab.cts_type = SHT_STRTAB;
726         dmp->dm_strtab.cts_flags = 0;
727         dmp->dm_strtab.cts_data = NULL;
728         dmp->dm_strtab.cts_size = 0;
729         dmp->dm_strtab.cts_entsize = 0;
730         dmp->dm_strtab.cts_offset = 0;

732         /*
733          * Attempt to load the module's CTF section, symbol table section, and
734          * string table section.  Note that modules may not contain CTF data:
735          * this will result in a successful load_sect but data of size zero.
736          * We will then fail if dt_module_getctf() is called, as shown below.
737          */
738         if (dt_module_load_sect(dtp, dmp, &dmp->dm_ctdata) == -1 ||
739             dt_module_load_sect(dtp, dmp, &dmp->dm_symtab) == -1 ||
740             dt_module_load_sect(dtp, dmp, &dmp->dm_strtab) == -1) {
741                 dt_module_unload(dtp, dmp);
742                 return (-1); /* dt_errno is set for us */
743         }

745         /*
746          * Allocate the hash chains and hash buckets for symbol name lookup.
747          * This is relatively simple since the symbol table is of fixed size
748          * and is known in advance.  We allocate one extra element since we
749          * use element indices instead of pointers and zero is our sentinel.
750          */
751         dmp->dm_nsymelems =
752             dmp->dm_symtab.cts_size / dmp->dm_symtab.cts_entsize;

754         dmp->dm_nsymbuckets = _dtrace_strbuckets;
755         dmp->dm_symfree = 1;            /* first free element is index 1 */

757         dmp->dm_symbuckets = malloc(sizeof (uint_t) * dmp->dm_nsymbuckets);
758         dmp->dm_symchains = malloc(sizeof (dt_sym_t) * dmp->dm_nsymelems + 1);

760         if (dmp->dm_symbuckets == NULL || dmp->dm_symchains == NULL) {
761                 dt_module_unload(dtp, dmp);
762                 return (dt_set_errno(dtp, EDT_NOMEM));
763         }

765         bzero(dmp->dm_symbuckets, sizeof (uint_t) * dmp->dm_nsymbuckets);
766         bzero(dmp->dm_symchains, sizeof (dt_sym_t) * dmp->dm_nsymelems + 1);

768         /*
769          * Iterate over the symbol table data buffer and insert each symbol
770          * name into the name hash if the name and type are valid.  Then
771          * allocate the address map, fill it in, and sort it.
772          */
773         dmp->dm_asrsv = dmp->dm_ops->do_syminit(dmp);

775         dt_dprintf("hashed %s [%s] (%u symbols)\n",
776             dmp->dm_name, dmp->dm_symtab.cts_name, dmp->dm_symfree - 1);

778         if ((dmp->dm_asmap = malloc(sizeof (void *) * dmp->dm_asrsv)) == NULL) {
779                 dt_module_unload(dtp, dmp);
780                 return (dt_set_errno(dtp, EDT_NOMEM));
781         }
```

```
783         dmp->dm_ops->do_symsort(dmp);
785         dt_dprintf("sorted %s [%s] (%u symbols)\n",
786             dmp->dm_name, dmp->dm_symtab.cts_name, dmp->dm_aslen);
788         dmp->dm_flags |= DT_DM_LOADED;
789         return (0);
790 }
792 int
793 dt_module_hasctf(dtrace_hdl_t *dtp, dt_module_t *dmp)
794 {
795         if (dmp->dm_pid != 0 && dmp->dm_nctflibs > 0)
796                 return (1);
797         return (dt_module_getctf(dtp, dmp) != NULL);
798 }
800 #endif /* ! codereview */
801 ctf_file_t *
802 dt_module_getctf(dtrace_hdl_t *dtp, dt_module_t *dmp)
803 {
804         const char *parent;
805         dt_module_t *pmp;
806         ctf_file_t *pfp;
807         int model;
809         if (dmp->dm_ctfp != NULL || dt_module_load(dtp, dmp) != 0)
810                 return (dmp->dm_ctfp);
812         if (dmp->dm_ops == &dt_modops_64)
813                 model = CTF_MODEL_LP64;
814         else
815                 model = CTF_MODEL_ILP32;
817         /*
818          * If the data model of the module does not match our program data
819          * model, then do not permit CTF from this module to be opened and
820          * returned to the compiler.  If we support mixed data models in the
821          * future for combined kernel/user tracing, this can be removed.
822          */
823         if (dtp->dt_conf.dtc_ctfmodel != model) {
824                 (void) dt_set_errno(dtp, EDT_DATAMODEL);
825                 return (NULL);
826         }
828         if (dmp->dm_ctdata.cts_size == 0) {
829                 (void) dt_set_errno(dtp, EDT_NOCTF);
830                 return (NULL);
831         }
833         dmp->dm_ctfp = ctf_bufopen(&dmp->dm_ctdata,
834             &dmp->dm_symtab, &dmp->dm_strtab, &dtp->dt_ctferr);
836         if (dmp->dm_ctfp == NULL) {
837                 (void) dt_set_errno(dtp, EDT_CTF);
838                 return (NULL);
839         }
841         (void) ctf_setmodel(dmp->dm_ctfp, model);
842         ctf_setspecific(dmp->dm_ctfp, dmp);
844         if ((parent = ctf_parent_name(dmp->dm_ctfp)) != NULL) {
845                 if ((pmp = dt_module_create(dtp, parent)) == NULL ||
846                     (pfp = dt_module_getctf(dtp, pmp)) == NULL) {
847                         if (pmp == NULL)
848                                 (void) dt_set_errno(dtp, EDT_NOMEM);
```



```
849                         goto err;
850                 }
852                 if (ctf_import(dmp->dm_ctfp, pfp) == CTF_ERR) {
853                         dtp->dt_ctferr = ctf_errno(dmp->dm_ctfp);
854                         (void) dt_set_errno(dtp, EDT_CTF);
855                         goto err;
856                 }
857         }
859         dt_dprintf("loaded CTF container for %s (%p)\n",
860             dmp->dm_name, (void *)dmp->dm_ctfp);
862         return (dmp->dm_ctfp);
864 err:
865         ctf_close(dmp->dm_ctfp);
866         dmp->dm_ctfp = NULL;
867         return (NULL);
868 }
870 /*ARGSUSED*/
871 void
872 dt_module_unload(dtrace_hdl_t *dtp, dt_module_t *dmp)
873 {
874         int i;
876 #endif /* ! codereview */
877         ctf_close(dmp->dm_ctfp);
878         dmp->dm_ctfp = NULL;
880         if (dmp->dm_libctfp != NULL) {
881                 for (i = 0; i < dmp->dm_nctflibs; i++) {
882                         ctf_close(dmp->dm_libctfp[i]);
883                         free(dmp->dm_libctfn[i]);
884                 }
885                 free(dmp->dm_libctfp);
886                 free(dmp->dm_libctfn);
887                 dmp->dm_libctfp = NULL;
888                 dmp->dm_nctflibs = 0;
889         }
891 #endif /* ! codereview */
892         bzero(&dmp->dm_ctdata, sizeof (ctf_sect_t));
893         bzero(&dmp->dm_symtab, sizeof (ctf_sect_t));
894         bzero(&dmp->dm_strtab, sizeof (ctf_sect_t));
896         if (dmp->dm_symbuckets != NULL) {
897                 free(dmp->dm_symbuckets);
898                 dmp->dm_symbuckets = NULL;
899         }
901         if (dmp->dm_symchains != NULL) {
902                 free(dmp->dm_symchains);
903                 dmp->dm_symchains = NULL;
904         }
906         if (dmp->dm_asmap != NULL) {
907                 free(dmp->dm_asmap);
908                 dmp->dm_asmap = NULL;
909         }
911         dmp->dm_symfree = 0;
912         dmp->dm_nsymbuckets = 0;
913         dmp->dm_nsymelems = 0;
914         dmp->dm_asrsv = 0;
```

```
915         dmp->dm_aslen = 0;

917         dmp->dm_text_va = NULL;
918         dmp->dm_text_size = 0;
919         dmp->dm_data_va = NULL;
920         dmp->dm_data_size = 0;
921         dmp->dm_bss_va = NULL;
922         dmp->dm_bss_size = 0;

924         if (dmp->dm_extern != NULL) {
925                 dt_idhash_destroy(dmp->dm_extern);
926                 dmp->dm_extern = NULL;
927         }

929         (void) elf_end(dmp->dm_elf);
930         dmp->dm_elf = NULL;

932         dmp->dm_pid = 0;

934 #endif /* ! codereview */
935         dmp->dm_flags &= ~DT_DM_LOADED;
936 }

938 void
939 dt_module_destroy(dtrace_hdl_t *dtp, dt_module_t *dmp)
940 {
941         uint_t h = dt_strtab_hash(dmp->dm_name, NULL) % dtp->dt_modbuckets;
942         dt_module_t **dmpp = &dtp->dt_mods[h];

944         dt_list_delete(&dtp->dt_modlist, dmp);
945         assert(dtp->dt_nmods != 0);
946         dtp->dt_nmods--;

948         /*
949          * Now remove this module from its hash chain.  We expect to always
950          * find the module on its hash chain, so in this loop we assert that
951          * we don't run off the end of the list.
952          */
953         while (*dmpp != dmp) {
954                 dmpp = &((*dmpp)->dm_next);
955                 assert(*dmpp != NULL);
956         }

958         *dmpp = dmp->dm_next;

960         dt_module_unload(dtp, dmp);
961         free(dmp);
962 }

964 /*
965  * Insert a new external symbol reference into the specified module.  The new
966  * symbol will be marked as undefined and is assigned a symbol index beyond
967  * any existing cached symbols from this module.  We use the ident's di_data
968  * field to store a pointer to a copy of the dtrace_syminfo_t for this symbol.
969  */
970 dt_ident_t *
971 dt_module_extern(dtrace_hdl_t *dtp, dt_module_t *dmp,
972     const char *name, const dtrace_typeinfo_t *tip)
973 {
974         dtrace_syminfo_t *sip;
975         dt_ident_t *idp;
976         uint_t id;

978         if (dmp->dm_extern == NULL && (dmp->dm_extern = dt_idhash_create(
979             "extern", NULL, dmp->dm_nsymelems, UINT_MAX)) == NULL) {
980                 (void) dt_set_errno(dtp, EDT_NOMEM);
```



```
981                 return (NULL);
982         }

984         if (dt_idhash_nextid(dmp->dm_extern, &id) == -1) {
985                 (void) dt_set_errno(dtp, EDT_SYMOFLOW);
986                 return (NULL);
987         }

989         if ((sip = malloc(sizeof (dtrace_syminfo_t))) == NULL) {
990                 (void) dt_set_errno(dtp, EDT_NOMEM);
991                 return (NULL);
992         }

994         idp = dt_idhash_insert(dmp->dm_extern, name, DT_IDENT_SYMBOL, 0, id,
995             _dtrace_symattr, 0, &dt_idops_thaw, NULL, dtp->dt_gen);

997         if (idp == NULL) {
998                 (void) dt_set_errno(dtp, EDT_NOMEM);
999                 free(sip);
1000                return (NULL);
1001        }

1003        sip->dts_object = dmp->dm_name;
1004        sip->dts_name = idp->di_name;
1005        sip->dts_id = idp->di_id;

1007        idp->di_data = sip;
1008        idp->di_ctfp = tip->dtt_ctfp;
1009        idp->di_type = tip->dtt_type;

1011        return (idp);
1012 }

1014 const char *
1015 dt_module_modelname(dt_module_t *dmp)
1016 {
1017         if (dmp->dm_ops == &dt_modops_64)
1018                 return ("64-bit");
1019         else
1020                 return ("32-bit");
1021 }

1023 /* ARGSUSED */
1024 int
1025 dt_module_getlibid(dtrace_hdl_t *dtp, dt_module_t *dmp, const ctf_file_t *fp)
1026 {
1027         int i;

1029         for (i = 0; i < dmp->dm_nctflibs; i++) {
1030                 if (dmp->dm_libctfp[i] == fp)
1031                         return (i);
1032         }

1034         return (-1);
1035 }

1037 /* ARGSUSED */
1038 ctf_file_t *
1039 dt_module_getctflib(dtrace_hdl_t *dtp, dt_module_t *dmp, const char *name)
1040 {
1041         int i;

1043         for (i = 0; i < dmp->dm_nctflibs; i++) {
1044                 if (strcmp(dmp->dm_libctfn[i], name) == 0)
1045                         return (dmp->dm_libctfp[i]);
1046         }
```

```
1048         return (NULL);
1049 }

1051 #endif /* ! codereview */
1052 /*
1053  * Update our module cache by adding an entry for the specified module 'name'.
1054  * We create the dt_module_t and populate it using /system/object/<name>/.
1055  */
1056 static void
1057 dt_module_update(dtrace_hdl_t *dtp, const char *name)
1058 {
1059         char fname[MAXPATHLEN];
1060         struct stat64 st;
1061         int fd, err, bits;

1063         dt_module_t *dmp;
1064         const char *s;
1065         size_t shstrs;
1066         GElf_Shdr sh;
1067         Elf_Data *dp;
1068         Elf_Scn *sp;

1070         (void) snprintf(fname, sizeof (fname),
1071             "%s/%s/object", OBJFS_ROOT, name);

1073         if ((fd = open(fname, O_RDONLY)) == -1 || fstat64(fd, &st) == -1 ||
1074             (dmp = dt_module_create(dtp, name)) == NULL) {
1075                 dt_dprintf("failed to open %s: %s\n", fname, strerror(errno));
1076                 (void) close(fd);
1077                 return;
1078         }

1080         /*
1081          * Since the module can unload out from under us (and /system/object
1082          * will return ENOENT), tell libelf to cook the entire file now and
1083          * then close the underlying file descriptor immediately.  If this
1084          * succeeds, we know that we can continue safely using dmp->dm_elf.
1085          */
1086         dmp->dm_elf = elf_begin(fd, ELF_C_READ, NULL);
1087         err = elf_cntl(dmp->dm_elf, ELF_C_FDREAD);
1088         (void) close(fd);

1090         if (dmp->dm_elf == NULL || err == -1 ||
1091             elf_getshdrstrndx(dmp->dm_elf, &shstrs) == -1) {
1092                 dt_dprintf("failed to load %s: %s\n",
1093                     fname, elf_errmsg(elf_errno()));
1094                 dt_module_destroy(dtp, dmp);
1095                 return;
1096         }

1098         switch (gelf_getclass(dmp->dm_elf)) {
1099         case ELFCLASS32:
1100                 dmp->dm_ops = &dt_modops_32;
1101                 bits = 32;
1102                 break;
1103         case ELFCLASS64:
1104                 dmp->dm_ops = &dt_modops_64;
1105                 bits = 64;
1106                 break;
1107         default:
1108                 dt_dprintf("failed to load %s: unknown ELF class\n", fname);
1109                 dt_module_destroy(dtp, dmp);
1110                 return;
1111         }
```



```
1113         /*
1114          * Iterate over the section headers locating various sections of
1115          * interest and use their attributes to flesh out the dt_module_t.
1116          */
1117         for (sp = NULL; (sp = elf_nextscn(dmp->dm_elf, sp)) != NULL; ) {
1118                 if (gelf_getshdr(sp, &sh) == NULL || sh.sh_type == SHT_NULL ||
1119                     (s = elf_strptr(dmp->dm_elf, shstrs, sh.sh_name)) == NULL)
1120                         continue; /* skip any malformed sections */

1122                 if (strcmp(s, ".text") == 0) {
1123                         dmp->dm_text_size = sh.sh_size;
1124                         dmp->dm_text_va = sh.sh_addr;
1125                 } else if (strcmp(s, ".data") == 0) {
1126                         dmp->dm_data_size = sh.sh_size;
1127                         dmp->dm_data_va = sh.sh_addr;
1128                 } else if (strcmp(s, ".bss") == 0) {
1129                         dmp->dm_bss_size = sh.sh_size;
1130                         dmp->dm_bss_va = sh.sh_addr;
1131                 } else if (strcmp(s, ".info") == 0 &&
1132                     (dp = elf_getdata(sp, NULL)) != NULL) {
1133                         bcopy(dp->d_buf, &dmp->dm_info,
1134                             MIN(sh.sh_size, sizeof (dmp->dm_info)));
1135                 } else if (strcmp(s, ".filename") == 0 &&
1136                     (dp = elf_getdata(sp, NULL)) != NULL) {
1137                         (void) strlcpy(dmp->dm_file,
1138                             dp->d_buf, sizeof (dmp->dm_file));
1139                 }
1140         }

1142         dmp->dm_flags |= DT_DM_KERNEL;
1143         dmp->dm_modid = (int)OBJFS_MODID(st.st_ino);

1145         if (dmp->dm_info.objfs_info_primary)
1146                 dmp->dm_flags |= DT_DM_PRIMARY;

1148         dt_dprintf("opened %d-bit module %s (%s) [%d]\n",
1149             bits, dmp->dm_name, dmp->dm_file, dmp->dm_modid);
1150 }

1152 /*
1153  * Unload all the loaded modules and then refresh the module cache with the
1154  * latest list of loaded modules and their address ranges.
1155  */
1156 void
1157 dtrace_update(dtrace_hdl_t *dtp)
1158 {
1159         dt_module_t *dmp;
1160         DIR *dirp;

1162         for (dmp = dt_list_next(&dtp->dt_modlist);
1163             dmp != NULL; dmp = dt_list_next(dmp))
1164                 dt_module_unload(dtp, dmp);

1166         /*
1167          * Open /system/object and attempt to create a libdtrace module for
1168          * each kernel module that is loaded on the current system.
1169          */
1170         if (!(dtp->dt_oflags & DTRACE_O_NOSYS) &&
1171             (dirp = opendir(OBJFS_ROOT)) != NULL) {
1172                 struct dirent *dp;

1174                 while ((dp = readdir(dirp)) != NULL) {
1175                         if (dp->d_name[0] != '.')
1176                                 dt_module_update(dtp, dp->d_name);
1177                 }
```

```
1179                (void) closedir(dirp);
1180        }
1181
1182        /*
1183         * Look up all the macro identifiers and set di_id to the latest value.
1184         * This code collaborates with dt_lex.l on the use of di_id.  We will
1185         * need to implement something fancier if we need to support non-ints.
1186         */
1187        dt_idhash_lookup(dtp->dt_macros, "egid")->di_id = getegid();
1188        dt_idhash_lookup(dtp->dt_macros, "euid")->di_id = geteuid();
1189        dt_idhash_lookup(dtp->dt_macros, "gid")->di_id = getgid();
1190        dt_idhash_lookup(dtp->dt_macros, "pid")->di_id = getpid();
1191        dt_idhash_lookup(dtp->dt_macros, "pgid")->di_id = getpgid(0);
1192        dt_idhash_lookup(dtp->dt_macros, "ppid")->di_id = getppid();
1193        dt_idhash_lookup(dtp->dt_macros, "projid")->di_id = getprojid();
1194        dt_idhash_lookup(dtp->dt_macros, "sid")->di_id = getsid(0);
1195        dt_idhash_lookup(dtp->dt_macros, "taskid")->di_id = gettaskid();
1196        dt_idhash_lookup(dtp->dt_macros, "uid")->di_id = getuid();
1197
1198        /*
1199         * Cache the pointers to the modules representing the base executable
1200         * and the run-time linker in the dtrace client handle. Note that on
1201         * x86 krtld is folded into unix, so if we don't find it, use unix
1202         * instead.
1203         */
1204        dtp->dt_exec = dt_module_lookup_by_name(dtp, "genunix");
1205        dtp->dt_rtld = dt_module_lookup_by_name(dtp, "krtld");
1206        if (dtp->dt_rtld == NULL)
1207                dtp->dt_rtld = dt_module_lookup_by_name(dtp, "unix");
1208
1209        /*
1210         * If this is the first time we are initializing the module list,
1211         * remove the module for genunix from the module list and then move it
1212         * to the front of the module list.  We do this so that type and symbol
1213         * queries encounter genunix and thereby optimize for the common case
1214         * in dtrace_lookup_by_name() and dtrace_lookup_by_type(), below.
1215         */
1216        if (dtp->dt_exec != NULL &&
1217            dtp->dt_cdefs == NULL && dtp->dt_ddefs == NULL) {
1218                dt_list_delete(&dtp->dt_modlist, dtp->dt_exec);
1219                dt_list_prepend(&dtp->dt_modlist, dtp->dt_exec);
1220        }
1221 }
1222
1223 static dt_module_t *
1224 dt_module_from_object(dtrace_hdl_t *dtp, const char *object)
1225 {
1226        int err = EDT_NOMOD;
1227        dt_module_t *dmp;
1228
1229        switch ((uintptr_t)object) {
1230        case (uintptr_t)DTRACE_OBJ_EXEC:
1231                dmp = dtp->dt_exec;
1232                break;
1233        case (uintptr_t)DTRACE_OBJ_RTLD:
1234                dmp = dtp->dt_rtld;
1235                break;
1236        case (uintptr_t)DTRACE_OBJ_CDEFS:
1237                dmp = dtp->dt_cdefs;
1238                break;
1239        case (uintptr_t)DTRACE_OBJ_DDEFS:
1240                dmp = dtp->dt_ddefs;
1241                break;
1242        default:
1243                dmp = dt_module_create(dtp, object);
1244                err = EDT_NOMEM;
```



```
1245        }
1246
1247        if (dmp == NULL)
1248                (void) dt_set_errno(dtp, err);
1249
1250        return (dmp);
1251 }
1252
1253 /*
1254  * Exported interface to look up a symbol by name.  We return the GElf_Sym and
1255  * complete symbol information for the matching symbol.
1256  */
1257 int
1258 dtrace_lookup_by_name(dtrace_hdl_t *dtp, const char *object, const char *name,
1259     GElf_Sym *symp, dtrace_syminfo_t *sip)
1260 {
1261        dt_module_t *dmp;
1262        dt_ident_t *idp;
1263        uint_t n, id;
1264        GElf_Sym sym;
1265
1266        uint_t mask = 0; /* mask of dt_module flags to match */
1267        uint_t bits = 0; /* flag bits that must be present */
1268
1269        if (object != DTRACE_OBJ_EVERY &&
1270            object != DTRACE_OBJ_KMODS &&
1271            object != DTRACE_OBJ_UMODS) {
1272                if ((dmp = dt_module_from_object(dtp, object)) == NULL)
1273                        return (-1); /* dt_errno is set for us */
1274
1275                if (dt_module_load(dtp, dmp) == -1)
1276                        return (-1); /* dt_errno is set for us */
1277                n = 1;
1278
1279        } else {
1280                if (object == DTRACE_OBJ_KMODS)
1281                        mask = bits = DT_DM_KERNEL;
1282                else if (object == DTRACE_OBJ_UMODS)
1283                        mask = DT_DM_KERNEL;
1284
1285                dmp = dt_list_next(&dtp->dt_modlist);
1286                n = dtp->dt_nmods;
1287        }
1288
1289        if (symp == NULL)
1290                symp = &sym;
1291
1292        for (; n > 0; n--, dmp = dt_list_next(dmp)) {
1293                if ((dmp->dm_flags & mask) != bits)
1294                        continue; /* failed to match required attributes */
1295
1296                if (dt_module_load(dtp, dmp) == -1)
1297                        continue; /* failed to load symbol table */
1298
1299                if (dmp->dm_ops->do_symname(dmp, name, symp, &id) != NULL) {
1300                        if (sip != NULL) {
1301                                sip->dts_object = dmp->dm_name;
1302                                sip->dts_name = (const char *)
1303                                    dmp->dm_strtab.cts_data + symp->st_name;
1304                                sip->dts_id = id;
1305                        }
1306                        return (0);
1307                }
1308
1309                if (dmp->dm_extern != NULL &&
1310                    (idp = dt_idhash_lookup(dmp->dm_extern, name)) != NULL) {
```

```
1311                        if (symp != &sym) {
1312                                symp->st_name = (uintptr_t)idp->di_name;
1313                                symp->st_info =
1314                                    GELF_ST_INFO(STB_GLOBAL, STT_NOTYPE);
1315                                symp->st_other = 0;
1316                                symp->st_shndx = SHN_UNDEF;
1317                                symp->st_value = 0;
1318                                symp->st_size =
1319                                    ctf_type_size(idp->di_ctfp, idp->di_type);
1320                        }

1322                        if (sip != NULL) {
1323                                sip->dts_object = dmp->dm_name;
1324                                sip->dts_name = idp->di_name;
1325                                sip->dts_id = idp->di_id;
1326                        }

1328                        return (0);
1329                }
1330        }

1332        return (dt_set_errno(dtp, EDT_NOSYM));
1333 }

1335 /*
1336  * Exported interface to look up a symbol by address.  We return the GElf_Sym
1337  * and complete symbol information for the matching symbol.
1338  */
1339 int
1340 dtrace_lookup_by_addr(dtrace_hdl_t *dtp, GElf_Addr addr,
1341     GElf_Sym *symp, dtrace_syminfo_t *sip)
1342 {
1343        dt_module_t *dmp;
1344        uint_t id;
1345        const dtrace_vector_t *v = dtp->dt_vector;

1347        if (v != NULL)
1348                return (v->dtv_lookup_by_addr(dtp->dt_varg, addr, symp, sip));

1350        for (dmp = dt_list_next(&dtp->dt_modlist); dmp != NULL;
1351            dmp = dt_list_next(dmp)) {
1352                if (addr - dmp->dm_text_va < dmp->dm_text_size ||
1353                    addr - dmp->dm_data_va < dmp->dm_data_size ||
1354                    addr - dmp->dm_bss_va < dmp->dm_bss_size)
1355                        break;
1356        }

1358        if (dmp == NULL)
1359                return (dt_set_errno(dtp, EDT_NOSYMADDR));

1361        if (dt_module_load(dtp, dmp) == -1)
1362                return (-1); /* dt_errno is set for us */

1364        if (symp != NULL) {
1365                if (dmp->dm_ops->do_symaddr(dmp, addr, symp, &id) == NULL)
1366                        return (dt_set_errno(dtp, EDT_NOSYMADDR));
1367        }

1369        if (sip != NULL) {
1370                sip->dts_object = dmp->dm_name;

1372                if (symp != NULL) {
1373                        sip->dts_name = (const char *)
1374                            dmp->dm_strtab.cts_data + symp->st_name;
1375                        sip->dts_id = id;
1376                } else {
```



```
1377                        sip->dts_name = NULL;
1378                        sip->dts_id = 0;
1379                }
1380        }

1382        return (0);
1383 }

1385 int
1386 dtrace_lookup_by_type(dtrace_hdl_t *dtp, const char *object, const char *name,
1387     dtrace_typeinfo_t *tip)
1388 {
1389        dtrace_typeinfo_t ti;
1390        dt_module_t *dmp;
1391        int found = 0;
1392        ctf_id_t id;
1393        uint_t n, i;
  25        uint_t n;
1394        int justone;
1395        ctf_file_t *fp;
1396        char *buf, *p, *q;
1397 #endif /* ! codereview */

1399        uint_t mask = 0; /* mask of dt_module flags to match */
1400        uint_t bits = 0; /* flag bits that must be present */

1402        if (object != DTRACE_OBJ_EVERY &&
1403            object != DTRACE_OBJ_KMODS &&
1404            object != DTRACE_OBJ_UMODS) {
1405                if ((dmp = dt_module_from_object(dtp, object)) == NULL)
1406                        return (-1); /* dt_errno is set for us */

1408                if (dt_module_load(dtp, dmp) == -1)
1409                        return (-1); /* dt_errno is set for us */
1410                n = 1;
1411                justone = 1;

1412        } else {
1413                if (object == DTRACE_OBJ_KMODS)
1414                        mask = bits = DT_DM_KERNEL;
1415                else if (object == DTRACE_OBJ_UMODS)
1416                        mask = DT_DM_KERNEL;

1418                dmp = dt_list_next(&dtp->dt_modlist);
1419                n = dtp->dt_nmods;
1420                justone = 0;
1421        }

1423        if (tip == NULL)
1424                tip = &ti;

1426        for (; n > 0; n--, dmp = dt_list_next(dmp)) {
1427                if ((dmp->dm_flags & mask) != bits)
1428                        continue; /* failed to match required attributes */

1430                /*
1431                 * If we can't load the CTF container, continue on to the next
1432                 * module.  If our search was scoped to only one module then
1433                 * return immediately leaving dt_errno unmodified.
1434                 */
1435                if (dt_module_hasctf(dtp, dmp) == 0) {
  51                if (dt_module_getctf(dtp, dmp) == NULL) {
1436                        if (justone)
1437                                return (-1);
1438                        continue;
1439                }
```

```
1441                 /*
1442                  * Look up the type in the module's CTF container.  If our
1443                  * match is a forward declaration tag, save this choice in
1444                  * 'tip' and keep going in the hope that we will locate the
1445                  * underlying structure definition.  Otherwise just return.
1446                  */
1447                 if (dmp->dm_pid == 0) {
1448                         id = ctf_lookup_by_name(dmp->dm_ctfp, name);
1449                         fp = dmp->dm_ctfp;
1450                 } else {
1451                         if ((p = strchr(name, '`')) != NULL) {
1452                                 buf = strdup(name);
1453                                 if (buf == NULL)
1454                                         return (dt_set_errno(dtp, EDT_NOMEM));
1455                                 p = strchr(buf, '`');
1456                                 if ((q = strchr(p + 1, '`')) != NULL)
1457                                         p = q;
1458                                 *p = '\0';
1459                                 fp = dt_module_getctflib(dtp, dmp, buf);
1460                                 if (fp == NULL || (id = ctf_lookup_by_name(fp,
1461                                     p + 1)) == CTF_ERR)
1462                                         id = CTF_ERR;
1463                                 free(buf);
1464                         } else {
1465                                 for (i = 0; i < dmp->dm_nctflibs; i++) {
1466                                         fp = dmp->dm_libctfp[i];
1467                                         id = ctf_lookup_by_name(fp, name);
1468                                         if (id != CTF_ERR)
1469                                                 break;
1470                                 }
1471                         }
1472                 }
1473                 if (id != CTF_ERR) {
  63                 if ((id = ctf_lookup_by_name(dmp->dm_ctfp, name)) != CTF_ERR) {
1474                         tip->dtt_object = dmp->dm_name;
1475                         tip->dtt_ctfp = fp;
  65                     tip->dtt_ctfp = dmp->dm_ctfp;
1476                         tip->dtt_type = id;
1477                         if (ctf_type_kind(fp, ctf_type_resolve(fp, id)) !=
1478                             CTF_K_FORWARD)

  68                         if (ctf_type_kind(dmp->dm_ctfp, ctf_type_resolve(
  69                             dmp->dm_ctfp, id)) != CTF_K_FORWARD)
1479                                 return (0);

1481                         found++;
1482                 }
1483         }

1485         if (found == 0)
1486                 return (dt_set_errno(dtp, EDT_NOTYPE));

1488         return (0);
1489 }

1491 int
1492 dtrace_symbol_type(dtrace_hdl_t *dtp, const GElf_Sym *symp,
1493     const dtrace_syminfo_t *sip, dtrace_typeinfo_t *tip)
1494 {
1495         dt_module_t *dmp;

1497         tip->dtt_object = NULL;
1498         tip->dtt_ctfp = NULL;
1499         tip->dtt_type = CTF_ERR;
1500         tip->dtt_flags = 0;
```



```
1501 #endif /* ! codereview */

1503         if ((dmp = dt_module_lookup_by_name(dtp, sip->dts_object)) == NULL)
1504                 return (dt_set_errno(dtp, EDT_NOMOD));

1506         if (symp->st_shndx == SHN_UNDEF && dmp->dm_extern != NULL) {
1507                 dt_ident_t *idp =
1508                     dt_idhash_lookup(dmp->dm_extern, sip->dts_name);

1510                 if (idp == NULL)
1511                         return (dt_set_errno(dtp, EDT_NOSYM));

1513                 tip->dtt_ctfp = idp->di_ctfp;
1514                 tip->dtt_type = idp->di_type;

1516         } else if (GELF_ST_TYPE(symp->st_info) != STT_FUNC) {
1517                 if (dt_module_getctf(dtp, dmp) == NULL)
1518                         return (-1); /* errno is set for us */

1520                 tip->dtt_ctfp = dmp->dm_ctfp;
1521                 tip->dtt_type = ctf_lookup_by_symbol(dmp->dm_ctfp, sip->dts_id);

1523                 if (tip->dtt_type == CTF_ERR) {
1524                         dtp->dt_ctferr = ctf_errno(tip->dtt_ctfp);
1525                         return (dt_set_errno(dtp, EDT_CTF));
1526                 }

1528         } else {
1529                 tip->dtt_ctfp = DT_FPTR_CTFP(dtp);
1530                 tip->dtt_type = DT_FPTR_TYPE(dtp);
1531         }

1533         tip->dtt_object = dmp->dm_name;
1534         return (0);
1535 }

1537 static dtrace_objinfo_t *
1538 dt_module_info(const dt_module_t *dmp, dtrace_objinfo_t *dto)
1539 {
1540         dto->dto_name = dmp->dm_name;
1541         dto->dto_file = dmp->dm_file;
1542         dto->dto_id = dmp->dm_modid;
1543         dto->dto_flags = 0;

1545         if (dmp->dm_flags & DT_DM_KERNEL)
1546                 dto->dto_flags |= DTRACE_OBJ_F_KERNEL;
1547         if (dmp->dm_flags & DT_DM_PRIMARY)
1548                 dto->dto_flags |= DTRACE_OBJ_F_PRIMARY;

1550         dto->dto_text_va = dmp->dm_text_va;
1551         dto->dto_text_size = dmp->dm_text_size;
1552         dto->dto_data_va = dmp->dm_data_va;
1553         dto->dto_data_size = dmp->dm_data_size;
1554         dto->dto_bss_va = dmp->dm_bss_va;
1555         dto->dto_bss_size = dmp->dm_bss_size;

1557         return (dto);
1558 }

1560 int
1561 dtrace_object_iter(dtrace_hdl_t *dtp, dtrace_obj_f *func, void *data)
1562 {
1563         const dt_module_t *dmp = dt_list_next(&dtp->dt_modlist);
1564         dtrace_objinfo_t dto;
1565         int rv;
```

```
1567         for (; dmp != NULL; dmp = dt_list_next(dmp)) {
1568                 if ((rv = (*func)(dtp, dt_module_info(dmp, &dto), data)) != 0)
1569                         return (rv);
1570         }

1572         return (0);
1573 }

1575 int
1576 dtrace_object_info(dtrace_hdl_t *dtp, const char *object, dtrace_objinfo_t *dto)
1577 {
1578         dt_module_t *dmp;

1580         if (object == DTRACE_OBJ_EVERY || object == DTRACE_OBJ_KMODS ||
1581             object == DTRACE_OBJ_UMODS || dto == NULL)
1582                 return (dt_set_errno(dtp, EINVAL));

1584         if ((dmp = dt_module_from_object(dtp, object)) == NULL)
1585                 return (-1); /* dt_errno is set for us */

1587         if (dt_module_load(dtp, dmp) == -1)
1588                 return (-1); /* dt_errno is set for us */

1590         (void) dt_module_info(dmp, dto);
1591         return (0);
1592 }
```

```
**********************************************************
    2049 Tue Jan 14 16:48:56 2014
new/usr/src/lib/libdtrace/common/dt_module.h
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013, Joyent, Inc.  All rights reserved.
 */
#endif /* ! codereview */

#ifndef _DT_MODULE_H
#define _DT_MODULE_H

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

#include <dt_impl.h>

#ifdef  __cplusplus
extern "C" {
#endif

extern dt_module_t *dt_module_create(dtrace_hdl_t *, const char *);
extern int dt_module_load(dtrace_hdl_t *, dt_module_t *);
extern void dt_module_unload(dtrace_hdl_t *, dt_module_t *);
extern void dt_module_destroy(dtrace_hdl_t *, dt_module_t *);

extern dt_module_t *dt_module_lookup_by_name(dtrace_hdl_t *, const char *);
extern dt_module_t *dt_module_lookup_by_ctf(dtrace_hdl_t *, ctf_file_t *);

extern int dt_module_hasctf(dtrace_hdl_t *, dt_module_t *);
#endif /* ! codereview */
extern ctf_file_t *dt_module_getctf(dtrace_hdl_t *, dt_module_t *);
extern dt_ident_t *dt_module_extern(dtrace_hdl_t *, dt_module_t *,
    const char *, const dtrace_typeinfo_t *);

extern const char *dt_module_modelname(dt_module_t *);
```



```
extern int dt_module_getlibid(dtrace_hdl_t *, dt_module_t *,
    const ctf_file_t *);
extern ctf_file_t *dt_module_getctflib(dtrace_hdl_t *, dt_module_t *,
    const char *);
#endif /* ! codereview */

#ifdef  __cplusplus
}
#endif

#endif  /* _DT_MODULE_H */
```

```
**********************************************************
   54253 Tue Jan 14 16:48:56 2014
new/usr/src/lib/libdtrace/common/dt_open.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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, Joyent, Inc. All rights reserved.
 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

#include <sys/types.h>
#include <sys/modctl.h>
#include <sys/systeminfo.h>
#include <sys/resource.h>

#include <libelf.h>
#include <strings.h>
#include <alloca.h>
#include <limits.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <assert.h>

#define _POSIX_PTHREAD_SEMANTICS
#include <dirent.h>
#undef  _POSIX_PTHREAD_SEMANTICS

#include <dt_impl.h>
#include <dt_program.h>
#include <dt_module.h>
#include <dt_printf.h>
#include <dt_string.h>
#include <dt_provider.h>

/*
```



```
 * Stability and versioning definitions.  These #defines are used in the tables
 * of identifiers below to fill in the attribute and version fields associated
 * with each identifier.  The DT_ATTR_* macros are a convenience to permit more
 * concise declarations of common attributes such as Stable/Stable/Common.  The
 * DT_VERS_* macros declare the encoded integer values of all versions used so
 * far.  DT_VERS_LATEST must correspond to the latest version value among all
 * versions exported by the D compiler.  DT_VERS_STRING must be an ASCII string
 * that contains DT_VERS_LATEST within it along with any suffixes (e.g. Beta).
 * You must update DT_VERS_LATEST and DT_VERS_STRING when adding a new version,
 * and then add the new version to the _dtrace_versions[] array declared below.
 * Refer to the Solaris Dynamic Tracing Guide Stability and Versioning chapters
 * respectively for an explanation of these DTrace features and their values.
 *
 * NOTE: Although the DTrace versioning scheme supports the labeling and
 *       introduction of incompatible changes (e.g. dropping an interface in a
 *       major release), the libdtrace code does not currently support this.
 *       All versions are assumed to strictly inherit from one another.  If
 *       we ever need to provide divergent interfaces, this will need work.
 */
#define DT_ATTR_STABCMN { DTRACE_STABILITY_STABLE, \
        DTRACE_STABILITY_STABLE, DTRACE_CLASS_COMMON }

#define DT_ATTR_EVOLCMN { DTRACE_STABILITY_EVOLVING, \
        DTRACE_STABILITY_EVOLVING, DTRACE_CLASS_COMMON \
}

/*
 * The version number should be increased for every customer visible release
 * of DTrace. The major number should be incremented when a fundamental
 * change has been made that would affect all consumers, and would reflect
 * sweeping changes to DTrace or the D language. The minor number should be
 * incremented when a change is introduced that could break scripts that had
 * previously worked; for example, adding a new built-in variable could break
 * a script which was already using that identifier. The micro number should
 * be changed when introducing functionality changes or major bug fixes that
 * do not affect backward compatibility -- this is merely to make capabilities
 * easily determined from the version number. Minor bugs do not require any
 * modification to the version number.
 */
#define DT_VERS_1_0     DT_VERSION_NUMBER(1, 0, 0)
#define DT_VERS_1_1     DT_VERSION_NUMBER(1, 1, 0)
#define DT_VERS_1_2     DT_VERSION_NUMBER(1, 2, 0)
#define DT_VERS_1_2_1   DT_VERSION_NUMBER(1, 2, 1)
#define DT_VERS_1_2_2   DT_VERSION_NUMBER(1, 2, 2)
#define DT_VERS_1_3     DT_VERSION_NUMBER(1, 3, 0)
#define DT_VERS_1_4     DT_VERSION_NUMBER(1, 4, 0)
#define DT_VERS_1_4_1   DT_VERSION_NUMBER(1, 4, 1)
#define DT_VERS_1_5     DT_VERSION_NUMBER(1, 5, 0)
#define DT_VERS_1_6     DT_VERSION_NUMBER(1, 6, 0)
#define DT_VERS_1_6_1   DT_VERSION_NUMBER(1, 6, 1)
#define DT_VERS_1_6_2   DT_VERSION_NUMBER(1, 6, 2)
#define DT_VERS_1_6_3   DT_VERSION_NUMBER(1, 6, 3)
#define DT_VERS_1_7     DT_VERSION_NUMBER(1, 7, 0)
#define DT_VERS_1_7_1   DT_VERSION_NUMBER(1, 7, 1)
#define DT_VERS_1_8     DT_VERSION_NUMBER(1, 8, 0)
#define DT_VERS_1_8_1   DT_VERSION_NUMBER(1, 8, 1)
#define DT_VERS_1_9     DT_VERSION_NUMBER(1, 9, 0)
#define DT_VERS_1_9_1   DT_VERSION_NUMBER(1, 9, 1)
#define DT_VERS_1_10    DT_VERSION_NUMBER(1, 10, 0)
#define DT_VERS_1_11    DT_VERSION_NUMBER(1, 11, 0)
#define DT_VERS_1_12    DT_VERSION_NUMBER(1, 12, 0)
#endif /* ! codereview */
#define DT_VERS_LATEST  DT_VERS_1_11
#define DT_VERS_STRING  "Sun D 1.12"
#define DT_VERS_STRING  "Sun D 1.11"
```

```
121 const dt_version_t _dtrace_versions[] = {
122         DT_VERS_1_0,    /* D API 1.0.0 (PSARC 2001/466) Solaris 10 FCS */
123         DT_VERS_1_1,    /* D API 1.1.0 Solaris Express 6/05 */
124         DT_VERS_1_2,    /* D API 1.2.0 Solaris 10 Update 1 */
125         DT_VERS_1_2_1,  /* D API 1.2.1 Solaris Express 4/06 */
126         DT_VERS_1_2_2,  /* D API 1.2.2 Solaris Express 6/06 */
127         DT_VERS_1_3,    /* D API 1.3 Solaris Express 10/06 */
128         DT_VERS_1_4,    /* D API 1.4 Solaris Express 2/07 */
129         DT_VERS_1_4_1,  /* D API 1.4.1 Solaris Express 4/07 */
130         DT_VERS_1_5,    /* D API 1.5 Solaris Express 7/07 */
131         DT_VERS_1_6,    /* D API 1.6 */
132         DT_VERS_1_6_1,  /* D API 1.6.1 */
133         DT_VERS_1_6_2,  /* D API 1.6.2 */
134         DT_VERS_1_6_3,  /* D API 1.6.3 */
135         DT_VERS_1_7,    /* D API 1.7 */
136         DT_VERS_1_7_1,  /* D API 1.7.1 */
137         DT_VERS_1_8,    /* D API 1.8 */
138         DT_VERS_1_8_1,  /* D API 1.8.1 */
139         DT_VERS_1_9,    /* D API 1.9 */
140         DT_VERS_1_9_1,  /* D API 1.9.1 */
141         DT_VERS_1_10,   /* D API 1.10 */
142         DT_VERS_1_11,   /* D API 1.11 */
143         DT_VERS_1_12,   /* D API 1.12 */
144 #endif /* ! codereview */
145         0
146 };

148 /*
149  * Table of global identifiers.  This is used to populate the global identifier
150  * hash when a new dtrace client open occurs.  For more info see dt_ident.h.
151  * The global identifiers that represent functions use the dt_idops_func ops
152  * and specify the private data pointer as a prototype string which is parsed
153  * when the identifier is first encountered.  These prototypes look like ANSI
154  * C function prototypes except that the special symbol "@" can be used as a
155  * wildcard to represent a single parameter of any type (i.e. any dt_node_t).
156  * The standard "..." notation can also be used to represent varargs.  An empty
157  * parameter list is taken to mean void (that is, no arguments are permitted).
158  * A parameter enclosed in square brackets (e.g. "[int]") denotes an optional
159  * argument.
160  */
161 static const dt_ident_t _dtrace_globals[] = {
162 { "alloca", DT_IDENT_FUNC, 0, DIF_SUBR_ALLOCA, DT_ATTR_STABCMN, DT_VERS_1_0,
163         &dt_idops_func, "void *(size_t)" },
164 { "arg0", DT_IDENT_SCALAR, 0, DIF_VAR_ARG0, DT_ATTR_STABCMN, DT_VERS_1_0,
165         &dt_idops_type, "int64_t" },
166 { "arg1", DT_IDENT_SCALAR, 0, DIF_VAR_ARG1, DT_ATTR_STABCMN, DT_VERS_1_0,
167         &dt_idops_type, "int64_t" },
168 { "arg2", DT_IDENT_SCALAR, 0, DIF_VAR_ARG2, DT_ATTR_STABCMN, DT_VERS_1_0,
169         &dt_idops_type, "int64_t" },
170 { "arg3", DT_IDENT_SCALAR, 0, DIF_VAR_ARG3, DT_ATTR_STABCMN, DT_VERS_1_0,
171         &dt_idops_type, "int64_t" },
172 { "arg4", DT_IDENT_SCALAR, 0, DIF_VAR_ARG4, DT_ATTR_STABCMN, DT_VERS_1_0,
173         &dt_idops_type, "int64_t" },
174 { "arg5", DT_IDENT_SCALAR, 0, DIF_VAR_ARG5, DT_ATTR_STABCMN, DT_VERS_1_0,
175         &dt_idops_type, "int64_t" },
176 { "arg6", DT_IDENT_SCALAR, 0, DIF_VAR_ARG6, DT_ATTR_STABCMN, DT_VERS_1_0,
177         &dt_idops_type, "int64_t" },
178 { "arg7", DT_IDENT_SCALAR, 0, DIF_VAR_ARG7, DT_ATTR_STABCMN, DT_VERS_1_0,
179         &dt_idops_type, "int64_t" },
180 { "arg8", DT_IDENT_SCALAR, 0, DIF_VAR_ARG8, DT_ATTR_STABCMN, DT_VERS_1_0,
181         &dt_idops_type, "int64_t" },
182 { "arg9", DT_IDENT_SCALAR, 0, DIF_VAR_ARG9, DT_ATTR_STABCMN, DT_VERS_1_0,
183         &dt_idops_type, "int64_t" },
184 { "args", DT_IDENT_ARRAY, 0, DIF_VAR_ARGS, DT_ATTR_STABCMN, DT_VERS_1_0,
185         &dt_idops_args, NULL },
186 { "avg", DT_IDENT_AGGFUNC, 0, DTRACEAGG_AVG, DT_ATTR_STABCMN, DT_VERS_1_0,
```



```
187         &dt_idops_func, "void(@)" },
188 { "basename", DT_IDENT_FUNC, 0, DIF_SUBR_BASENAME, DT_ATTR_STABCMN, DT_VERS_1_0,
189         &dt_idops_func, "string(const char *)" },
190 { "bcopy", DT_IDENT_FUNC, 0, DIF_SUBR_BCOPY, DT_ATTR_STABCMN, DT_VERS_1_0,
191         &dt_idops_func, "void(void *, void *, size_t)" },
192 { "breakpoint", DT_IDENT_ACTFUNC, 0, DT_ACT_BREAKPOINT,
193         DT_ATTR_STABCMN, DT_VERS_1_0,
194         &dt_idops_func, "void()" },
195 { "caller", DT_IDENT_SCALAR, 0, DIF_VAR_CALLER, DT_ATTR_STABCMN, DT_VERS_1_0,
196         &dt_idops_type, "uintptr_t" },
197 { "chill", DT_IDENT_ACTFUNC, 0, DT_ACT_CHILL, DT_ATTR_STABCMN, DT_VERS_1_0,
198         &dt_idops_func, "void(int)" },
199 { "cleanpath", DT_IDENT_FUNC, 0, DIF_SUBR_CLEANPATH, DT_ATTR_STABCMN,
200         DT_VERS_1_0, &dt_idops_func, "string(const char *)" },
201 { "clear", DT_IDENT_ACTFUNC, 0, DT_ACT_CLEAR, DT_ATTR_STABCMN, DT_VERS_1_0,
202         &dt_idops_func, "void(...)" },
203 { "commit", DT_IDENT_ACTFUNC, 0, DT_ACT_COMMIT, DT_ATTR_STABCMN, DT_VERS_1_0,
204         &dt_idops_func, "void(int)" },
205 { "copyin", DT_IDENT_FUNC, 0, DIF_SUBR_COPYIN, DT_ATTR_STABCMN, DT_VERS_1_0,
206         &dt_idops_func, "void *(uintptr_t, size_t)" },
207 { "copyinstr", DT_IDENT_FUNC, 0, DIF_SUBR_COPYINSTR,
208         DT_ATTR_STABCMN, DT_VERS_1_0,
209         &dt_idops_func, "string(uintptr_t, [size_t])" },
210 { "copyinto", DT_IDENT_FUNC, 0, DIF_SUBR_COPYINTO, DT_ATTR_STABCMN,
211         DT_VERS_1_0, &dt_idops_func, "void(uintptr_t, size_t, void *)" },
212 { "copyout", DT_IDENT_FUNC, 0, DIF_SUBR_COPYOUT, DT_ATTR_STABCMN, DT_VERS_1_0,
213         &dt_idops_func, "void(void *, uintptr_t, size_t)" },
214 { "copyoutstr", DT_IDENT_FUNC, 0, DIF_SUBR_COPYOUTSTR,
215         DT_ATTR_STABCMN, DT_VERS_1_0,
216         &dt_idops_func, "void(char *, uintptr_t, size_t)" },
217 { "count", DT_IDENT_AGGFUNC, 0, DTRACEAGG_COUNT, DT_ATTR_STABCMN, DT_VERS_1_0,
218         &dt_idops_func, "void()" },
219 { "curthread", DT_IDENT_SCALAR, 0, DIF_VAR_CURTHREAD,
220         { DTRACE_STABILITY_STABLE, DTRACE_STABILITY_PRIVATE,
221         DTRACE_CLASS_COMMON }, DT_VERS_1_0,
222         &dt_idops_type, "genunix`kthread_t *" },
223 { "ddi_pathname", DT_IDENT_FUNC, 0, DIF_SUBR_DDI_PATHNAME,
224         DT_ATTR_EVOLCMN, DT_VERS_1_0,
225         &dt_idops_func, "string(void *, int64_t)" },
226 { "denormalize", DT_IDENT_ACTFUNC, 0, DT_ACT_DENORMALIZE, DT_ATTR_STABCMN,
227         DT_VERS_1_0, &dt_idops_func, "void(...)" },
228 { "dirname", DT_IDENT_FUNC, 0, DIF_SUBR_DIRNAME, DT_ATTR_STABCMN, DT_VERS_1_0,
229         &dt_idops_func, "string(const char *)" },
230 { "discard", DT_IDENT_ACTFUNC, 0, DT_ACT_DISCARD, DT_ATTR_STABCMN, DT_VERS_1_0,
231         &dt_idops_func, "void(int)" },
232 { "epid", DT_IDENT_SCALAR, 0, DIF_VAR_EPID, DT_ATTR_STABCMN, DT_VERS_1_0,
233         &dt_idops_type, "uint_t" },
234 { "errno", DT_IDENT_SCALAR, 0, DIF_VAR_ERRNO, DT_ATTR_STABCMN, DT_VERS_1_0,
235         &dt_idops_type, "int" },
236 { "execname", DT_IDENT_SCALAR, 0, DIF_VAR_EXECNAME,
237         DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
238 { "exit", DT_IDENT_ACTFUNC, 0, DT_ACT_EXIT, DT_ATTR_STABCMN, DT_VERS_1_0,
239         &dt_idops_func, "void(int)" },
240 { "freopen", DT_IDENT_ACTFUNC, 0, DT_ACT_FREOPEN, DT_ATTR_STABCMN,
241         DT_VERS_1_1, &dt_idops_func, "void(@, ...)" },
242 { "ftruncate", DT_IDENT_ACTFUNC, 0, DT_ACT_FTRUNCATE, DT_ATTR_STABCMN,
243         DT_VERS_1_0, &dt_idops_func, "void()" },
244 { "func", DT_IDENT_ACTFUNC, 0, DT_ACT_SYM, DT_ATTR_STABCMN,
245         DT_VERS_1_2, &dt_idops_func, "_symaddr(uintptr_t)" },
246 { "getmajor", DT_IDENT_FUNC, 0, DIF_SUBR_GETMAJOR,
247         DT_ATTR_EVOLCMN, DT_VERS_1_0,
248         &dt_idops_func, "genunix`major_t(genunix`dev_t)" },
249 { "getminor", DT_IDENT_FUNC, 0, DIF_SUBR_GETMINOR,
250         DT_ATTR_EVOLCMN, DT_VERS_1_0,
251         &dt_idops_func, "genunix`minor_t(genunix`dev_t)" },
252 { "htonl", DT_IDENT_FUNC, 0, DIF_SUBR_HTONL, DT_ATTR_EVOLCMN, DT_VERS_1_3,
```

```
253         &dt_idops_func, "uint32_t(uint32_t)" },
254 { "htonll", DT_IDENT_FUNC, 0, DIF_SUBR_HTONLL, DT_ATTR_EVOLCMN, DT_VERS_1_3,
255         &dt_idops_func, "uint64_t(uint64_t)" },
256 { "htons", DT_IDENT_FUNC, 0, DIF_SUBR_HTONS, DT_ATTR_EVOLCMN, DT_VERS_1_3,
257         &dt_idops_func, "uint16_t(uint16_t)" },
258 { "getf", DT_IDENT_FUNC, 0, DIF_SUBR_GETF, DT_ATTR_STABCMN, DT_VERS_1_10,
259         &dt_idops_func, "file_t *(int)" },
260 { "gid", DT_IDENT_SCALAR, 0, DIF_VAR_GID, DT_ATTR_STABCMN, DT_VERS_1_0,
261         &dt_idops_type, "gid_t" },
262 { "id", DT_IDENT_SCALAR, 0, DIF_VAR_ID, DT_ATTR_STABCMN, DT_VERS_1_0,
263         &dt_idops_type, "uint_t" },
264 { "index", DT_IDENT_FUNC, 0, DIF_SUBR_INDEX, DT_ATTR_STABCMN, DT_VERS_1_1,
265         &dt_idops_func, "int(const char *, const char *, [int])" },
266 { "inet_ntoa", DT_IDENT_FUNC, 0, DIF_SUBR_INET_NTOA, DT_ATTR_STABCMN,
267         DT_VERS_1_5, &dt_idops_func, "string(ipaddr_t *)" },
268 { "inet_ntoa6", DT_IDENT_FUNC, 0, DIF_SUBR_INET_NTOA6, DT_ATTR_STABCMN,
269         DT_VERS_1_5, &dt_idops_func, "string(in6_addr_t *)" },
270 { "inet_ntop", DT_IDENT_FUNC, 0, DIF_SUBR_INET_NTOP, DT_ATTR_STABCMN,
271         DT_VERS_1_5, &dt_idops_func, "string(int, void *)" },
272 { "ipl", DT_IDENT_SCALAR, 0, DIF_VAR_IPL, DT_ATTR_STABCMN, DT_VERS_1_0,
273         &dt_idops_type, "uint_t" },
274 { "json", DT_IDENT_FUNC, 0, DIF_SUBR_JSON, DT_ATTR_STABCMN, DT_VERS_1_11,
275         &dt_idops_func, "string(const char *, const char *)" },
276 { "jstack", DT_IDENT_ACTFUNC, 0, DT_ACT_JSTACK, DT_ATTR_STABCMN, DT_VERS_1_0,
277         &dt_idops_func, "stack(...)" },
278 { "lltostr", DT_IDENT_FUNC, 0, DIF_SUBR_LLTOSTR, DT_ATTR_STABCMN, DT_VERS_1_0,
279         &dt_idops_func, "string(int64_t, [int])" },
280 { "llquantize", DT_IDENT_AGGFUNC, 0, DTRACEAGG_LLQUANTIZE, DT_ATTR_STABCMN,
281         DT_VERS_1_7, &dt_idops_func,
282         "void(@, int32_t, int32_t, int32_t, int32_t, ...)" },
283 { "lquantize", DT_IDENT_AGGFUNC, 0, DTRACEAGG_LQUANTIZE,
284         DT_ATTR_STABCMN, DT_VERS_1_0,
285         &dt_idops_func, "void(@, int32_t, int32_t, ...)" },
286 { "max", DT_IDENT_AGGFUNC, 0, DTRACEAGG_MAX, DT_ATTR_STABCMN, DT_VERS_1_0,
287         &dt_idops_func, "void(@)" },
288 { "min", DT_IDENT_AGGFUNC, 0, DTRACEAGG_MIN, DT_ATTR_STABCMN, DT_VERS_1_0,
289         &dt_idops_func, "void(@)" },
290 { "mod", DT_IDENT_ACTFUNC, 0, DT_ACT_MOD, DT_ATTR_STABCMN,
291         DT_VERS_1_2, &dt_idops_func, "_symaddr(uintptr_t)" },
292 { "msgdsize", DT_IDENT_FUNC, 0, DIF_SUBR_MSGDSIZE,
293         DT_ATTR_STABCMN, DT_VERS_1_0,
294         &dt_idops_func, "size_t(mblk_t *)" },
295 { "msgsize", DT_IDENT_FUNC, 0, DIF_SUBR_MSGSIZE,
296         DT_ATTR_STABCMN, DT_VERS_1_0,
297         &dt_idops_func, "size_t(mblk_t *)" },
298 { "mutex_owned", DT_IDENT_FUNC, 0, DIF_SUBR_MUTEX_OWNED,
299         DT_ATTR_EVOLCMN, DT_VERS_1_0,
300         &dt_idops_func, "int(genunix`kmutex_t *)" },
301 { "mutex_owner", DT_IDENT_FUNC, 0, DIF_SUBR_MUTEX_OWNER,
302         DT_ATTR_EVOLCMN, DT_VERS_1_0,
303         &dt_idops_func, "genunix`kthread_t *(genunix`kmutex_t *)" },
304 { "mutex_type_adaptive", DT_IDENT_FUNC, 0, DIF_SUBR_MUTEX_TYPE_ADAPTIVE,
305         DT_ATTR_EVOLCMN, DT_VERS_1_0,
306         &dt_idops_func, "int(genunix`kmutex_t *)" },
307 { "mutex_type_spin", DT_IDENT_FUNC, 0, DIF_SUBR_MUTEX_TYPE_SPIN,
308         DT_ATTR_EVOLCMN, DT_VERS_1_0,
309         &dt_idops_func, "int(genunix`kmutex_t *)" },
310 { "ntohl", DT_IDENT_FUNC, 0, DIF_SUBR_NTOHL, DT_ATTR_EVOLCMN, DT_VERS_1_3,
311         &dt_idops_func, "uint32_t(uint32_t)" },
312 { "ntohll", DT_IDENT_FUNC, 0, DIF_SUBR_NTOHLL, DT_ATTR_EVOLCMN, DT_VERS_1_3,
313         &dt_idops_func, "uint64_t(uint64_t)" },
314 { "ntohs", DT_IDENT_FUNC, 0, DIF_SUBR_NTOHS, DT_ATTR_EVOLCMN, DT_VERS_1_3,
315         &dt_idops_func, "uint16_t(uint16_t)" },
316 { "normalize", DT_IDENT_ACTFUNC, 0, DT_ACT_NORMALIZE, DT_ATTR_STABCMN,
317         DT_VERS_1_0, &dt_idops_func, "void(...)" },
318 { "panic", DT_IDENT_ACTFUNC, 0, DT_ACT_PANIC, DT_ATTR_STABCMN, DT_VERS_1_0,
```



```
319         &dt_idops_func, "void()" },
320 { "pid", DT_IDENT_SCALAR, 0, DIF_VAR_PID, DT_ATTR_STABCMN, DT_VERS_1_0,
321         &dt_idops_type, "pid_t" },
322 { "ppid", DT_IDENT_SCALAR, 0, DIF_VAR_PPID, DT_ATTR_STABCMN, DT_VERS_1_0,
323         &dt_idops_type, "pid_t" },
324 { "print", DT_IDENT_ACTFUNC, 0, DT_ACT_PRINT, DT_ATTR_STABCMN, DT_VERS_1_9,
325         &dt_idops_func, "void(@)" },
326 { "printa", DT_IDENT_ACTFUNC, 0, DT_ACT_PRINTA, DT_ATTR_STABCMN, DT_VERS_1_0,
327         &dt_idops_func, "void(@, ...)" },
328 { "printf", DT_IDENT_ACTFUNC, 0, DT_ACT_PRINTF, DT_ATTR_STABCMN, DT_VERS_1_0,
329         &dt_idops_func, "void(@, ...)" },
330 { "probefunc", DT_IDENT_SCALAR, 0, DIF_VAR_PROBEFUNC,
331         DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
332 { "probemod", DT_IDENT_SCALAR, 0, DIF_VAR_PROBEMOD,
333         DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
334 { "probename", DT_IDENT_SCALAR, 0, DIF_VAR_PROBENAME,
335         DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
336 { "probeprov", DT_IDENT_SCALAR, 0, DIF_VAR_PROBEPROV,
337         DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
338 { "progenyof", DT_IDENT_FUNC, 0, DIF_SUBR_PROGENYOF,
339         DT_ATTR_STABCMN, DT_VERS_1_0,
340         &dt_idops_func, "int(pid_t)" },
341 { "quantize", DT_IDENT_AGGFUNC, 0, DTRACEAGG_QUANTIZE,
342         DT_ATTR_STABCMN, DT_VERS_1_0,
343         &dt_idops_func, "void(@, ...)" },
344 { "raise", DT_IDENT_ACTFUNC, 0, DT_ACT_RAISE, DT_ATTR_STABCMN, DT_VERS_1_0,
345         &dt_idops_func, "void(int)" },
346 { "rand", DT_IDENT_FUNC, 0, DIF_SUBR_RAND, DT_ATTR_STABCMN, DT_VERS_1_0,
347         &dt_idops_func, "int()" },
348 { "rindex", DT_IDENT_FUNC, 0, DIF_SUBR_RINDEX, DT_ATTR_STABCMN, DT_VERS_1_1,
349         &dt_idops_func, "int(const char *, const char *, [int])" },
350 { "rw_iswriter", DT_IDENT_FUNC, 0, DIF_SUBR_RW_ISWRITER,
351         DT_ATTR_EVOLCMN, DT_VERS_1_0,
352         &dt_idops_func, "int(genunix`krwlock_t *)" },
353 { "rw_read_held", DT_IDENT_FUNC, 0, DIF_SUBR_RW_READ_HELD,
354         DT_ATTR_EVOLCMN, DT_VERS_1_0,
355         &dt_idops_func, "int(genunix`krwlock_t *)" },
356 { "rw_write_held", DT_IDENT_FUNC, 0, DIF_SUBR_RW_WRITE_HELD,
357         DT_ATTR_EVOLCMN, DT_VERS_1_0,
358         &dt_idops_func, "int(genunix`krwlock_t *)" },
359 { "self", DT_IDENT_PTR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0,
360         &dt_idops_type, "void" },
361 { "setopt", DT_IDENT_ACTFUNC, 0, DT_ACT_SETOPT, DT_ATTR_STABCMN,
362         DT_VERS_1_2, &dt_idops_func, "void(const char *, [const char *])" },
363 { "speculate", DT_IDENT_ACTFUNC, 0, DT_ACT_SPECULATE,
364         DT_ATTR_STABCMN, DT_VERS_1_0,
365         &dt_idops_func, "void(int)" },
366 { "speculation", DT_IDENT_FUNC, 0, DIF_SUBR_SPECULATION,
367         DT_ATTR_STABCMN, DT_VERS_1_0,
368         &dt_idops_func, "int()" },
369 { "stack", DT_IDENT_ACTFUNC, 0, DT_ACT_STACK, DT_ATTR_STABCMN, DT_VERS_1_0,
370         &dt_idops_func, "stack(...)" },
371 { "stackdepth", DT_IDENT_SCALAR, 0, DIF_VAR_STACKDEPTH,
372         DT_ATTR_STABCMN, DT_VERS_1_0,
373         &dt_idops_type, "uint32_t" },
374 { "stddev", DT_IDENT_AGGFUNC, 0, DTRACEAGG_STDDEV, DT_ATTR_STABCMN,
375         DT_VERS_1_6, &dt_idops_func, "void(@)" },
376 { "stop", DT_IDENT_ACTFUNC, 0, DT_ACT_STOP, DT_ATTR_STABCMN, DT_VERS_1_0,
377         &dt_idops_func, "void()" },
378 { "strchr", DT_IDENT_FUNC, 0, DIF_SUBR_STRCHR, DT_ATTR_STABCMN, DT_VERS_1_1,
379         &dt_idops_func, "string(const char *, char)" },
380 { "strlen", DT_IDENT_FUNC, 0, DIF_SUBR_STRLEN, DT_ATTR_STABCMN, DT_VERS_1_0,
381         &dt_idops_func, "size_t(const char *)" },
382 { "strjoin", DT_IDENT_FUNC, 0, DIF_SUBR_STRJOIN, DT_ATTR_STABCMN, DT_VERS_1_0,
383         &dt_idops_func, "string(const char *, const char *)" },
384 { "strrchr", DT_IDENT_FUNC, 0, DIF_SUBR_STRRCHR, DT_ATTR_STABCMN, DT_VERS_1_1,
```

```
385         &dt_idops_func, "string(const char *, char)" },
386 { "strstr", DT_IDENT_FUNC, 0, DIF_SUBR_STRSTR, DT_ATTR_STABCMN, DT_VERS_1_1,
387         &dt_idops_func, "string(const char *, const char *)" },
388 { "strtok", DT_IDENT_FUNC, 0, DIF_SUBR_STRTOK, DT_ATTR_STABCMN, DT_VERS_1_1,
389         &dt_idops_func, "string(const char *, const char *)" },
390 { "strtoll", DT_IDENT_FUNC, 0, DIF_SUBR_STRTOLL, DT_ATTR_STABCMN, DT_VERS_1_11,
391         &dt_idops_func, "int64_t(const char *, [int])" },
392 { "substr", DT_IDENT_FUNC, 0, DIF_SUBR_SUBSTR, DT_ATTR_STABCMN, DT_VERS_1_1,
393         &dt_idops_func, "string(const char *, int, [int])" },
394 { "sum", DT_IDENT_AGGFUNC, 0, DTRACEAGG_SUM, DT_ATTR_STABCMN, DT_VERS_1_0,
395         &dt_idops_func, "void(@)" },
396 { "sym", DT_IDENT_ACTFUNC, 0, DT_ACT_SYM, DT_ATTR_STABCMN,
397         DT_VERS_1_2, &dt_idops_func, "_symaddr(uintptr_t)" },
398 { "system", DT_IDENT_ACTFUNC, 0, DT_ACT_SYSTEM, DT_ATTR_STABCMN, DT_VERS_1_0,
399         &dt_idops_func, "void(@, ...)" },
400 { "this", DT_IDENT_PTR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0,
401         &dt_idops_type, "void" },
402 { "tid", DT_IDENT_SCALAR, 0, DIF_VAR_TID, DT_ATTR_STABCMN, DT_VERS_1_0,
403         &dt_idops_type, "id_t" },
404 { "timestamp", DT_IDENT_SCALAR, 0, DIF_VAR_TIMESTAMP,
405         DT_ATTR_STABCMN, DT_VERS_1_0,
406         &dt_idops_type, "uint64_t" },
407 { "tolower", DT_IDENT_FUNC, 0, DIF_SUBR_TOLOWER, DT_ATTR_STABCMN, DT_VERS_1_8,
408         &dt_idops_func, "string(const char *)" },
409 { "toupper", DT_IDENT_FUNC, 0, DIF_SUBR_TOUPPER, DT_ATTR_STABCMN, DT_VERS_1_8,
410         &dt_idops_func, "string(const char *)" },
411 { "trace", DT_IDENT_ACTFUNC, 0, DT_ACT_TRACE, DT_ATTR_STABCMN, DT_VERS_1_0,
412         &dt_idops_func, "void(@)" },
413 { "tracemem", DT_IDENT_ACTFUNC, 0, DT_ACT_TRACEMEM,
414         DT_ATTR_STABCMN, DT_VERS_1_0,
415         &dt_idops_func, "void(@, size_t, ...)" },
416 { "trunc", DT_IDENT_ACTFUNC, 0, DT_ACT_TRUNC, DT_ATTR_STABCMN,
417         DT_VERS_1_0, &dt_idops_func, "void(...)" },
418 { "uaddr", DT_IDENT_ACTFUNC, 0, DT_ACT_UADDR, DT_ATTR_STABCMN,
419         DT_VERS_1_2, &dt_idops_func, "_usymaddr(uintptr_t)" },
420 { "ucaller", DT_IDENT_SCALAR, 0, DIF_VAR_UCALLER, DT_ATTR_STABCMN,
421         DT_VERS_1_2, &dt_idops_type, "uint64_t" },
422 { "ufunc", DT_IDENT_ACTFUNC, 0, DT_ACT_USYM, DT_ATTR_STABCMN,
423         DT_VERS_1_2, &dt_idops_func, "_usymaddr(uintptr_t)" },
424 { "uid", DT_IDENT_SCALAR, 0, DIF_VAR_UID, DT_ATTR_STABCMN, DT_VERS_1_0,
425         &dt_idops_type, "uid_t" },
426 { "umod", DT_IDENT_ACTFUNC, 0, DT_ACT_UMOD, DT_ATTR_STABCMN,
427         DT_VERS_1_2, &dt_idops_func, "_usymaddr(uintptr_t)" },
428 { "uregs", DT_IDENT_ARRAY, 0, DIF_VAR_UREGS, DT_ATTR_STABCMN, DT_VERS_1_0,
429         &dt_idops_regs, NULL },
430 { "ustack", DT_IDENT_ACTFUNC, 0, DT_ACT_USTACK, DT_ATTR_STABCMN, DT_VERS_1_0,
431         &dt_idops_func, "stack(...)" },
432 { "ustackdepth", DT_IDENT_SCALAR, 0, DIF_VAR_USTACKDEPTH,
433         DT_ATTR_STABCMN, DT_VERS_1_2,
434         &dt_idops_type, "uint32_t" },
435 { "usym", DT_IDENT_ACTFUNC, 0, DT_ACT_USYM, DT_ATTR_STABCMN,
436         DT_VERS_1_2, &dt_idops_func, "_usymaddr(uintptr_t)" },
437 { "vmregs", DT_IDENT_ARRAY, 0, DIF_VAR_VMREGS, DT_ATTR_STABCMN, DT_VERS_1_7,
438         &dt_idops_regs, NULL },
439 { "vtimestamp", DT_IDENT_SCALAR, 0, DIF_VAR_VTIMESTAMP,
440         DT_ATTR_STABCMN, DT_VERS_1_0,
441         &dt_idops_type, "uint64_t" },
442 { "walltimestamp", DT_IDENT_SCALAR, 0, DIF_VAR_WALLTIMESTAMP,
443         DT_ATTR_STABCMN, DT_VERS_1_0,
444         &dt_idops_type, "int64_t" },
445 { "zonename", DT_IDENT_SCALAR, 0, DIF_VAR_ZONENAME,
446         DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
447 { NULL, 0, 0, 0, { 0, 0, 0 }, 0, NULL, NULL }
448 };

450 /*
```



```
451  * Tables of ILP32 intrinsic integer and floating-point type templates to use
452  * to populate the dynamic "C" CTF type container.
453  */
454 static const dt_intrinsic_t _dtrace_intrinsics_32[] = {
455 { "void", { CTF_INT_SIGNED, 0, 0 }, CTF_K_INTEGER },
456 { "signed", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
457 { "unsigned", { 0, 0, 32 }, CTF_K_INTEGER },
458 { "char", { CTF_INT_SIGNED | CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
459 { "short", { CTF_INT_SIGNED, 0, 16 }, CTF_K_INTEGER },
460 { "int", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
461 { "long", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
462 { "long long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
463 { "signed char", { CTF_INT_SIGNED | CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
464 { "signed short", { CTF_INT_SIGNED, 0, 16 }, CTF_K_INTEGER },
465 { "signed int", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
466 { "signed long", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
467 { "signed long long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
468 { "unsigned char", { CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
469 { "unsigned short", { 0, 0, 16 }, CTF_K_INTEGER },
470 { "unsigned int", { 0, 0, 32 }, CTF_K_INTEGER },
471 { "unsigned long", { 0, 0, 32 }, CTF_K_INTEGER },
472 { "unsigned long long", { 0, 0, 64 }, CTF_K_INTEGER },
473 { "_Bool", { CTF_INT_BOOL, 0, 8 }, CTF_K_INTEGER },
474 { "float", { CTF_FP_SINGLE, 0, 32 }, CTF_K_FLOAT },
475 { "double", { CTF_FP_DOUBLE, 0, 64 }, CTF_K_FLOAT },
476 { "long double", { CTF_FP_LDOUBLE, 0, 128 }, CTF_K_FLOAT },
477 { "float imaginary", { CTF_FP_IMAGRY, 0, 32 }, CTF_K_FLOAT },
478 { "double imaginary", { CTF_FP_DIMAGRY, 0, 64 }, CTF_K_FLOAT },
479 { "long double imaginary", { CTF_FP_LDIMAGRY, 0, 128 }, CTF_K_FLOAT },
480 { "float complex", { CTF_FP_CPLX, 0, 64 }, CTF_K_FLOAT },
481 { "double complex", { CTF_FP_DCPLX, 0, 128 }, CTF_K_FLOAT },
482 { "long double complex", { CTF_FP_LDCPLX, 0, 256 }, CTF_K_FLOAT },
483 { NULL, { 0, 0, 0 }, 0 }
484 };

486 /*
487  * Tables of LP64 intrinsic integer and floating-point type templates to use
488  * to populate the dynamic "C" CTF type container.
489  */
490 static const dt_intrinsic_t _dtrace_intrinsics_64[] = {
491 { "void", { CTF_INT_SIGNED, 0, 0 }, CTF_K_INTEGER },
492 { "signed", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
493 { "unsigned", { 0, 0, 32 }, CTF_K_INTEGER },
494 { "char", { CTF_INT_SIGNED | CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
495 { "short", { CTF_INT_SIGNED, 0, 16 }, CTF_K_INTEGER },
496 { "int", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
497 { "long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
498 { "long long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
499 { "signed char", { CTF_INT_SIGNED | CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
500 { "signed short", { CTF_INT_SIGNED, 0, 16 }, CTF_K_INTEGER },
501 { "signed int", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
502 { "signed long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
503 { "signed long long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
504 { "unsigned char", { CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
505 { "unsigned short", { 0, 0, 16 }, CTF_K_INTEGER },
506 { "unsigned int", { 0, 0, 32 }, CTF_K_INTEGER },
507 { "unsigned long", { 0, 0, 64 }, CTF_K_INTEGER },
508 { "unsigned long long", { 0, 0, 64 }, CTF_K_INTEGER },
509 { "_Bool", { CTF_INT_BOOL, 0, 8 }, CTF_K_INTEGER },
510 { "float", { CTF_FP_SINGLE, 0, 32 }, CTF_K_FLOAT },
511 { "double", { CTF_FP_DOUBLE, 0, 64 }, CTF_K_FLOAT },
512 { "long double", { CTF_FP_LDOUBLE, 0, 128 }, CTF_K_FLOAT },
513 { "float imaginary", { CTF_FP_IMAGRY, 0, 32 }, CTF_K_FLOAT },
514 { "double imaginary", { CTF_FP_DIMAGRY, 0, 64 }, CTF_K_FLOAT },
515 { "long double imaginary", { CTF_FP_LDIMAGRY, 0, 128 }, CTF_K_FLOAT },
516 { "float complex", { CTF_FP_CPLX, 0, 64 }, CTF_K_FLOAT },
```

```
{ "double complex", { CTF_FP_DCPLX, 0, 128 }, CTF_K_FLOAT },
{ "long double complex", { CTF_FP_LDCPLX, 0, 256 }, CTF_K_FLOAT },
{ NULL, { 0, 0, 0 }, 0 }
};

/*
 * Tables of ILP32 typedefs to use to populate the dynamic "D" CTF container.
 * These aliases ensure that D definitions can use typical <sys/types.h> names.
 */
static const dt_typedef_t _dtrace_typedefs_32[] = {
{ "char", "int8_t" },
{ "short", "int16_t" },
{ "int", "int32_t" },
{ "long long", "int64_t" },
{ "int", "intptr_t" },
{ "int", "ssize_t" },
{ "unsigned char", "uint8_t" },
{ "unsigned short", "uint16_t" },
{ "unsigned", "uint32_t" },
{ "unsigned long long", "uint64_t" },
{ "unsigned char", "uchar_t" },
{ "unsigned short", "ushort_t" },
{ "unsigned", "uint_t" },
{ "unsigned long", "ulong_t" },
{ "unsigned long long", "u_longlong_t" },
{ "int", "ptrdiff_t" },
{ "unsigned", "uintptr_t" },
{ "unsigned", "size_t" },
{ "long", "id_t" },
{ "long", "pid_t" },
{ NULL, NULL }
};

/*
 * Tables of LP64 typedefs to use to populate the dynamic "D" CTF container.
 * These aliases ensure that D definitions can use typical <sys/types.h> names.
 */
static const dt_typedef_t _dtrace_typedefs_64[] = {
{ "char", "int8_t" },
{ "short", "int16_t" },
{ "int", "int32_t" },
{ "long", "int64_t" },
{ "long", "intptr_t" },
{ "long", "ssize_t" },
{ "unsigned char", "uint8_t" },
{ "unsigned short", "uint16_t" },
{ "unsigned", "uint32_t" },
{ "unsigned long", "uint64_t" },
{ "unsigned char", "uchar_t" },
{ "unsigned short", "ushort_t" },
{ "unsigned", "uint_t" },
{ "unsigned long", "ulong_t" },
{ "unsigned long long", "u_longlong_t" },
{ "long", "ptrdiff_t" },
{ "unsigned long", "uintptr_t" },
{ "unsigned long", "size_t" },
{ "int", "id_t" },
{ "int", "pid_t" },
{ NULL, NULL }
};

/*
 * Tables of ILP32 integer type templates used to populate the dtp->dt_ints[]
 * cache when a new dtrace client open occurs.  Values are set by dtrace_open().
 */
static const dt_intdesc_t _dtrace_ints_32[] = {
```



```
{ "int", NULL, CTF_ERR, 0x7fffffffULL },
{ "unsigned int", NULL, CTF_ERR, 0xffffffffULL },
{ "long", NULL, CTF_ERR, 0x7fffffffULL },
{ "unsigned long", NULL, CTF_ERR, 0xffffffffULL },
{ "long long", NULL, CTF_ERR, 0x7fffffffffffffffULL },
{ "unsigned long long", NULL, CTF_ERR, 0xffffffffffffffffULL }
};

/*
 * Tables of LP64 integer type templates used to populate the dtp->dt_ints[]
 * cache when a new dtrace client open occurs.  Values are set by dtrace_open().
 */
static const dt_intdesc_t _dtrace_ints_64[] = {
{ "int", NULL, CTF_ERR, 0x7fffffffULL },
{ "unsigned int", NULL, CTF_ERR, 0xffffffffULL },
{ "long", NULL, CTF_ERR, 0x7fffffffffffffffULL },
{ "unsigned long", NULL, CTF_ERR, 0xffffffffffffffffULL },
{ "long long", NULL, CTF_ERR, 0x7fffffffffffffffULL },
{ "unsigned long long", NULL, CTF_ERR, 0xffffffffffffffffULL }
};

/*
 * Table of macro variable templates used to populate the macro identifier hash
 * when a new dtrace client open occurs.  Values are set by dtrace_update().
 */
static const dt_ident_t _dtrace_macros[] = {
{ "egid", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ "euid", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ "gid", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ "pid", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ "pgid", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ "ppid", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ "projid", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ "sid", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ "taskid", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ "target", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ "uid", DT_IDENT_SCALAR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0 },
{ NULL, 0, 0, 0, { 0, 0, 0 }, 0 }
};

/*
 * Hard-wired definition string to be compiled and cached every time a new
 * DTrace library handle is initialized.  This string should only be used to
 * contain definitions that should be present regardless of DTRACE_O_NOLIBS.
 */
static const char _dtrace_hardwire[] = "\
inline long NULL = 0; \n\
#pragma D binding \"1.0\" NULL\n\
";

/*
 * Default DTrace configuration to use when opening libdtrace DTRACE_O_NODEV.
 * If DTRACE_O_NODEV is not set, we load the configuration from the kernel.
 * The use of CTF_MODEL_NATIVE is more subtle than it might appear: we are
 * relying on the fact that when running dtrace(1M), isaexec will invoke the
 * binary with the same bitness as the kernel, which is what we want by default
 * when generating our DIF.  The user can override the choice using oflags.
 */
static const dtrace_conf_t _dtrace_conf = {
        DIF_VERSION,            /* dtc_difversion */
        DIF_DIR_NREGS,          /* dtc_difintregs */
        DIF_DTR_NREGS,          /* dtc_diftupregs */
        CTF_MODEL_NATIVE        /* dtc_ctfmodel */
};

const dtrace_attribute_t _dtrace_maxattr = {
```

```
649         DTRACE_STABILITY_MAX,
650         DTRACE_STABILITY_MAX,
651         DTRACE_CLASS_MAX
652 };

654 const dtrace_attribute_t _dtrace_defattr = {
655         DTRACE_STABILITY_STABLE,
656         DTRACE_STABILITY_STABLE,
657         DTRACE_CLASS_COMMON
658 };

660 const dtrace_attribute_t _dtrace_symattr = {
661         DTRACE_STABILITY_PRIVATE,
662         DTRACE_STABILITY_PRIVATE,
663         DTRACE_CLASS_UNKNOWN
664 };

666 const dtrace_attribute_t _dtrace_typattr = {
667         DTRACE_STABILITY_PRIVATE,
668         DTRACE_STABILITY_PRIVATE,
669         DTRACE_CLASS_UNKNOWN
670 };

672 const dtrace_attribute_t _dtrace_prvattr = {
673         DTRACE_STABILITY_PRIVATE,
674         DTRACE_STABILITY_PRIVATE,
675         DTRACE_CLASS_UNKNOWN
676 };

678 const dtrace_pattr_t _dtrace_prvdesc = {
679 { DTRACE_STABILITY_UNSTABLE, DTRACE_STABILITY_UNSTABLE, DTRACE_CLASS_COMMON },
680 { DTRACE_STABILITY_UNSTABLE, DTRACE_STABILITY_UNSTABLE, DTRACE_CLASS_COMMON },
681 { DTRACE_STABILITY_UNSTABLE, DTRACE_STABILITY_UNSTABLE, DTRACE_CLASS_COMMON },
682 { DTRACE_STABILITY_UNSTABLE, DTRACE_STABILITY_UNSTABLE, DTRACE_CLASS_COMMON },
683 { DTRACE_STABILITY_UNSTABLE, DTRACE_STABILITY_UNSTABLE, DTRACE_CLASS_COMMON },
684 };

686 const char *_dtrace_defcpp = "/usr/ccs/lib/cpp"; /* default cpp(1) to invoke */
687 const char *_dtrace_defld = "/usr/ccs/bin/ld";   /* default ld(1) to invoke */

689 const char *_dtrace_libdir = "/usr/lib/dtrace"; /* default library directory */
690 const char *_dtrace_provdir = "/dev/dtrace/provider"; /* provider directory */

692 int _dtrace_strbuckets = 211;   /* default number of hash buckets (prime) */
693 int _dtrace_intbuckets = 256;   /* default number of integer buckets (Pof2) */
694 uint_t _dtrace_strsize = 256;   /* default size of string intrinsic type */
695 uint_t _dtrace_stkindent = 14;  /* default whitespace indent for stack/ustack */
696 uint_t _dtrace_pidbuckets = 64; /* default number of pid hash buckets */
697 uint_t _dtrace_pidlrulim = 8;   /* default number of pid handles to cache */
698 size_t _dtrace_bufsize = 512;   /* default dt_buf_create() size */
699 int _dtrace_argmax = 32;        /* default maximum number of probe arguments */

701 int _dtrace_debug = 0;          /* debug messages enabled (off) */
702 const char *const _dtrace_version = DT_VERS_STRING; /* API version string */
703 int _dtrace_rdvers = RD_VERSION; /* rtld_db feature version */

705 typedef struct dt_fdlist {
706         int *df_fds;            /* array of provider driver file descriptors */
707         uint_t df_ents;         /* number of valid elements in df_fds[] */
708         uint_t df_size;         /* size of df_fds[] */
709 } dt_fdlist_t;

711 #pragma init(_dtrace_init)
712 void
713 _dtrace_init(void)
714 {
```



```
715         _dtrace_debug = getenv("DTRACE_DEBUG") != NULL;

717         for (; _dtrace_rdvers > 0; _dtrace_rdvers--) {
718                 if (rd_init(_dtrace_rdvers) == RD_OK)
719                         break;
720         }
721 }

723 static dtrace_hdl_t *
724 set_open_errno(dtrace_hdl_t *dtp, int *errp, int err)
725 {
726         if (dtp != NULL)
727                 dtrace_close(dtp);
728         if (errp != NULL)
729                 *errp = err;
730         return (NULL);
731 }

733 static void
734 dt_provmod_open(dt_provmod_t **provmod, dt_fdlist_t *dfp)
735 {
736         dt_provmod_t *prov;
737         char path[PATH_MAX];
738         struct dirent *dp, *ep;
739         DIR *dirp;
740         int fd;

742         if ((dirp = opendir(_dtrace_provdir)) == NULL)
743                 return; /* failed to open directory; just skip it */

745         ep = alloca(sizeof (struct dirent) + PATH_MAX + 1);
746         bzero(ep, sizeof (struct dirent) + PATH_MAX + 1);

748         while (readdir_r(dirp, ep, &dp) == 0 && dp != NULL) {
749                 if (dp->d_name[0] == '.')
750                         continue; /* skip "." and ".." */

752                 if (dfp->df_ents == dfp->df_size) {
753                         uint_t size = dfp->df_size ? dfp->df_size * 2 : 16;
754                         int *fds = realloc(dfp->df_fds, size * sizeof (int));

756                         if (fds == NULL)
757                                 break; /* skip the rest of this directory */

759                         dfp->df_fds = fds;
760                         dfp->df_size = size;
761                 }

763                 (void) snprintf(path, sizeof (path), "%s/%s",
764                     _dtrace_provdir, dp->d_name);

766                 if ((fd = open(path, O_RDONLY)) == -1)
767                         continue; /* failed to open driver; just skip it */

769                 if (((prov = malloc(sizeof (dt_provmod_t))) == NULL) ||
770                     (prov->dp_name = malloc(strlen(dp->d_name) + 1)) == NULL) {
771                         free(prov);
772                         (void) close(fd);
773                         break;
774                 }

776                 (void) strcpy(prov->dp_name, dp->d_name);
777                 prov->dp_next = *provmod;
778                 *provmod = prov;

780                 dt_dprintf("opened provider %s\n", dp->d_name);
```

```
781                 dfp->df_fds[dfp->df_ents++] = fd;
782         }
783
784         (void) closedir(dirp);
785 }
786
787 static void
788 dt_provmod_destroy(dt_provmod_t **provmod)
789 {
790         dt_provmod_t *next, *current;
791
792         for (current = *provmod; current != NULL; current = next) {
793                 next = current->dp_next;
794                 free(current->dp_name);
795                 free(current);
796         }
797
798         *provmod = NULL;
799 }
800
801 static const char *
802 dt_get_sysinfo(int cmd, char *buf, size_t len)
803 {
804         ssize_t rv = sysinfo(cmd, buf, len);
805         char *p = buf;
806
807         if (rv < 0 || rv > len)
808                 (void) snprintf(buf, len, "%s", "Unknown");
809
810         while ((p = strchr(p, '.')) != NULL)
811                 *p++ = '_';
812
813         return (buf);
814 }
815
816 static dtrace_hdl_t *
817 dt_vopen(int version, int flags, int *errp,
818     const dtrace_vector_t *vector, void *arg)
819 {
820         dtrace_hdl_t *dtp = NULL;
821         int dtfd = -1, ftfd = -1, fterr = 0;
822         dtrace_prog_t *pgp;
823         dt_module_t *dmp;
824         dt_provmod_t *provmod = NULL;
825         int i, err;
826         struct rlimit rl;
827
828         const dt_intrinsic_t *dinp;
829         const dt_typedef_t *dtyp;
830         const dt_ident_t *idp;
831
832         dtrace_typeinfo_t dtt;
833         ctf_funcinfo_t ctc;
834         ctf_arinfo_t ctr;
835
836         dt_fdlist_t df = { NULL, 0, 0 };
837
838         char isadef[32], utsdef[32];
839         char s1[64], s2[64];
840
841         if (version <= 0)
842                 return (set_open_errno(dtp, errp, EINVAL));
843
844         if (version > DTRACE_VERSION)
845                 return (set_open_errno(dtp, errp, EDT_VERSION));
```



```
847         if (version < DTRACE_VERSION) {
848                 /*
849                  * Currently, increasing the library version number is used to
850                  * denote a binary incompatible change.  That is, a consumer
851                  * of the library cannot run on a version of the library with
852                  * a higher DTRACE_VERSION number than the consumer compiled
853                  * against.  Once the library API has been committed to,
854                  * backwards binary compatibility will be required; at that
855                  * time, this check should change to return EDT_OVERSION only
856                  * if the specified version number is less than the version
857                  * number at the time of interface commitment.
858                  */
859                 return (set_open_errno(dtp, errp, EDT_OVERSION));
860         }
861
862         if (flags & ~DTRACE_O_MASK)
863                 return (set_open_errno(dtp, errp, EINVAL));
864
865         if ((flags & DTRACE_O_LP64) && (flags & DTRACE_O_ILP32))
866                 return (set_open_errno(dtp, errp, EINVAL));
867
868         if (vector == NULL && arg != NULL)
869                 return (set_open_errno(dtp, errp, EINVAL));
870
871         if (elf_version(EV_CURRENT) == EV_NONE)
872                 return (set_open_errno(dtp, errp, EDT_ELFVERSION));
873
874         if (vector != NULL || (flags & DTRACE_O_NODEV))
875                 goto alloc; /* do not attempt to open dtrace device */
876
877         /*
878          * Before we get going, crank our limit on file descriptors up to the
879          * hard limit.  This is to allow for the fact that libproc keeps file
880          * descriptors to objects open for the lifetime of the proc handle;
881          * without raising our hard limit, we would have an acceptably small
882          * bound on the number of processes that we could concurrently
883          * instrument with the pid provider.
884          */
885         if (getrlimit(RLIMIT_NOFILE, &rl) == 0) {
886                 rl.rlim_cur = rl.rlim_max;
887                 (void) setrlimit(RLIMIT_NOFILE, &rl);
888         }
889
890         /*
891          * Get the device path of each of the providers.  We hold them open
892          * in the df.df_fds list until we open the DTrace driver itself,
893          * allowing us to see all of the probes provided on this system.  Once
894          * we have the DTrace driver open, we can safely close all the providers
895          * now that they have registered with the framework.
896          */
897         dt_provmod_open(&provmod, &df);
898
899         dtfd = open("/dev/dtrace/dtrace", O_RDWR);
900         err = errno; /* save errno from opening dtfd */
901
902         ftfd = open("/dev/dtrace/provider/fasttrap", O_RDWR);
903         fterr = ftfd == -1 ? errno : 0; /* save errno from open ftfd */
904
905         while (df.df_ents-- != 0)
906                 (void) close(df.df_fds[df.df_ents]);
907
908         free(df.df_fds);
909
910         /*
911          * If we failed to open the dtrace device, fail dtrace_open().
912          * We convert some kernel errnos to custom libdtrace errnos to
```

```
913          * improve the resulting message from the usual strerror().
914          */
915         if (dtfd == -1) {
916                 dt_provmod_destroy(&provmod);
917                 switch (err) {
918                 case ENOENT:
919                         err = EDT_NOENT;
920                         break;
921                 case EBUSY:
922                         err = EDT_BUSY;
923                         break;
924                 case EACCES:
925                         err = EDT_ACCESS;
926                         break;
927                 }
928                 return (set_open_errno(dtp, errp, err));
929         }

931         (void) fcntl(dtfd, F_SETFD, FD_CLOEXEC);
932         (void) fcntl(ftfd, F_SETFD, FD_CLOEXEC);

934 alloc:
935         if ((dtp = malloc(sizeof (dtrace_hdl_t))) == NULL)
936                 return (set_open_errno(dtp, errp, EDT_NOMEM));

938         bzero(dtp, sizeof (dtrace_hdl_t));
939         dtp->dt_oflags = flags;
940         dtp->dt_prcmode = DT_PROC_STOP_PREINIT;
941         dtp->dt_linkmode = DT_LINK_KERNEL;
942         dtp->dt_linktype = DT_LTYP_ELF;
943         dtp->dt_xlatemode = DT_XL_STATIC;
944         dtp->dt_stdcmode = DT_STDC_XA;
945         dtp->dt_version = version;
946         dtp->dt_fd = dtfd;
947         dtp->dt_ftfd = ftfd;
948         dtp->dt_fterr = fterr;
949         dtp->dt_cdefs_fd = -1;
950         dtp->dt_ddefs_fd = -1;
951         dtp->dt_stdout_fd = -1;
952         dtp->dt_modbuckets = _dtrace_strbuckets;
953         dtp->dt_mods = calloc(dtp->dt_modbuckets, sizeof (dt_module_t *));
954         dtp->dt_provbuckets = _dtrace_strbuckets;
955         dtp->dt_provs = calloc(dtp->dt_provbuckets, sizeof (dt_provider_t *));
956         dt_proc_init(dtp);
957         dtp->dt_vmax = DT_VERS_LATEST;
958         dtp->dt_cpp_path = strdup(_dtrace_defcpp);
959         dtp->dt_cpp_argv = malloc(sizeof (char *));
960         dtp->dt_cpp_argc = 1;
961         dtp->dt_cpp_args = 1;
962         dtp->dt_ld_path = strdup(_dtrace_defld);
963         dtp->dt_provmod = provmod;
964         dtp->dt_vector = vector;
965         dtp->dt_varg = arg;
966         dt_dof_init(dtp);
967         (void) uname(&dtp->dt_uts);

969         if (dtp->dt_mods == NULL || dtp->dt_provs == NULL ||
970             dtp->dt_procs == NULL || dtp->dt_proc_env == NULL ||
971             dtp->dt_ld_path == NULL || dtp->dt_cpp_path == NULL ||
972             dtp->dt_cpp_argv == NULL)
973                 return (set_open_errno(dtp, errp, EDT_NOMEM));

975         for (i = 0; i < DTRACEOPT_MAX; i++)
976                 dtp->dt_options[i] = DTRACEOPT_UNSET;

978         dtp->dt_cpp_argv[0] = (char *)strbasename(dtp->dt_cpp_path);
```



```
980         (void) snprintf(isadef, sizeof (isadef), "-D__SUNW_D_%u",
981             (uint_t)(sizeof (void *) * NBBY));

983         (void) snprintf(utsdef, sizeof (utsdef), "-D__%s_%s",
984             dt_get_sysinfo(SI_SYSNAME, s1, sizeof (s1)),
985             dt_get_sysinfo(SI_RELEASE, s2, sizeof (s2)));

987         if (dt_cpp_add_arg(dtp, "-D__sun") == NULL ||
988             dt_cpp_add_arg(dtp, "-D__unix") == NULL ||
989             dt_cpp_add_arg(dtp, "-D__SVR4") == NULL ||
990             dt_cpp_add_arg(dtp, "-D__SUNW_D=1") == NULL ||
991             dt_cpp_add_arg(dtp, isadef) == NULL ||
992             dt_cpp_add_arg(dtp, utsdef) == NULL)
993                 return (set_open_errno(dtp, errp, EDT_NOMEM));

995         if (flags & DTRACE_O_NODEV)
996                 bcopy(&_dtrace_conf, &dtp->dt_conf, sizeof (_dtrace_conf));
997         else if (dt_ioctl(dtp, DTRACEIOC_CONF, &dtp->dt_conf) != 0)
998                 return (set_open_errno(dtp, errp, errno));

1000        if (flags & DTRACE_O_LP64)
1001                dtp->dt_conf.dtc_ctfmodel = CTF_MODEL_LP64;
1002        else if (flags & DTRACE_O_ILP32)
1003                dtp->dt_conf.dtc_ctfmodel = CTF_MODEL_ILP32;

1005 #ifdef __sparc
1006        /*
1007         * On SPARC systems, __sparc is always defined for <sys/isa_defs.h>
1008         * and __sparcv9 is defined if we are doing a 64-bit compile.
1009         */
1010        if (dt_cpp_add_arg(dtp, "-D__sparc") == NULL)
1011                return (set_open_errno(dtp, errp, EDT_NOMEM));

1013        if (dtp->dt_conf.dtc_ctfmodel == CTF_MODEL_LP64 &&
1014            dt_cpp_add_arg(dtp, "-D__sparcv9") == NULL)
1015                return (set_open_errno(dtp, errp, EDT_NOMEM));
1016 #endif

1018 #ifdef __x86
1019        /*
1020         * On x86 systems, __i386 is defined for <sys/isa_defs.h> for 32-bit
1021         * compiles and __amd64 is defined for 64-bit compiles.  Unlike SPARC,
1022         * they are defined exclusive of one another (see PSARC 2004/619).
1023         */
1024        if (dtp->dt_conf.dtc_ctfmodel == CTF_MODEL_LP64) {
1025                if (dt_cpp_add_arg(dtp, "-D__amd64") == NULL)
1026                        return (set_open_errno(dtp, errp, EDT_NOMEM));
1027        } else {
1028                if (dt_cpp_add_arg(dtp, "-D__i386") == NULL)
1029                        return (set_open_errno(dtp, errp, EDT_NOMEM));
1030        }
1031 #endif

1033        if (dtp->dt_conf.dtc_difversion < DIF_VERSION)
1034                return (set_open_errno(dtp, errp, EDT_DIFVERS));

1036        if (dtp->dt_conf.dtc_ctfmodel == CTF_MODEL_ILP32)
1037                bcopy(_dtrace_ints_32, dtp->dt_ints, sizeof (_dtrace_ints_32));
1038        else
1039                bcopy(_dtrace_ints_64, dtp->dt_ints, sizeof (_dtrace_ints_64));

1041        dtp->dt_macros = dt_idhash_create("macro", NULL, 0, UINT_MAX);
1042        dtp->dt_aggs = dt_idhash_create("aggregation", NULL,
1043            DTRACE_AGGVARIDNONE + 1, UINT_MAX);
```

```
1045         dtp->dt_globals = dt_idhash_create("global", _dtrace_globals,
1046             DIF_VAR_OTHER_UBASE, DIF_VAR_OTHER_MAX);
1047
1048         dtp->dt_tls = dt_idhash_create("thread local", NULL,
1049             DIF_VAR_OTHER_UBASE, DIF_VAR_OTHER_MAX);
1050
1051         if (dtp->dt_macros == NULL || dtp->dt_aggs == NULL ||
1052             dtp->dt_globals == NULL || dtp->dt_tls == NULL)
1053                 return (set_open_errno(dtp, errp, EDT_NOMEM));
1054
1055         /*
1056          * Populate the dt_macros identifier hash table by hand: we can't use
1057          * the dt_idhash_populate() mechanism because we're not yet compiling
1058          * and dtrace_update() needs to immediately reference these idents.
1059          */
1060         for (idp = _dtrace_macros; idp->di_name != NULL; idp++) {
1061                 if (dt_idhash_insert(dtp->dt_macros, idp->di_name,
1062                     idp->di_kind, idp->di_flags, idp->di_id, idp->di_attr,
1063                     idp->di_vers, idp->di_ops ? idp->di_ops : &dt_idops_thaw,
1064                     idp->di_iarg, 0) == NULL)
1065                         return (set_open_errno(dtp, errp, EDT_NOMEM));
1066         }
1067
1068         /*
1069          * Update the module list using /system/object and load the values for
1070          * the macro variable definitions according to the current process.
1071          */
1072         dtrace_update(dtp);
1073
1074         /*
1075          * Select the intrinsics and typedefs we want based on the data model.
1076          * The intrinsics are under "C".  The typedefs are added under "D".
1077          */
1078         if (dtp->dt_conf.dtc_ctfmodel == CTF_MODEL_ILP32) {
1079                 dinp = _dtrace_intrinsics_32;
1080                 dtyp = _dtrace_typedefs_32;
1081         } else {
1082                 dinp = _dtrace_intrinsics_64;
1083                 dtyp = _dtrace_typedefs_64;
1084         }
1085
1086         /*
1087          * Create a dynamic CTF container under the "C" scope for intrinsic
1088          * types and types defined in ANSI-C header files that are included.
1089          */
1090         if ((dmp = dtp->dt_cdefs = dt_module_create(dtp, "C")) == NULL)
1091                 return (set_open_errno(dtp, errp, EDT_NOMEM));
1092
1093         if ((dmp->dm_ctfp = ctf_create(&dtp->dt_ctferr)) == NULL)
1094                 return (set_open_errno(dtp, errp, EDT_CTF));
1095
1096         dt_dprintf("created CTF container for %s (%p)\n",
1097             dmp->dm_name, (void *)dmp->dm_ctfp);
1098
1099         (void) ctf_setmodel(dmp->dm_ctfp, dtp->dt_conf.dtc_ctfmodel);
1100         ctf_setspecific(dmp->dm_ctfp, dmp);
1101
1102         dmp->dm_flags = DT_DM_LOADED; /* fake up loaded bit */
1103         dmp->dm_modid = -1; /* no module ID */
1104
1105         /*
1106          * Fill the dynamic "C" CTF container with all of the intrinsic
1107          * integer and floating-point types appropriate for this data model.
1108          */
1109         for (; dinp->din_name != NULL; dinp++) {
1110                 if (dinp->din_kind == CTF_K_INTEGER) {
```



```
1111                         err = ctf_add_integer(dmp->dm_ctfp, CTF_ADD_ROOT,
1112                             dinp->din_name, &dinp->din_data);
1113                 } else {
1114                         err = ctf_add_float(dmp->dm_ctfp, CTF_ADD_ROOT,
1115                             dinp->din_name, &dinp->din_data);
1116                 }
1117
1118                 if (err == CTF_ERR) {
1119                         dt_dprintf("failed to add %s to C container: %s\n",
1120                             dinp->din_name, ctf_errmsg(
1121                             ctf_errno(dmp->dm_ctfp)));
1122                         return (set_open_errno(dtp, errp, EDT_CTF));
1123                 }
1124         }
1125
1126         if (ctf_update(dmp->dm_ctfp) != 0) {
1127                 dt_dprintf("failed to update C container: %s\n",
1128                     ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1129                 return (set_open_errno(dtp, errp, EDT_CTF));
1130         }
1131
1132         /*
1133          * Add intrinsic pointer types that are needed to initialize printf
1134          * format dictionary types (see table in dt_printf.c).
1135          */
1136         (void) ctf_add_pointer(dmp->dm_ctfp, CTF_ADD_ROOT,
1137             ctf_lookup_by_name(dmp->dm_ctfp, "void"));
1138
1139         (void) ctf_add_pointer(dmp->dm_ctfp, CTF_ADD_ROOT,
1140             ctf_lookup_by_name(dmp->dm_ctfp, "char"));
1141
1142         (void) ctf_add_pointer(dmp->dm_ctfp, CTF_ADD_ROOT,
1143             ctf_lookup_by_name(dmp->dm_ctfp, "int"));
1144
1145         if (ctf_update(dmp->dm_ctfp) != 0) {
1146                 dt_dprintf("failed to update C container: %s\n",
1147                     ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1148                 return (set_open_errno(dtp, errp, EDT_CTF));
1149         }
1150
1151         /*
1152          * Create a dynamic CTF container under the "D" scope for types that
1153          * are defined by the D program itself or on-the-fly by the D compiler.
1154          * The "D" CTF container is a child of the "C" CTF container.
1155          */
1156         if ((dmp = dtp->dt_ddefs = dt_module_create(dtp, "D")) == NULL)
1157                 return (set_open_errno(dtp, errp, EDT_NOMEM));
1158
1159         if ((dmp->dm_ctfp = ctf_create(&dtp->dt_ctferr)) == NULL)
1160                 return (set_open_errno(dtp, errp, EDT_CTF));
1161
1162         dt_dprintf("created CTF container for %s (%p)\n",
1163             dmp->dm_name, (void *)dmp->dm_ctfp);
1164
1165         (void) ctf_setmodel(dmp->dm_ctfp, dtp->dt_conf.dtc_ctfmodel);
1166         ctf_setspecific(dmp->dm_ctfp, dmp);
1167
1168         dmp->dm_flags = DT_DM_LOADED; /* fake up loaded bit */
1169         dmp->dm_modid = -1; /* no module ID */
1170
1171         if (ctf_import(dmp->dm_ctfp, dtp->dt_cdefs->dm_ctfp) == CTF_ERR) {
1172                 dt_dprintf("failed to import D parent container: %s\n",
1173                     ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1174                 return (set_open_errno(dtp, errp, EDT_CTF));
1175         }
```

```
1177         /*
1178          * Fill the dynamic "D" CTF container with all of the built-in typedefs
1179          * that we need to use for our D variable and function definitions.
1180          * This ensures that basic inttypes.h names are always available to us.
1181          */
1182         for (; dtyp->dty_src != NULL; dtyp++) {
1183                 if (ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1184                     dtyp->dty_dst, ctf_lookup_by_name(dmp->dm_ctfp,
1185                     dtyp->dty_src)) == CTF_ERR) {
1186                         dt_dprintf("failed to add typedef %s %s to D "
1187                             "container: %s", dtyp->dty_src, dtyp->dty_dst,
1188                             ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1189                         return (set_open_errno(dtp, errp, EDT_CTF));
1190                 }
1191         }

1193         /*
1194          * Insert a CTF ID corresponding to a pointer to a type of kind
1195          * CTF_K_FUNCTION we can use in the compiler for function pointers.
1196          * CTF treats all function pointers as "int (*)()" so we only need one.
1197          */
1198         ctc.ctc_return = ctf_lookup_by_name(dmp->dm_ctfp, "int");
1199         ctc.ctc_argc = 0;
1200         ctc.ctc_flags = 0;

1202         dtp->dt_type_func = ctf_add_function(dmp->dm_ctfp,
1203             CTF_ADD_ROOT, &ctc, NULL);

1205         dtp->dt_type_fptr = ctf_add_pointer(dmp->dm_ctfp,
1206             CTF_ADD_ROOT, dtp->dt_type_func);

1208         /*
1209          * We also insert CTF definitions for the special D intrinsic types
1210          * string and <DYN> into the D container.  The string type is added
1211          * as a typedef of char[n].  The <DYN> type is an alias for void.
1212          * We compare types to these special CTF ids throughout the compiler.
1213          */
1214         ctr.ctr_contents = ctf_lookup_by_name(dmp->dm_ctfp, "char");
1215         ctr.ctr_index = ctf_lookup_by_name(dmp->dm_ctfp, "long");
1216         ctr.ctr_nelems = _dtrace_strsize;

1218         dtp->dt_type_str = ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1219             "string", ctf_add_array(dmp->dm_ctfp, CTF_ADD_ROOT, &ctr));

1221         dtp->dt_type_dyn = ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1222             "<DYN>", ctf_lookup_by_name(dmp->dm_ctfp, "void"));

1224         dtp->dt_type_stack = ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1225             "stack", ctf_lookup_by_name(dmp->dm_ctfp, "void"));

1227         dtp->dt_type_symaddr = ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1228             "_symaddr", ctf_lookup_by_name(dmp->dm_ctfp, "void"));

1230         dtp->dt_type_usymaddr = ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1231             "_usymaddr", ctf_lookup_by_name(dmp->dm_ctfp, "void"));

1233         if (dtp->dt_type_func == CTF_ERR || dtp->dt_type_fptr == CTF_ERR ||
1234             dtp->dt_type_str == CTF_ERR || dtp->dt_type_dyn == CTF_ERR ||
1235             dtp->dt_type_stack == CTF_ERR || dtp->dt_type_symaddr == CTF_ERR ||
1236             dtp->dt_type_usymaddr == CTF_ERR) {
1237                 dt_dprintf("failed to add intrinsic to D container: %s\n",
1238                     ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1239                 return (set_open_errno(dtp, errp, EDT_CTF));
1240         }

1242         if (ctf_update(dmp->dm_ctfp) != 0) {
```



```
1243                 dt_dprintf("failed update D container: %s\n",
1244                     ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1245                 return (set_open_errno(dtp, errp, EDT_CTF));
1246         }

1248         /*
1249          * Initialize the integer description table used to convert integer
1250          * constants to the appropriate types.  Refer to the comments above
1251          * dt_node_int() for a complete description of how this table is used.
1252          */
1253         for (i = 0; i < sizeof (dtp->dt_ints) / sizeof (dtp->dt_ints[0]); i++) {
1254                 if (dtrace_lookup_by_type(dtp, DTRACE_OBJ_EVERY,
1255                     dtp->dt_ints[i].did_name, &dtt) != 0) {
1256                         dt_dprintf("failed to lookup integer type %s: %s\n",
1257                             dtp->dt_ints[i].did_name,
1258                             dtrace_errmsg(dtp, dtrace_errno(dtp)));
1259                         return (set_open_errno(dtp, errp, dtp->dt_errno));
1260                 }
1261                 dtp->dt_ints[i].did_ctfp = dtt.dtt_ctfp;
1262                 dtp->dt_ints[i].did_type = dtt.dtt_type;
1263         }

1265         /*
1266          * Now that we've created the "C" and "D" containers, move them to the
1267          * start of the module list so that these types and symbols are found
1268          * first (for stability) when iterating through the module list.
1269          */
1270         dt_list_delete(&dtp->dt_modlist, dtp->dt_ddefs);
1271         dt_list_prepend(&dtp->dt_modlist, dtp->dt_ddefs);

1273         dt_list_delete(&dtp->dt_modlist, dtp->dt_cdefs);
1274         dt_list_prepend(&dtp->dt_modlist, dtp->dt_cdefs);

1276         if (dt_pfdict_create(dtp) == -1)
1277                 return (set_open_errno(dtp, errp, dtp->dt_errno));

1279         /*
1280          * If we are opening libdtrace DTRACE_O_NODEV enable C_ZDEFS by default
1281          * because without /dev/dtrace open, we will not be able to load the
1282          * names and attributes of any providers or probes from the kernel.
1283          */
1284         if (flags & DTRACE_O_NODEV)
1285                 dtp->dt_cflags |= DTRACE_C_ZDEFS;

1287         /*
1288          * Load hard-wired inlines into the definition cache by calling the
1289          * compiler on the raw definition string defined above.
1290          */
1291         if ((pgp = dtrace_program_strcompile(dtp, _dtrace_hardwire,
1292             DTRACE_PROBESPEC_NONE, DTRACE_C_EMPTY, 0, NULL)) == NULL) {
1293                 dt_dprintf("failed to load hard-wired definitions: %s\n",
1294                     dtrace_errmsg(dtp, dtrace_errno(dtp)));
1295                 return (set_open_errno(dtp, errp, EDT_HARDWIRE));
1296         }

1298         dt_program_destroy(dtp, pgp);

1300         /*
1301          * Set up the default DTrace library path.  Once set, the next call to
1302          * dt_compile() will compile all the libraries.  We intentionally defer
1303          * library processing to improve overhead for clients that don't ever
1304          * compile, and to provide better error reporting (because the full
1305          * reporting of compiler errors requires dtrace_open() to succeed).
1306          */
1307         if (dtrace_setopt(dtp, "libdir", _dtrace_libdir) != 0)
1308                 return (set_open_errno(dtp, errp, dtp->dt_errno));
```

```
1310         return (dtp);
1311 }

1313 dtrace_hdl_t *
1314 dtrace_open(int version, int flags, int *errp)
1315 {
1316         return (dt_vopen(version, flags, errp, NULL, NULL));
1317 }

1319 dtrace_hdl_t *
1320 dtrace_vopen(int version, int flags, int *errp,
1321     const dtrace_vector_t *vector, void *arg)
1322 {
1323         return (dt_vopen(version, flags, errp, vector, arg));
1324 }

1326 void
1327 dtrace_close(dtrace_hdl_t *dtp)
1328 {
1329         dt_ident_t *idp, *ndp;
1330         dt_module_t *dmp;
1331         dt_provider_t *pvp;
1332         dtrace_prog_t *pgp;
1333         dt_xlator_t *dxp;
1334         dt_dirpath_t *dirp;
1335         int i;

1337         if (dtp->dt_procs != NULL)
1338                 dt_proc_fini(dtp);

1340         while ((pgp = dt_list_next(&dtp->dt_programs)) != NULL)
1341                 dt_program_destroy(dtp, pgp);

1343         while ((dxp = dt_list_next(&dtp->dt_xlators)) != NULL)
1344                 dt_xlator_destroy(dtp, dxp);

1346         dt_free(dtp, dtp->dt_xlatormap);

1348         for (idp = dtp->dt_externs; idp != NULL; idp = ndp) {
1349                 ndp = idp->di_next;
1350                 dt_ident_destroy(idp);
1351         }

1353         if (dtp->dt_macros != NULL)
1354                 dt_idhash_destroy(dtp->dt_macros);
1355         if (dtp->dt_aggs != NULL)
1356                 dt_idhash_destroy(dtp->dt_aggs);
1357         if (dtp->dt_globals != NULL)
1358                 dt_idhash_destroy(dtp->dt_globals);
1359         if (dtp->dt_tls != NULL)
1360                 dt_idhash_destroy(dtp->dt_tls);

1362         while ((dmp = dt_list_next(&dtp->dt_modlist)) != NULL)
1363                 dt_module_destroy(dtp, dmp);

1365         while ((pvp = dt_list_next(&dtp->dt_provlist)) != NULL)
1366                 dt_provider_destroy(dtp, pvp);

1368         if (dtp->dt_fd != -1)
1369                 (void) close(dtp->dt_fd);
1370         if (dtp->dt_ftfd != -1)
1371                 (void) close(dtp->dt_ftfd);
1372         if (dtp->dt_cdefs_fd != -1)
1373                 (void) close(dtp->dt_cdefs_fd);
1374         if (dtp->dt_ddefs_fd != -1)
```



```
1375                 (void) close(dtp->dt_ddefs_fd);
1376         if (dtp->dt_stdout_fd != -1)
1377                 (void) close(dtp->dt_stdout_fd);

1379         dt_epid_destroy(dtp);
1380         dt_aggid_destroy(dtp);
1381         dt_format_destroy(dtp);
1382         dt_strdata_destroy(dtp);
1383         dt_buffered_destroy(dtp);
1384         dt_aggregate_destroy(dtp);
1385         dt_pfdict_destroy(dtp);
1386         dt_provmod_destroy(&dtp->dt_provmod);
1387         dt_dof_fini(dtp);

1389         for (i = 1; i < dtp->dt_cpp_argc; i++)
1390                 free(dtp->dt_cpp_argv[i]);

1392         while ((dirp = dt_list_next(&dtp->dt_lib_path)) != NULL) {
1393                 dt_list_delete(&dtp->dt_lib_path, dirp);
1394                 free(dirp->dir_path);
1395                 free(dirp);
1396         }

1398         free(dtp->dt_cpp_argv);
1399         free(dtp->dt_cpp_path);
1400         free(dtp->dt_ld_path);

1402         free(dtp->dt_mods);
1403         free(dtp->dt_provs);
1404         free(dtp);
1405 }

1407 int
1408 dtrace_provider_modules(dtrace_hdl_t *dtp, const char **mods, int nmods)
1409 {
1410         dt_provmod_t *prov;
1411         int i = 0;

1413         for (prov = dtp->dt_provmod; prov != NULL; prov = prov->dp_next, i++) {
1414                 if (i < nmods)
1415                         mods[i] = prov->dp_name;
1416         }

1418         return (i);
1419 }

1421 int
1422 dtrace_ctlfd(dtrace_hdl_t *dtp)
1423 {
1424         return (dtp->dt_fd);
1425 }
```

```
**********************************************************
  142714 Tue Jan 14 16:48:57 2014
new/usr/src/lib/libdtrace/common/dt_parser.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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, Joyent Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2011, Joyent Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

/*
 * DTrace D Language Parser
 *
 * The D Parser is a lex/yacc parser consisting of the lexer dt_lex.l, the
 * parsing grammar dt_grammar.y, and this file, dt_parser.c, which handles
 * the construction of the parse tree nodes and their syntactic validation.
 * The parse tree is constructed of dt_node_t structures (see <dt_parser.h>)
 * that are built in two passes: (1) the "create" pass, where the parse tree
 * nodes are allocated by calls from the grammar to dt_node_*() subroutines,
 * and (2) the "cook" pass, where nodes are coalesced, assigned D types, and
 * validated according to the syntactic rules of the language.
 *
 * All node allocations are performed using dt_node_alloc().  All node frees
 * during the parsing phase are performed by dt_node_free(), which frees node-
 * internal state but does not actually free the nodes.  All final node frees
 * are done as part of the end of dt_compile() or as part of destroying
 * persistent identifiers or translators which have embedded nodes.
 *
 * The dt_node_* routines that implement pass (1) may allocate new nodes.  The
 * dt_cook_* routines that implement pass (2) may *not* allocate new nodes.
 * They may free existing nodes using dt_node_free(), but they may not actually
 * deallocate any dt_node_t's.  Currently dt_cook_op2() is an exception to this
 * rule: see the comments therein for how this issue is resolved.
 *
 * The dt_cook_* routines are responsible for (at minimum) setting the final
 * node type (dn_ctfp/dn_type) and attributes (dn_attr).  If dn_ctfp/dn_type
 * are set manually (i.e. not by one of the type assignment functions), then
```



```
 * the DT_NF_COOKED flag must be set manually on the node.
 *
 * The cooking pass can be applied to the same parse tree more than once (used
 * in the case of a comma-separated list of probe descriptions).  As such, the
 * cook routines must not perform any parse tree transformations which would
 * be invalid if the tree were subsequently cooked using a different context.
 *
 * The dn_ctfp and dn_type fields form the type of the node.  This tuple can
 * take on the following set of values, which form our type invariants:
 *
 * 1. dn_ctfp = NULL, dn_type = CTF_ERR
 *
 *    In this state, the node has unknown type and is not yet cooked.  The
 *    DT_NF_COOKED flag is not yet set on the node.
 *
 * 2. dn_ctfp = DT_DYN_CTFP(dtp), dn_type = DT_DYN_TYPE(dtp)
 *
 *    In this state, the node is a dynamic D type.  This means that generic
 *    operations are not valid on this node and only code that knows how to
 *    examine the inner details of the node can operate on it.  A <DYN> node
 *    must have dn_ident set to point to an identifier describing the object
 *    and its type.  The DT_NF_REF flag is set for all nodes of type <DYN>.
 *    At present, the D compiler uses the <DYN> type for:
 *
 *    - associative arrays that do not yet have a value type defined
 *    - translated data (i.e. the result of the xlate operator)
 *    - aggregations
 *
 * 3. dn_ctfp = DT_STR_CTFP(dtp), dn_type = DT_STR_TYPE(dtp)
 *
 *    In this state, the node is of type D string.  The string type is really
 *    a char[0] typedef, but requires special handling throughout the compiler.
 *
 * 4. dn_ctfp != NULL, dn_type = any other type ID
 *
 *    In this state, the node is of some known D/CTF type.  The normal libctf
 *    APIs can be used to learn more about the type name or structure.  When
 *    the type is assigned, the DT_NF_SIGNED, DT_NF_REF, and DT_NF_BITFIELD
 *    flags cache the corresponding attributes of the underlying CTF type.
 */

#include <sys/param.h>
#include <sys/sysmacros.h>
#include <limits.h>
#include <setjmp.h>
#include <strings.h>
#include <assert.h>
#include <alloca.h>
#include <stdlib.h>
#include <stdarg.h>
#include <stdio.h>
#include <errno.h>
#include <ctype.h>

#include <dt_impl.h>
#include <dt_grammar.h>
#include <dt_module.h>
#include <dt_provider.h>
#include <dt_string.h>
#include <dt_as.h>

dt_pcb_t *yypcb;        /* current control block for parser */
dt_node_t *yypragma;    /* lex token list for control lines */
char yyintprefix;       /* int token macro prefix (+/-) */
char yyintsuffix[4];    /* int token suffix string [uU][lL] */
int yyintdecimal;       /* int token format flag (1=decimal, 0=octal/hex) */
```

```
122 static const char *
123 opstr(int op)
124 {
125         switch (op) {
126         case DT_TOK_COMMA:      return (",");
127         case DT_TOK_ELLIPSIS:   return ("...");
128         case DT_TOK_ASGN:       return ("=");
129         case DT_TOK_ADD_EQ:     return ("+=");
130         case DT_TOK_SUB_EQ:     return ("-=");
131         case DT_TOK_MUL_EQ:     return ("*=");
132         case DT_TOK_DIV_EQ:     return ("/=");
133         case DT_TOK_MOD_EQ:     return ("%=");
134         case DT_TOK_AND_EQ:     return ("&=");
135         case DT_TOK_XOR_EQ:     return ("^=");
136         case DT_TOK_OR_EQ:      return ("|=");
137         case DT_TOK_LSH_EQ:     return ("<<=");
138         case DT_TOK_RSH_EQ:     return (">>=");
139         case DT_TOK_QUESTION:   return ("?");
140         case DT_TOK_COLON:      return (":");
141         case DT_TOK_LOR:        return ("||");
142         case DT_TOK_LXOR:       return ("^^");
143         case DT_TOK_LAND:       return ("&&");
144         case DT_TOK_BOR:        return ("|");
145         case DT_TOK_XOR:        return ("^");
146         case DT_TOK_BAND:       return ("&");
147         case DT_TOK_EQU:        return ("==");
148         case DT_TOK_NEQ:        return ("!=");
149         case DT_TOK_LT:         return ("<");
150         case DT_TOK_LE:         return ("<=");
151         case DT_TOK_GT:         return (">");
152         case DT_TOK_GE:         return (">=");
153         case DT_TOK_LSH:        return ("<<");
154         case DT_TOK_RSH:        return (">>");
155         case DT_TOK_ADD:        return ("+");
156         case DT_TOK_SUB:        return ("-");
157         case DT_TOK_MUL:        return ("*");
158         case DT_TOK_DIV:        return ("/");
159         case DT_TOK_MOD:        return ("%");
160         case DT_TOK_LNEG:       return ("!");
161         case DT_TOK_BNEG:       return ("~");
162         case DT_TOK_ADDADD:     return ("++");
163         case DT_TOK_PREINC:     return ("++");
164         case DT_TOK_POSTINC:    return ("++");
165         case DT_TOK_SUBSUB:     return ("--");
166         case DT_TOK_PREDEC:     return ("--");
167         case DT_TOK_POSTDEC:    return ("--");
168         case DT_TOK_IPOS:       return ("+");
169         case DT_TOK_INEG:       return ("-");
170         case DT_TOK_DEREF:      return ("*");
171         case DT_TOK_ADDROF:     return ("&");
172         case DT_TOK_OFFSETOF:   return ("offsetof");
173         case DT_TOK_SIZEOF:     return ("sizeof");
174         case DT_TOK_STRINGOF:   return ("stringof");
175         case DT_TOK_XLATE:      return ("xlate");
176         case DT_TOK_LPAR:       return ("(");
177         case DT_TOK_RPAR:       return (")");
178         case DT_TOK_LBRAC:      return ("[");
179         case DT_TOK_RBRAC:      return ("]");
180         case DT_TOK_PTR:        return ("->");
181         case DT_TOK_DOT:        return (".");
182         case DT_TOK_STRING:     return ("<string>");
183         case DT_TOK_IDENT:      return ("<ident>");
184         case DT_TOK_TNAME:      return ("<type>");
185         case DT_TOK_INT:        return ("<int>");
186         default:                return ("<?>");
```



```
187         }
188 }

190 int
191 dt_type_lookup(const char *s, dtrace_typeinfo_t *tip)
192 {
193         static const char delimiters[] = " \t\n\r\v\f*'";
194         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
195         const char *p, *q, *r, *end, *obj;
195         const char *p, *q, *end, *obj;

197         for (p = s, end = s + strlen(s); *p != '\0'; p = q) {
198                 while (isspace(*p))
199                         p++;    /* skip leading whitespace prior to token */

201                 if (p == end || (q = strpbrk(p + 1, delimiters)) == NULL)
202                         break;  /* empty string or single token remaining */

204                 if (*q == '`') {
205                         char *object = alloca((size_t)(q - p) + 1);
206                         char *type = alloca((size_t)(end - s) + 1);

208                         /*
209                          * Copy from the start of the token (p) to the location
210                          * backquote (q) to extract the nul-terminated object.
211                          */
212                         bcopy(p, object, (size_t)(q - p));
213                         object[(size_t)(q - p)] = '\0';

215                         /*
216                          * Copy the original string up to the start of this
217                          * token (p) into type, and then concatenate everything
218                          * after q.  This is the type name without the object.
219                          */
220                         bcopy(s, type, (size_t)(p - s));
221                         bcopy(q + 1, type + (size_t)(p - s), strlen(q + 1) + 1);

223                         /*
224                          * There may be at most three delimeters. The second
225                          * delimeter is usually used to distinguish the type
226                          * within a given module, however, there could be a link
227                          * map id on the scene in which case that delimeter
228                          * would be the third. We determine presence of the lmid
229                          * if it rouglhly meets the from LM[0-9]
230                          */
231                         if ((r = strchr(q + 1, '`')) != NULL &&
232                             ((r = strchr(r + 1, '`')) != NULL)) {
233                                 if (strchr(r + 1, '`') != NULL)
234                                         return (dt_set_errno(dtp,
235                                             EDT_BADSCOPE));
236                                 if (q[1] != 'L' || q[2] != 'M')
237                                         return (dt_set_errno(dtp,
238                                             EDT_BADSCOPE));
239                         }
223                         if (strchr(q + 1, '`') != NULL)
224                                 return (dt_set_errno(dtp, EDT_BADSCOPE));

241                         return (dtrace_lookup_by_type(dtp, object, type, tip));
242                 }
243         }

245         if (yypcb->pcb_idepth != 0)
246                 obj = DTRACE_OBJ_CDEFS;
247         else
248                 obj = DTRACE_OBJ_EVERY;
```

```
250         return (dtrace_lookup_by_type(dtp, obj, s, tip));
251 }

253 /*
254  * When we parse type expressions or parse an expression with unary "&", we
255  * need to find a type that is a pointer to a previously known type.
256  * Unfortunately CTF is limited to a per-container view, so ctf_type_pointer()
257  * alone does not suffice for our needs.  We provide a more intelligent wrapper
258  * for the compiler that attempts to compute a pointer to either the given type
259  * or its base (that is, we try both "foo_t *" and "struct foo *"), and also
260  * to potentially construct the required type on-the-fly.
261  */
262 int
263 dt_type_pointer(dtrace_typeinfo_t *tip)
264 {
265         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
266         ctf_file_t *ctfp = tip->dtt_ctfp;
267         ctf_id_t type = tip->dtt_type;
268         ctf_id_t base = ctf_type_resolve(ctfp, type);
269         uint_t bflags = tip->dtt_flags;
270 #endif /* ! codereview */

272         dt_module_t *dmp;
273         ctf_id_t ptr;

275         if ((ptr = ctf_type_pointer(ctfp, type)) != CTF_ERR ||
276             (ptr = ctf_type_pointer(ctfp, base)) != CTF_ERR) {
277                 tip->dtt_type = ptr;
278                 return (0);
279         }

281         if (yypcb->pcb_idepth != 0)
282                 dmp = dtp->dt_cdefs;
283         else
284                 dmp = dtp->dt_ddefs;

286         if (ctfp != dmp->dm_ctfp && ctfp != ctf_parent_file(dmp->dm_ctfp) &&
287             (type = ctf_add_type(dmp->dm_ctfp, ctfp, type)) == CTF_ERR) {
288                 dtp->dt_ctferr = ctf_errno(dmp->dm_ctfp);
289                 return (dt_set_errno(dtp, EDT_CTF));
290         }

292         ptr = ctf_add_pointer(dmp->dm_ctfp, CTF_ADD_ROOT, type);

294         if (ptr == CTF_ERR || ctf_update(dmp->dm_ctfp) == CTF_ERR) {
295                 dtp->dt_ctferr = ctf_errno(dmp->dm_ctfp);
296                 return (dt_set_errno(dtp, EDT_CTF));
297         }

299         tip->dtt_object = dmp->dm_name;
300         tip->dtt_ctfp = dmp->dm_ctfp;
301         tip->dtt_type = ptr;
302         tip->dtt_flags = bflags;
303 #endif /* ! codereview */

305         return (0);
306 }

308 const char *
309 dt_type_name(ctf_file_t *ctfp, ctf_id_t type, char *buf, size_t len)
310 {
311         dtrace_hdl_t *dtp = yypcb->pcb_hdl;

313         if (ctfp == DT_FPTR_CTFP(dtp) && type == DT_FPTR_TYPE(dtp))
314                 (void) snprintf(buf, len, "function pointer");
315         else if (ctfp == DT_FUNC_CTFP(dtp) && type == DT_FUNC_TYPE(dtp))
```



```
316                 (void) snprintf(buf, len, "function");
317         else if (ctfp == DT_DYN_CTFP(dtp) && type == DT_DYN_TYPE(dtp))
318                 (void) snprintf(buf, len, "dynamic variable");
319         else if (ctfp == NULL)
320                 (void) snprintf(buf, len, "<none>");
321         else if (ctf_type_name(ctfp, type, buf, len) == NULL)
322                 (void) snprintf(buf, len, "unknown");

324         return (buf);
325 }

327 /*
328  * Perform the "usual arithmetic conversions" to determine which of the two
329  * input operand types should be promoted and used as a result type.  The
330  * rules for this are described in ISOC[6.3.1.8] and K&R[A6.5].
331  */
332 static void
333 dt_type_promote(dt_node_t *lp, dt_node_t *rp, ctf_file_t **ofp, ctf_id_t *otype)
334 {
335         ctf_file_t *lfp = lp->dn_ctfp;
336         ctf_id_t ltype = lp->dn_type;

338         ctf_file_t *rfp = rp->dn_ctfp;
339         ctf_id_t rtype = rp->dn_type;

341         ctf_id_t lbase = ctf_type_resolve(lfp, ltype);
342         uint_t lkind = ctf_type_kind(lfp, lbase);

344         ctf_id_t rbase = ctf_type_resolve(rfp, rtype);
345         uint_t rkind = ctf_type_kind(rfp, rbase);

347         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
348         ctf_encoding_t le, re;
349         uint_t lrank, rrank;

351         assert(lkind == CTF_K_INTEGER || lkind == CTF_K_ENUM);
352         assert(rkind == CTF_K_INTEGER || rkind == CTF_K_ENUM);

354         if (lkind == CTF_K_ENUM) {
355                 lfp = DT_INT_CTFP(dtp);
356                 ltype = lbase = DT_INT_TYPE(dtp);
357         }

359         if (rkind == CTF_K_ENUM) {
360                 rfp = DT_INT_CTFP(dtp);
361                 rtype = rbase = DT_INT_TYPE(dtp);
362         }

364         if (ctf_type_encoding(lfp, lbase, &le) == CTF_ERR) {
365                 yypcb->pcb_hdl->dt_ctferr = ctf_errno(lfp);
366                 longjmp(yypcb->pcb_jmpbuf, EDT_CTF);
367         }

369         if (ctf_type_encoding(rfp, rbase, &re) == CTF_ERR) {
370                 yypcb->pcb_hdl->dt_ctferr = ctf_errno(rfp);
371                 longjmp(yypcb->pcb_jmpbuf, EDT_CTF);
372         }

374         /*
375          * Compute an integer rank based on the size and unsigned status.
376          * If rank is identical, pick the "larger" of the equivalent types
377          * which we define as having a larger base ctf_id_t.  If rank is
378          * different, pick the type with the greater rank.
379          */
380         lrank = le.cte_bits + ((le.cte_format & CTF_INT_SIGNED) == 0);
381         rrank = re.cte_bits + ((re.cte_format & CTF_INT_SIGNED) == 0);
```

```
383         if (lrank == rrank) {
384                 if (lbase - rbase < 0)
385                         goto return_rtype;
386                 else
387                         goto return_ltype;
388         } else if (lrank > rrank) {
389                 goto return_ltype;
390         } else
391                 goto return_rtype;

393 return_ltype:
394         *ofp = lfp;
395         *otype = ltype;
396         return;

398 return_rtype:
399         *ofp = rfp;
400         *otype = rtype;
401 }

403 void
404 dt_node_promote(dt_node_t *lp, dt_node_t *rp, dt_node_t *dnp)
405 {
406         dt_type_promote(lp, rp, &dnp->dn_ctfp, &dnp->dn_type);
407         dt_node_type_assign(dnp, dnp->dn_ctfp, dnp->dn_type, B_FALSE);
254         dt_node_type_assign(dnp, dnp->dn_ctfp, dnp->dn_type);
408         dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));
409 }
```

*unchanged_portion_omitted_*

```
675 void
676 dt_node_type_assign(dt_node_t *dnp, ctf_file_t *fp, ctf_id_t type,
677     boolean_t user)
523 dt_node_type_assign(dt_node_t *dnp, ctf_file_t *fp, ctf_id_t type)
678 {
679         ctf_id_t base = ctf_type_resolve(fp, type);
680         uint_t kind = ctf_type_kind(fp, base);
681         ctf_encoding_t e;

683         dnp->dn_flags &=
684             ~(DT_NF_SIGNED | DT_NF_REF | DT_NF_BITFIELD | DT_NF_USERLAND);

686         if (kind == CTF_K_INTEGER && ctf_type_encoding(fp, base, &e) == 0) {
687                 size_t size = e.cte_bits / NBBY;

689                 if (size > 8 || (e.cte_bits % NBBY) != 0 || (size & (size - 1)))
690                         dnp->dn_flags |= DT_NF_BITFIELD;

692                 if (e.cte_format & CTF_INT_SIGNED)
693                         dnp->dn_flags |= DT_NF_SIGNED;
694         }

696         if (kind == CTF_K_FLOAT && ctf_type_encoding(fp, base, &e) == 0) {
697                 if (e.cte_bits / NBBY > sizeof (uint64_t))
698                         dnp->dn_flags |= DT_NF_REF;
699         }

701         if (kind == CTF_K_STRUCT || kind == CTF_K_UNION ||
702             kind == CTF_K_FORWARD ||
703             kind == CTF_K_ARRAY || kind == CTF_K_FUNCTION)
704                 dnp->dn_flags |= DT_NF_REF;
705         else if (yypcb != NULL && fp == DT_DYN_CTFP(yypcb->pcb_hdl) &&
706             type == DT_DYN_TYPE(yypcb->pcb_hdl))
707                 dnp->dn_flags |= DT_NF_REF;
```



```
709         if (user)
710                 dnp->dn_flags |= DT_NF_USERLAND;

712 #endif /* ! codereview */
713         dnp->dn_flags |= DT_NF_COOKED;
714         dnp->dn_ctfp = fp;
715         dnp->dn_type = type;
716 }

718 void
719 dt_node_type_propagate(const dt_node_t *src, dt_node_t *dst)
720 {
721         assert(src->dn_flags & DT_NF_COOKED);
722         dst->dn_flags = src->dn_flags & ~DT_NF_LVALUE;
723         dst->dn_ctfp = src->dn_ctfp;
724         dst->dn_type = src->dn_type;
725 }

727 const char *
728 dt_node_type_name(const dt_node_t *dnp, char *buf, size_t len)
729 {
730         if (dt_node_is_dynamic(dnp) && dnp->dn_ident != NULL) {
731                 (void) snprintf(buf, len, "%s",
732                     dt_idkind_name(dt_ident_resolve(dnp->dn_ident)->di_kind));
733                 return (buf);
734         }

736         if (dnp->dn_flags & DT_NF_USERLAND) {
737                 size_t n = snprintf(buf, len, "userland ");
738                 len = len > n ? len - n : 0;
739                 (void) dt_type_name(dnp->dn_ctfp, dnp->dn_type, buf + n, len);
740                 return (buf);
741         }

743         return (dt_type_name(dnp->dn_ctfp, dnp->dn_type, buf, len));
744 }

746 size_t
747 dt_node_type_size(const dt_node_t *dnp)
748 {
749         ctf_id_t base;
750         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
751 #endif /* ! codereview */

753         if (dnp->dn_kind == DT_NODE_STRING)
754                 return (strlen(dnp->dn_string) + 1);

756         if (dt_node_is_dynamic(dnp) && dnp->dn_ident != NULL)
757                 return (dt_ident_size(dnp->dn_ident));

759         base = ctf_type_resolve(dnp->dn_ctfp, dnp->dn_type);

761         if (ctf_type_kind(dnp->dn_ctfp, base) == CTF_K_FORWARD)
762                 return (0);

764         /*
765          * Here we have a 32-bit user pointer that is being used with a 64-bit
766          * kernel. When we're using it and its tagged as a userland reference --
767          * then we need to keep it as a 32-bit pointer. However, if we are
768          * referring to it as a kernel address, eg. being used after a copyin()
769          * then we need to make sure that we actually return the kernel's size
770          * of a pointer, 8 bytes.
771          */
772         if (ctf_type_kind(dnp->dn_ctfp, base) == CTF_K_POINTER &&
773             ctf_getmodel(dnp->dn_ctfp) == CTF_MODEL_ILP32 &&
774             !(dnp->dn_flags & DT_NF_USERLAND) &&
```

```
775             dtp->dt_conf.dtc_ctfmodel == CTF_MODEL_LP64)
776                     return (8);

778 #endif /* ! codereview */
779         return (ctf_type_size(dnp->dn_ctfp, dnp->dn_type));
780 }

782 /*
783  * Determine if the specified parse tree node references an identifier of the
784  * specified kind, and if so return a pointer to it; otherwise return NULL.
785  * This function resolves the identifier itself, following through any inlines.
786  */
787 dt_ident_t *
788 dt_node_resolve(const dt_node_t *dnp, uint_t idkind)
789 {
790         dt_ident_t *idp;

792         switch (dnp->dn_kind) {
793         case DT_NODE_VAR:
794         case DT_NODE_SYM:
795         case DT_NODE_FUNC:
796         case DT_NODE_AGG:
797         case DT_NODE_INLINE:
798         case DT_NODE_PROBE:
799                 idp = dt_ident_resolve(dnp->dn_ident);
800                 return (idp->di_kind == idkind ? idp : NULL);
801         }

803         if (dt_node_is_dynamic(dnp)) {
804                 idp = dt_ident_resolve(dnp->dn_ident);
805                 return (idp->di_kind == idkind ? idp : NULL);
806         }

808         return (NULL);
809 }

811 size_t
812 dt_node_sizeof(const dt_node_t *dnp)
813 {
814         dtrace_syminfo_t *sip;
815         GElf_Sym sym;
816         dtrace_hdl_t *dtp = yypcb->pcb_hdl;

818         /*
819          * The size of the node as used for the sizeof() operator depends on
820          * the kind of the node.  If the node is a SYM, the size is obtained
821          * from the symbol table; if it is not a SYM, the size is determined
822          * from the node's type.  This is slightly different from C's sizeof()
823          * operator in that (for example) when applied to a function, sizeof()
824          * will evaluate to the length of the function rather than the size of
825          * the function type.
826          */
827         if (dnp->dn_kind != DT_NODE_SYM)
828                 return (dt_node_type_size(dnp));

830         sip = dnp->dn_ident->di_data;

832         if (dtrace_lookup_by_name(dtp, sip->dts_object,
833             sip->dts_name, &sym, NULL) == -1)
834                 return (0);

836         return (sym.st_size);
837 }

839 int
840 dt_node_is_integer(const dt_node_t *dnp)
```



```
841 {
842         ctf_file_t *fp = dnp->dn_ctfp;
843         ctf_encoding_t e;
844         ctf_id_t type;
845         uint_t kind;

847         assert(dnp->dn_flags & DT_NF_COOKED);

849         type = ctf_type_resolve(fp, dnp->dn_type);
850         kind = ctf_type_kind(fp, type);

852         if (kind == CTF_K_INTEGER &&
853             ctf_type_encoding(fp, type, &e) == 0 && IS_VOID(e))
854                 return (0); /* void integer */

856         return (kind == CTF_K_INTEGER || kind == CTF_K_ENUM);
857 }

859 int
860 dt_node_is_float(const dt_node_t *dnp)
861 {
862         ctf_file_t *fp = dnp->dn_ctfp;
863         ctf_encoding_t e;
864         ctf_id_t type;
865         uint_t kind;

867         assert(dnp->dn_flags & DT_NF_COOKED);

869         type = ctf_type_resolve(fp, dnp->dn_type);
870         kind = ctf_type_kind(fp, type);

872         return (kind == CTF_K_FLOAT &&
873             ctf_type_encoding(dnp->dn_ctfp, type, &e) == 0 && (
874             e.cte_format == CTF_FP_SINGLE || e.cte_format == CTF_FP_DOUBLE ||
875             e.cte_format == CTF_FP_LDOUBLE));
876 }

878 int
879 dt_node_is_scalar(const dt_node_t *dnp)
880 {
881         ctf_file_t *fp = dnp->dn_ctfp;
882         ctf_encoding_t e;
883         ctf_id_t type;
884         uint_t kind;

886         assert(dnp->dn_flags & DT_NF_COOKED);

888         type = ctf_type_resolve(fp, dnp->dn_type);
889         kind = ctf_type_kind(fp, type);

891         if (kind == CTF_K_INTEGER &&
892             ctf_type_encoding(fp, type, &e) == 0 && IS_VOID(e))
893                 return (0); /* void cannot be used as a scalar */

895         return (kind == CTF_K_INTEGER || kind == CTF_K_ENUM ||
896             kind == CTF_K_POINTER);
897 }

899 int
900 dt_node_is_arith(const dt_node_t *dnp)
901 {
902         ctf_file_t *fp = dnp->dn_ctfp;
903         ctf_encoding_t e;
904         ctf_id_t type;
905         uint_t kind;
```

```
907         assert(dnp->dn_flags & DT_NF_COOKED);

909         type = ctf_type_resolve(fp, dnp->dn_type);
910         kind = ctf_type_kind(fp, type);

912         if (kind == CTF_K_INTEGER)
913                 return (ctf_type_encoding(fp, type, &e) == 0 && !IS_VOID(e));
914         else
915                 return (kind == CTF_K_ENUM);
916 }

918 int
919 dt_node_is_vfptr(const dt_node_t *dnp)
920 {
921         ctf_file_t *fp = dnp->dn_ctfp;
922         ctf_encoding_t e;
923         ctf_id_t type;
924         uint_t kind;

926         assert(dnp->dn_flags & DT_NF_COOKED);

928         type = ctf_type_resolve(fp, dnp->dn_type);
929         if (ctf_type_kind(fp, type) != CTF_K_POINTER)
930                 return (0); /* type is not a pointer */

932         type = ctf_type_resolve(fp, ctf_type_reference(fp, type));
933         kind = ctf_type_kind(fp, type);

935         return (kind == CTF_K_FUNCTION || (kind == CTF_K_INTEGER &&
936             ctf_type_encoding(fp, type, &e) == 0 && IS_VOID(e)));
937 }

939 int
940 dt_node_is_dynamic(const dt_node_t *dnp)
941 {
942         if (dnp->dn_kind == DT_NODE_VAR &&
943             (dnp->dn_ident->di_flags & DT_IDFLG_INLINE)) {
944                 const dt_idnode_t *inp = dnp->dn_ident->di_iarg;
945                 return (inp->din_root ? dt_node_is_dynamic(inp->din_root) : 0);
946         }

948         return (dnp->dn_ctfp == DT_DYN_CTFP(yypcb->pcb_hdl) &&
949             dnp->dn_type == DT_DYN_TYPE(yypcb->pcb_hdl));
950 }

952 int
953 dt_node_is_string(const dt_node_t *dnp)
954 {
955         return (dnp->dn_ctfp == DT_STR_CTFP(yypcb->pcb_hdl) &&
956             dnp->dn_type == DT_STR_TYPE(yypcb->pcb_hdl));
957 }

959 int
960 dt_node_is_stack(const dt_node_t *dnp)
961 {
962         return (dnp->dn_ctfp == DT_STACK_CTFP(yypcb->pcb_hdl) &&
963             dnp->dn_type == DT_STACK_TYPE(yypcb->pcb_hdl));
964 }

966 int
967 dt_node_is_symaddr(const dt_node_t *dnp)
968 {
969         return (dnp->dn_ctfp == DT_SYMADDR_CTFP(yypcb->pcb_hdl) &&
970             dnp->dn_type == DT_SYMADDR_TYPE(yypcb->pcb_hdl));
971 }
```



```
973 int
974 dt_node_is_usymaddr(const dt_node_t *dnp)
975 {
976         return (dnp->dn_ctfp == DT_USYMADDR_CTFP(yypcb->pcb_hdl) &&
977             dnp->dn_type == DT_USYMADDR_TYPE(yypcb->pcb_hdl));
978 }

980 int
981 dt_node_is_strcompat(const dt_node_t *dnp)
982 {
983         ctf_file_t *fp = dnp->dn_ctfp;
984         ctf_encoding_t e;
985         ctf_arinfo_t r;
986         ctf_id_t base;
987         uint_t kind;

989         assert(dnp->dn_flags & DT_NF_COOKED);

991         base = ctf_type_resolve(fp, dnp->dn_type);
992         kind = ctf_type_kind(fp, base);

994         if (kind == CTF_K_POINTER &&
995             (base = ctf_type_reference(fp, base)) != CTF_ERR &&
996             (base = ctf_type_resolve(fp, base)) != CTF_ERR &&
997             ctf_type_encoding(fp, base, &e) == 0 && IS_CHAR(e))
998                 return (1); /* promote char pointer to string */

1000        if (kind == CTF_K_ARRAY && ctf_array_info(fp, base, &r) == 0 &&
1001            (base = ctf_type_resolve(fp, r.ctr_contents)) != CTF_ERR &&
1002            ctf_type_encoding(fp, base, &e) == 0 && IS_CHAR(e))
1003                return (1); /* promote char array to string */

1005        return (0);
1006 }

1008 int
1009 dt_node_is_pointer(const dt_node_t *dnp)
1010 {
1011        ctf_file_t *fp = dnp->dn_ctfp;
1012        uint_t kind;

1014        assert(dnp->dn_flags & DT_NF_COOKED);

1016        if (dt_node_is_string(dnp))
1017                return (0); /* string are pass-by-ref but act like structs */

1019        kind = ctf_type_kind(fp, ctf_type_resolve(fp, dnp->dn_type));
1020        return (kind == CTF_K_POINTER || kind == CTF_K_ARRAY);
1021 }

1023 int
1024 dt_node_is_void(const dt_node_t *dnp)
1025 {
1026        ctf_file_t *fp = dnp->dn_ctfp;
1027        ctf_encoding_t e;
1028        ctf_id_t type;

1030        if (dt_node_is_dynamic(dnp))
1031                return (0); /* <DYN> is an alias for void but not the same */

1033        if (dt_node_is_stack(dnp))
1034                return (0);

1036        if (dt_node_is_symaddr(dnp) || dt_node_is_usymaddr(dnp))
1037                return (0);
```

```
1039         type = ctf_type_resolve(fp, dnp->dn_type);
1040
1041         return (ctf_type_kind(fp, type) == CTF_K_INTEGER &&
1042             ctf_type_encoding(fp, type, &e) == 0 && IS_VOID(e));
1043 }
1044
1045 int
1046 dt_node_is_ptrcompat(const dt_node_t *lp, const dt_node_t *rp,
1047     ctf_file_t **fpp, ctf_id_t *tp)
1048 {
1049         ctf_file_t *lfp = lp->dn_ctfp;
1050         ctf_file_t *rfp = rp->dn_ctfp;
1051
1052         ctf_id_t lbase = CTF_ERR, rbase = CTF_ERR;
1053         ctf_id_t lref = CTF_ERR, rref = CTF_ERR;
1054
1055         int lp_is_void, rp_is_void, lp_is_int, rp_is_int, compat;
1056         uint_t lkind, rkind;
1057         ctf_encoding_t e;
1058         ctf_arinfo_t r;
1059
1060         assert(lp->dn_flags & DT_NF_COOKED);
1061         assert(rp->dn_flags & DT_NF_COOKED);
1062
1063         if (dt_node_is_dynamic(lp) || dt_node_is_dynamic(rp))
1064                 return (0); /* fail if either node is a dynamic variable */
1065
1066         lp_is_int = dt_node_is_integer(lp);
1067         rp_is_int = dt_node_is_integer(rp);
1068
1069         if (lp_is_int && rp_is_int)
1070                 return (0); /* fail if both nodes are integers */
1071
1072         if (lp_is_int && (lp->dn_kind != DT_NODE_INT || lp->dn_value != 0))
1073                 return (0); /* fail if lp is an integer that isn't 0 constant */
1074
1075         if (rp_is_int && (rp->dn_kind != DT_NODE_INT || rp->dn_value != 0))
1076                 return (0); /* fail if rp is an integer that isn't 0 constant */
1077
1078         if ((lp_is_int == 0 && rp_is_int == 0) && (
1079             (lp->dn_flags & DT_NF_USERLAND) ^ (rp->dn_flags & DT_NF_USERLAND)))
1080                 return (0); /* fail if only one pointer is a userland address */
1081
1082         /*
1083          * Resolve the left-hand and right-hand types to their base type, and
1084          * then resolve the referenced type as well (assuming the base type
1085          * is CTF_K_POINTER or CTF_K_ARRAY).  Otherwise [lr]ref = CTF_ERR.
1086          */
1087         if (!lp_is_int) {
1088                 lbase = ctf_type_resolve(lfp, lp->dn_type);
1089                 lkind = ctf_type_kind(lfp, lbase);
1090
1091                 if (lkind == CTF_K_POINTER) {
1092                         lref = ctf_type_resolve(lfp,
1093                             ctf_type_reference(lfp, lbase));
1094                 } else if (lkind == CTF_K_ARRAY &&
1095                     ctf_array_info(lfp, lbase, &r) == 0) {
1096                         lref = ctf_type_resolve(lfp, r.ctr_contents);
1097                 }
1098         }
1099
1100         if (!rp_is_int) {
1101                 rbase = ctf_type_resolve(rfp, rp->dn_type);
1102                 rkind = ctf_type_kind(rfp, rbase);
1103
1104                 if (rkind == CTF_K_POINTER) {
```



```
1105                         rref = ctf_type_resolve(rfp,
1106                             ctf_type_reference(rfp, rbase));
1107                 } else if (rkind == CTF_K_ARRAY &&
1108                     ctf_array_info(rfp, rbase, &r) == 0) {
1109                         rref = ctf_type_resolve(rfp, r.ctr_contents);
1110                 }
1111         }
1112
1113         /*
1114          * We know that one or the other type may still be a zero-valued
1115          * integer constant.  To simplify the code below, set the integer
1116          * type variables equal to the non-integer types and proceed.
1117          */
1118         if (lp_is_int) {
1119                 lbase = rbase;
1120                 lkind = rkind;
1121                 lref = rref;
1122                 lfp = rfp;
1123         } else if (rp_is_int) {
1124                 rbase = lbase;
1125                 rkind = lkind;
1126                 rref = lref;
1127                 rfp = lfp;
1128         }
1129
1130         lp_is_void = ctf_type_encoding(lfp, lref, &e) == 0 && IS_VOID(e);
1131         rp_is_void = ctf_type_encoding(rfp, rref, &e) == 0 && IS_VOID(e);
1132
1133         /*
1134          * The types are compatible if both are pointers to the same type, or
1135          * if either pointer is a void pointer.  If they are compatible, set
1136          * tp to point to the more specific pointer type and return it.
1137          */
1138         compat = (lkind == CTF_K_POINTER || lkind == CTF_K_ARRAY) &&
1139             (rkind == CTF_K_POINTER || rkind == CTF_K_ARRAY) &&
1140             (lp_is_void || rp_is_void || ctf_type_compat(lfp, lref, rfp, rref));
1141
1142         if (compat) {
1143                 if (fpp != NULL)
1144                         *fpp = rp_is_void ? lfp : rfp;
1145                 if (tp != NULL)
1146                         *tp = rp_is_void ? lbase : rbase;
1147         }
1148
1149         return (compat);
1150 }
1151
1152 /*
1153  * The rules for checking argument types against parameter types are described
1154  * in the ANSI-C spec (see K&R[A7.3.2] and K&R[A7.17]).  We use the same rule
1155  * set to determine whether associative array arguments match the prototype.
1156  */
1157 int
1158 dt_node_is_argcompat(const dt_node_t *lp, const dt_node_t *rp)
1159 {
1160         ctf_file_t *lfp = lp->dn_ctfp;
1161         ctf_file_t *rfp = rp->dn_ctfp;
1162
1163         assert(lp->dn_flags & DT_NF_COOKED);
1164         assert(rp->dn_flags & DT_NF_COOKED);
1165
1166         if (dt_node_is_integer(lp) && dt_node_is_integer(rp))
1167                 return (1); /* integer types are compatible */
1168
1169         if (dt_node_is_strcompat(lp) && dt_node_is_strcompat(rp))
1170                 return (1); /* string types are compatible */
```

```
1172         if (dt_node_is_stack(lp) && dt_node_is_stack(rp))
1173                 return (1); /* stack types are compatible */

1175         if (dt_node_is_symaddr(lp) && dt_node_is_symaddr(rp))
1176                 return (1); /* symaddr types are compatible */

1178         if (dt_node_is_usymaddr(lp) && dt_node_is_usymaddr(rp))
1179                 return (1); /* usymaddr types are compatible */

1181         switch (ctf_type_kind(lfp, ctf_type_resolve(lfp, lp->dn_type))) {
1182         case CTF_K_FUNCTION:
1183         case CTF_K_STRUCT:
1184         case CTF_K_UNION:
1185                 return (ctf_type_compat(lfp, lp->dn_type, rfp, rp->dn_type));
1186         default:
1187                 return (dt_node_is_ptrcompat(lp, rp, NULL, NULL));
1188         }
1189 }

1191 /*
1192  * We provide dt_node_is_posconst() as a convenience routine for callers who
1193  * wish to verify that an argument is a positive non-zero integer constant.
1194  */
1195 int
1196 dt_node_is_posconst(const dt_node_t *dnp)
1197 {
1198         return (dnp->dn_kind == DT_NODE_INT && dnp->dn_value != 0 && (
1199             (dnp->dn_flags & DT_NF_SIGNED) == 0 || (int64_t)dnp->dn_value > 0));
1200 }

1202 int
1203 dt_node_is_actfunc(const dt_node_t *dnp)
1204 {
1205         return (dnp->dn_kind == DT_NODE_FUNC &&
1206             dnp->dn_ident->di_kind == DT_IDENT_ACTFUNC);
1207 }

1209 /*
1210  * The original rules for integer constant typing are described in K&R[A2.5.1].
1211  * However, since we support long long, we instead use the rules from ISO C99
1212  * clause 6.4.4.1 since that is where long longs are formally described.  The
1213  * rules require us to know whether the constant was specified in decimal or
1214  * in octal or hex, which we do by looking at our lexer's 'yyintdecimal' flag.
1215  * The type of an integer constant is the first of the corresponding list in
1216  * which its value can be represented:
1217  *
1218  * unsuffixed decimal:   int, long, long long
1219  * unsuffixed oct/hex:   int, unsigned int, long, unsigned long,
1220  *                       long long, unsigned long long
1221  * suffix [uU]:          unsigned int, unsigned long, unsigned long long
1222  * suffix [lL] decimal:  long, long long
1223  * suffix [lL] oct/hex:  long, unsigned long, long long, unsigned long long
1224  * suffix [uU][Ll]:      unsigned long, unsigned long long
1225  * suffix ll/LL decimal: long long
1226  * suffix ll/LL oct/hex: long long, unsigned long long
1227  * suffix [uU][ll/LL]:   unsigned long long
1228  *
1229  * Given that our lexer has already validated the suffixes by regexp matching,
1230  * there is an obvious way to concisely encode these rules: construct an array
1231  * of the types in the order int, unsigned int, long, unsigned long, long long,
1232  * unsigned long long.  Compute an integer array starting index based on the
1233  * suffix (e.g. none = 0, u = 1, ull = 5), and compute an increment based on
1234  * the specifier (dec/oct/hex) and suffix (u).  Then iterate from the starting
1235  * index to the end, advancing using the increment, and searching until we
1236  * find a limit that matches or we run out of choices (overflow).  To make it
```



```
1237  * even faster, we precompute the table of type information in dtrace_open().
1238  */
1239 dt_node_t *
1240 dt_node_int(uintmax_t value)
1241 {
1242         dt_node_t *dnp = dt_node_alloc(DT_NODE_INT);
1243         dtrace_hdl_t *dtp = yypcb->pcb_hdl;

1245         int n = (yyintdecimal | (yyintsuffix[0] == 'u')) + 1;
1246         int i = 0;

1248         const char *p;
1249         char c;

1251         dnp->dn_op = DT_TOK_INT;
1252         dnp->dn_value = value;

1254         for (p = yyintsuffix; (c = *p) != '\0'; p++) {
1255                 if (c == 'U' || c == 'u')
1256                         i += 1;
1257                 else if (c == 'L' || c == 'l')
1258                         i += 2;
1259         }

1261         for (; i < sizeof (dtp->dt_ints) / sizeof (dtp->dt_ints[0]); i += n) {
1262                 if (value <= dtp->dt_ints[i].did_limit) {
1263                         dt_node_type_assign(dnp,
1264                             dtp->dt_ints[i].did_ctfp,
1265                             dtp->dt_ints[i].did_type, B_FALSE);
 555                             dtp->dt_ints[i].did_type);

1267                         /*
1268                          * If a prefix character is present in macro text, add
1269                          * in the corresponding operator node (see dt_lex.l).
1270                          */
1271                         switch (yyintprefix) {
1272                         case '+':
1273                                 return (dt_node_op1(DT_TOK_IPOS, dnp));
1274                         case '-':
1275                                 return (dt_node_op1(DT_TOK_INEG, dnp));
1276                         default:
1277                                 return (dnp);
1278                         }
1279                 }
1280         }

1282         xyerror(D_INT_OFLOW, "integer constant 0x%llx cannot be represented "
1283             "in any built-in integral type\n", (u_longlong_t)value);
1284         /*NOTREACHED*/
1285         return (NULL);          /* keep gcc happy */
1286 }

1288 dt_node_t *
1289 dt_node_string(char *string)
1290 {
1291         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
1292         dt_node_t *dnp;

1294         if (string == NULL)
1295                 longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

1297         dnp = dt_node_alloc(DT_NODE_STRING);
1298         dnp->dn_op = DT_TOK_STRING;
1299         dnp->dn_string = string;
1300         dt_node_type_assign(dnp, DT_STR_CTFP(dtp), DT_STR_TYPE(dtp), B_FALSE);
 590         dt_node_type_assign(dnp, DT_STR_CTFP(dtp), DT_STR_TYPE(dtp));
```

```
1302         return (dnp);
1303 }
______unchanged_portion_omitted_

1344 /*
1345  * Create an empty node of type corresponding to the given declaration.
1346  * Explicit references to user types (C or D) are assigned the default
1347  * stability; references to other types are _dtrace_typattr (Private).
1348  */
1349 dt_node_t *
1350 dt_node_type(dt_decl_t *ddp)
1351 {
1352         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
1353         dtrace_typeinfo_t dtt;
1354         dt_node_t *dnp;
1355         char *name = NULL;
1356         int err;

1358         /*
1359          * If 'ddp' is NULL, we get a decl by popping the decl stack.  This
1360          * form of dt_node_type() is used by parameter rules in dt_grammar.y.
1361          */
1362         if (ddp == NULL)
1363                 ddp = dt_decl_pop_param(&name);

1365         err = dt_decl_type(ddp, &dtt);
1366         dt_decl_free(ddp);

1368         if (err != 0) {
1369                 free(name);
1370                 longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);
1371         }

1373         dnp = dt_node_alloc(DT_NODE_TYPE);
1374         dnp->dn_op = DT_TOK_IDENT;
1375         dnp->dn_string = name;

1377         dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type, dtt.dtt_flags);
 666         dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type);

1379         if (dtt.dtt_ctfp == dtp->dt_cdefs->dm_ctfp ||
1380             dtt.dtt_ctfp == dtp->dt_ddefs->dm_ctfp)
1381                 dt_node_attr_assign(dnp, _dtrace_defattr);
1382         else
1383                 dt_node_attr_assign(dnp, _dtrace_typattr);

1385         return (dnp);
1386 }
______unchanged_portion_omitted_

1405 /*
1406  * Instantiate a decl using the contents of the current declaration stack.  As
1407  * we do not currently permit decls to be initialized, this function currently
1408  * returns NULL and no parse node is created.  When this function is called,
1409  * the topmost scope's ds_ident pointer will be set to NULL (indicating no
1410  * init_declarator rule was matched) or will point to the identifier to use.
1411  */
1412 dt_node_t *
1413 dt_node_decl(void)
1414 {
1415         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
1416         dt_scope_t *dsp = &yypcb->pcb_dstack;
1417         dt_dclass_t class = dsp->ds_class;
1418         dt_decl_t *ddp = dt_decl_top();
```



```
1420         dt_module_t *dmp;
1421         dtrace_typeinfo_t dtt;
1422         ctf_id_t type;

1424         char n1[DT_TYPE_NAMELEN];
1425         char n2[DT_TYPE_NAMELEN];

1427         if (dt_decl_type(ddp, &dtt) != 0)
1428                 longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);

1430         /*
1431          * If we have no declaration identifier, then this is either a spurious
1432          * declaration of an intrinsic type (e.g. "extern int;") or declaration
1433          * or redeclaration of a struct, union, or enum type or tag.
1434          */
1435         if (dsp->ds_ident == NULL) {
1436                 if (ddp->dd_kind != CTF_K_STRUCT &&
1437                     ddp->dd_kind != CTF_K_UNION && ddp->dd_kind != CTF_K_ENUM)
1438                         xyerror(D_DECL_USELESS, "useless declaration\n");

1440                 dt_dprintf("type %s added as id %ld\n", dt_type_name(
1441                     ddp->dd_ctfp, ddp->dd_type, n1, sizeof (n1)), ddp->dd_type);

1443                 return (NULL);
1444         }

1446         if (strchr(dsp->ds_ident, '') != NULL) {
1447                 xyerror(D_DECL_SCOPE, "D scoping operator may not be used in "
1448                     "a declaration name (%s)\n", dsp->ds_ident);
1449         }

1451         /*
1452          * If we are nested inside of a C include file, add the declaration to
1453          * the C definition module; otherwise use the D definition module.
1454          */
1455         if (yypcb->pcb_idepth != 0)
1456                 dmp = dtp->dt_cdefs;
1457         else
1458                 dmp = dtp->dt_ddefs;

1460         /*
1461          * If we see a global or static declaration of a function prototype,
1462          * treat this as equivalent to a D extern declaration.
1463          */
1464         if (ctf_type_kind(dtt.dtt_ctfp, dtt.dtt_type) == CTF_K_FUNCTION &&
1465             (class == DT_DC_DEFAULT || class == DT_DC_STATIC))
1466                 class = DT_DC_EXTERN;

1468         switch (class) {
1469         case DT_DC_AUTO:
1470         case DT_DC_REGISTER:
1471         case DT_DC_STATIC:
1472                 xyerror(D_DECL_BADCLASS, "specified storage class not "
1473                     "appropriate in D\n");
1474                 /*NOTREACHED*/

1476         case DT_DC_EXTERN: {
1477                 dtrace_typeinfo_t ott;
1478                 dtrace_syminfo_t dts;
1479                 GElf_Sym sym;

1481                 int exists = dtrace_lookup_by_name(dtp,
1482                     dmp->dm_name, dsp->ds_ident, &sym, &dts) == 0;

1484                 if (exists && (dtrace_symbol_type(dtp, &sym, &dts, &ott) != 0 ||
1485                     ctf_type_cmp(dtt.dtt_ctfp, dtt.dtt_type,
```

```
1486                    ott.dtt_ctfp, ott.dtt_type) != 0)) {
1487                        xyerror(D_DECL_IDRED, "identifier redeclared: %s‘%s\n"
1488                            "\t current: %s\n\tprevious: %s\n",
1489                            dmp->dm_name, dsp->ds_ident,
1490                            dt_type_name(dtt.dtt_ctfp, dtt.dtt_type,
1491                            n1, sizeof (n1)),
1492                            dt_type_name(ott.dtt_ctfp, ott.dtt_type,
1493                            n2, sizeof (n2)));
1494                } else if (!exists && dt_module_extern(dtp, dmp,
1495                    dsp->ds_ident, &dtt) == NULL) {
1496                        xyerror(D_UNKNOWN,
1497                            "failed to extern %s: %s\n", dsp->ds_ident,
1498                            dtrace_errmsg(dtp, dtrace_errno(dtp)));
1499                } else {
1500                        dt_dprintf("extern %s‘%s type=<%s>\n",
1501                            dmp->dm_name, dsp->ds_ident,
1502                            dt_type_name(dtt.dtt_ctfp, dtt.dtt_type,
1503                            n1, sizeof (n1)));
1504                }
1505                break;
1506        }

1508        case DT_DC_TYPEDEF:
1509                if (dt_idstack_lookup(&yypcb->pcb_globals, dsp->ds_ident)) {
1510                        xyerror(D_DECL_IDRED, "global variable identifier "
1511                            "redeclared: %s\n", dsp->ds_ident);
1512                }

1514                if (ctf_lookup_by_name(dmp->dm_ctfp,
1515                    dsp->ds_ident) != CTF_ERR) {
1516                        xyerror(D_DECL_IDRED,
1517                            "typedef redeclared: %s\n", dsp->ds_ident);
1518                }

1520                /*
1521                 * If the source type for the typedef is not defined in the
1522                 * target container or its parent, copy the type to the target
1523                 * container and reset dtt_ctfp and dtt_type to the copy.
1524                 */
1525                if (dtt.dtt_ctfp != dmp->dm_ctfp &&
1526                    dtt.dtt_ctfp != ctf_parent_file(dmp->dm_ctfp)) {

1528                        dtt.dtt_type = ctf_add_type(dmp->dm_ctfp,
1529                            dtt.dtt_ctfp, dtt.dtt_type);
1530                        dtt.dtt_ctfp = dmp->dm_ctfp;

1532                        if (dtt.dtt_type == CTF_ERR ||
1533                            ctf_update(dtt.dtt_ctfp) == CTF_ERR) {
1534                                xyerror(D_UNKNOWN, "failed to copy typedef %s "
1535                                    "source type: %s\n", dsp->ds_ident,
1536                                    ctf_errmsg(ctf_errno(dtt.dtt_ctfp)));
1537                        }
1538                }

1540                type = ctf_add_typedef(dmp->dm_ctfp,
1541                    CTF_ADD_ROOT, dsp->ds_ident, dtt.dtt_type);

1543                if (type == CTF_ERR || ctf_update(dmp->dm_ctfp) == CTF_ERR) {
1544                        xyerror(D_UNKNOWN, "failed to typedef %s: %s\n",
1545                            dsp->ds_ident, ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1546                }

1548                dt_dprintf("typedef %s added as id %ld\n", dsp->ds_ident, type);
1549                break;

1551        default: {
```



```
1552                ctf_encoding_t cte;
1553                dt_idhash_t *dhp;
1554                dt_ident_t *idp;
1555                dt_node_t idn;
1556                int assc, idkind;
1557                uint_t id, kind;
1558                ushort_t idflags;

1560                switch (class) {
1561                case DT_DC_THIS:
1562                        dhp = yypcb->pcb_locals;
1563                        idflags = DT_IDFLG_LOCAL;
1564                        idp = dt_idhash_lookup(dhp, dsp->ds_ident);
1565                        break;
1566                case DT_DC_SELF:
1567                        dhp = dtp->dt_tls;
1568                        idflags = DT_IDFLG_TLS;
1569                        idp = dt_idhash_lookup(dhp, dsp->ds_ident);
1570                        break;
1571                default:
1572                        dhp = dtp->dt_globals;
1573                        idflags = 0;
1574                        idp = dt_idstack_lookup(
1575                            &yypcb->pcb_globals, dsp->ds_ident);
1576                        break;
1577                }

1579                if (ddp->dd_kind == CTF_K_ARRAY && ddp->dd_node == NULL) {
1580                        xyerror(D_DECL_ARRNULL,
1581                            "array declaration requires array dimension or "
1582                            "tuple signature: %s\n", dsp->ds_ident);
1583                }

1585                if (idp != NULL && idp->di_gen == 0) {
1586                        xyerror(D_DECL_IDRED, "built-in identifier "
1587                            "redeclared: %s\n", idp->di_name);
1588                }

1590                if (dtrace_lookup_by_type(dtp, DTRACE_OBJ_CDEFS,
1591                    dsp->ds_ident, NULL) == 0 ||
1592                    dtrace_lookup_by_type(dtp, DTRACE_OBJ_DDEFS,
1593                    dsp->ds_ident, NULL) == 0) {
1594                        xyerror(D_DECL_IDRED, "typedef identifier "
1595                            "redeclared: %s\n", dsp->ds_ident);
1596                }

1598                /*
1599                 * Cache some attributes of the decl to make the rest of this
1600                 * code simpler: if the decl is an array which is subscripted
1601                 * by a type rather than an integer, then it’s an associative
1602                 * array (assc).  We then expect to match either DT_IDENT_ARRAY
1603                 * for associative arrays or DT_IDENT_SCALAR for anything else.
1604                 */
1605                assc = ddp->dd_kind == CTF_K_ARRAY &&
1606                    ddp->dd_node->dn_kind == DT_NODE_TYPE;

1608                idkind = assc ? DT_IDENT_ARRAY : DT_IDENT_SCALAR;

1610                /*
1611                 * Create a fake dt_node_t on the stack so we can determine the
1612                 * type of any matching identifier by assigning to this node.
1613                 * If the pre-existing ident has its di_type set, propagate
1614                 * the type by hand so as not to trigger a prototype check for
1615                 * arrays (yet); otherwise we use dt_ident_cook() on the ident
1616                 * to ensure it is fully initialized before looking at it.
1617                 */
```

```
1618                bzero(&idn, sizeof (dt_node_t));

1620                if (idp != NULL && idp->di_type != CTF_ERR)
1621                        dt_node_type_assign(&idn, idp->di_ctfp, idp->di_type,
1622                            B_FALSE);
 910                        dt_node_type_assign(&idn, idp->di_ctfp, idp->di_type);
1623                else if (idp != NULL)
1624                        (void) dt_ident_cook(&idn, idp, NULL);

1626                if (assc) {
1627                        if (class == DT_DC_THIS) {
1628                                xyerror(D_DECL_LOCASSC, "associative arrays "
1629                                    "may not be declared as local variables:"
1630                                    " %s\n", dsp->ds_ident);
1631                        }

1633                        if (dt_decl_type(ddp->dd_next, &dtt) != 0)
1634                                longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);
1635                }

1637                if (idp != NULL && (idp->di_kind != idkind ||
1638                    ctf_type_cmp(dtt.dtt_ctfp, dtt.dtt_type,
1639                    idn.dn_ctfp, idn.dn_type) != 0)) {
1640                        xyerror(D_DECL_IDRED, "identifier redeclared: %s\n"
1641                            "\t current: %s %s\n\tprevious: %s %s\n",
1642                            dsp->ds_ident, dt_idkind_name(idkind),
1643                            dt_type_name(dtt.dtt_ctfp,
1644                            dtt.dtt_type, n1, sizeof (n1)),
1645                            dt_idkind_name(idp->di_kind),
1646                            dt_node_type_name(&idn, n2, sizeof (n2)));

1648                } else if (idp != NULL && assc) {
1649                        const dt_idsig_t *isp = idp->di_data;
1650                        dt_node_t *dnp = ddp->dd_node;
1651                        int argc = 0;

1653                        for (; dnp != NULL; dnp = dnp->dn_list, argc++) {
1654                                const dt_node_t *pnp = &isp->dis_args[argc];

1656                                if (argc >= isp->dis_argc)
1657                                        continue; /* tuple length mismatch */

1659                                if (ctf_type_cmp(dnp->dn_ctfp, dnp->dn_type,
1660                                    pnp->dn_ctfp, pnp->dn_type) == 0)
1661                                        continue;

1663                                xyerror(D_DECL_IDRED,
1664                                    "identifier redeclared: %s\n"
1665                                    "\t current: %s, key #%d of type %s\n"
1666                                    "\tprevious: %s, key #%d of type %s\n",
1667                                    dsp->ds_ident,
1668                                    dt_idkind_name(idkind), argc + 1,
1669                                    dt_node_type_name(dnp, n1, sizeof (n1)),
1670                                    dt_idkind_name(idp->di_kind), argc + 1,
1671                                    dt_node_type_name(pnp, n2, sizeof (n2)));
1672                        }

1674                        if (isp->dis_argc != argc) {
1675                                xyerror(D_DECL_IDRED,
1676                                    "identifier redeclared: %s\n"
1677                                    "\t current: %s of %s, tuple length %d\n"
1678                                    "\tprevious: %s of %s, tuple length %d\n",
1679                                    dsp->ds_ident, dt_idkind_name(idkind),
1680                                    dt_type_name(dtt.dtt_ctfp, dtt.dtt_type,
1681                                    n1, sizeof (n1)), argc,
1682                                    dt_idkind_name(idp->di_kind),
```



```
1683                                    dt_node_type_name(&idn, n2, sizeof (n2)),
1684                                    isp->dis_argc);
1685                        }

1687                } else if (idp == NULL) {
1688                        type = ctf_type_resolve(dtt.dtt_ctfp, dtt.dtt_type);
1689                        kind = ctf_type_kind(dtt.dtt_ctfp, type);

1691                        switch (kind) {
1692                        case CTF_K_INTEGER:
1693                                if (ctf_type_encoding(dtt.dtt_ctfp, type,
1694                                    &cte) == 0 && IS_VOID(cte)) {
1695                                        xyerror(D_DECL_VOIDOBJ, "cannot have "
1696                                            "void object: %s\n", dsp->ds_ident);
1697                                }
1698                                break;
1699                        case CTF_K_STRUCT:
1700                        case CTF_K_UNION:
1701                                if (ctf_type_size(dtt.dtt_ctfp, type) != 0)
1702                                        break; /* proceed to declaring */
1703                                /*FALLTHRU*/
1704                        case CTF_K_FORWARD:
1705                                xyerror(D_DECL_INCOMPLETE,
1706                                    "incomplete struct/union/enum %s: %s\n",
1707                                    dt_type_name(dtt.dtt_ctfp, dtt.dtt_type,
1708                                    n1, sizeof (n1)), dsp->ds_ident);
1709                                /*NOTREACHED*/
1710                        }

1712                        if (dt_idhash_nextid(dhp, &id) == -1) {
1713                                xyerror(D_ID_OFLOW, "cannot create %s: limit "
1714                                    "on number of %s variables exceeded\n",
1715                                    dsp->ds_ident, dt_idhash_name(dhp));
1716                        }

1718                        dt_dprintf("declare %s %s variable %s, id=%u\n",
1719                            dt_idhash_name(dhp), dt_idkind_name(idkind),
1720                            dsp->ds_ident, id);

1722                        idp = dt_idhash_insert(dhp, dsp->ds_ident, idkind,
1723                            idflags | DT_IDFLG_WRITE | DT_IDFLG_DECL, id,
1724                            _dtrace_defattr, 0, assc ? &dt_idops_assc :
1725                            &dt_idops_thaw, NULL, dtp->dt_gen);

1727                        if (idp == NULL)
1728                                longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

1730                        dt_ident_type_assign(idp, dtt.dtt_ctfp, dtt.dtt_type);

1732                        /*
1733                         * If we are declaring an associative array, use our
1734                         * fake parse node to cook the new assoc identifier.
1735                         * This will force the ident code to instantiate the
1736                         * array type signature corresponding to the list of
1737                         * types pointed to by ddp->dd_node.  We also reset
1738                         * the identifier's attributes based upon the result.
1739                         */
1740                        if (assc) {
1741                                idp->di_attr =
1742                                    dt_ident_cook(&idn, idp, &ddp->dd_node);
1743                        }
1744                }
1745        }

1747        } /* end of switch */
```

```
1749         free(dsp->ds_ident);
1750         dsp->ds_ident = NULL;

1752         return (NULL);
1753 }
```

_____unchanged_portion_omitted_

```
1791 /*
1792  * The offsetof() function is special because it takes a type name as an
1793  * argument.  It does not actually construct its own node; after looking up the
1794  * structure or union offset, we just return an integer node with the offset.
1795  */
1796 dt_node_t *
1797 dt_node_offsetof(dt_decl_t *ddp, char *s)
1798 {
1799         dtrace_typeinfo_t dtt;
1800         dt_node_t dn;
1801         char *name;
1802         int err;

1804         ctf_membinfo_t ctm;
1805         ctf_id_t type;
1806         uint_t kind;

1808         name = strdupa(s);
1809         free(s);

1811         err = dt_decl_type(ddp, &dtt);
1812         dt_decl_free(ddp);

1814         if (err != 0)
1815                 longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);

1817         type = ctf_type_resolve(dtt.dtt_ctfp, dtt.dtt_type);
1818         kind = ctf_type_kind(dtt.dtt_ctfp, type);

1820         if (kind != CTF_K_STRUCT && kind != CTF_K_UNION) {
1821                 xyerror(D_OFFSETOF_TYPE,
1822                     "offsetof operand must be a struct or union type\n");
1823         }

1825         if (ctf_member_info(dtt.dtt_ctfp, type, name, &ctm) == CTF_ERR) {
1826                 xyerror(D_UNKNOWN, "failed to determine offset of %s: %s\n",
1827                     name, ctf_errmsg(ctf_errno(dtt.dtt_ctfp)));
1828         }

1830         bzero(&dn, sizeof (dn));
1831         dt_node_type_assign(&dn, dtt.dtt_ctfp, ctm.ctm_type, B_FALSE);
1119         dt_node_type_assign(&dn, dtt.dtt_ctfp, ctm.ctm_type);

1833         if (dn.dn_flags & DT_NF_BITFIELD) {
1834                 xyerror(D_OFFSETOF_BITFIELD,
1835                     "cannot take offset of a bit-field: %s\n", name);
1836         }

1838         return (dt_node_int(ctm.ctm_offset / NBBY));
1839 }

1841 dt_node_t *
1842 dt_node_op1(int op, dt_node_t *cp)
1843 {
1844         dt_node_t *dnp;

1846         if (cp->dn_kind == DT_NODE_INT) {
1847                 switch (op) {
1848                 case DT_TOK_INEG:
```



```
1849                         /*
1850                          * If we're negating an unsigned integer, zero out any
1851                          * extra top bits to truncate the value to the size of
1852                          * the effective type determined by dt_node_int().
1853                          */
1854                         cp->dn_value = -cp->dn_value;
1855                         if (!(cp->dn_flags & DT_NF_SIGNED)) {
1856                                 cp->dn_value &= ~0ULL >>
1857                                     (64 - dt_node_type_size(cp) * NBBY);
1858                         }
1859                         /*FALLTHRU*/
1860                 case DT_TOK_IPOS:
1861                         return (cp);
1862                 case DT_TOK_BNEG:
1863                         cp->dn_value = ~cp->dn_value;
1864                         return (cp);
1865                 case DT_TOK_LNEG:
1866                         cp->dn_value = !cp->dn_value;
1867                         return (cp);
1868                 }
1869         }

1871         /*
1872          * If sizeof is applied to a type_name or string constant, we can
1873          * transform 'cp' into an integer constant in the node construction
1874          * pass so that it can then be used for arithmetic in this pass.
1875          */
1876         if (op == DT_TOK_SIZEOF &&
1877             (cp->dn_kind == DT_NODE_STRING || cp->dn_kind == DT_NODE_TYPE)) {
1878                 dtrace_hdl_t *dtp = yypcb->pcb_hdl;
1879                 size_t size = dt_node_type_size(cp);

1881                 if (size == 0) {
1882                         xyerror(D_SIZEOF_TYPE, "cannot apply sizeof to an "
1883                             "operand of unknown size\n");
1884                 }

1886                 dt_node_type_assign(cp, dtp->dt_ddefs->dm_ctfp,
1887                     ctf_lookup_by_name(dtp->dt_ddefs->dm_ctfp, "size_t"),
1888                     B_FALSE);
1175                     ctf_lookup_by_name(dtp->dt_ddefs->dm_ctfp, "size_t"));

1890                 cp->dn_kind = DT_NODE_INT;
1891                 cp->dn_op = DT_TOK_INT;
1892                 cp->dn_value = size;

1894                 return (cp);
1895         }

1897         dnp = dt_node_alloc(DT_NODE_OP1);
1898         assert(op <= USHRT_MAX);
1899         dnp->dn_op = (ushort_t)op;
1900         dnp->dn_child = cp;

1902         return (dnp);
1903 }
```

_____unchanged_portion_omitted_

```
1937 dt_node_t *
1938 dt_node_op2(int op, dt_node_t *lp, dt_node_t *rp)
1939 {
1940         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
1941         dt_node_t *dnp;

1943         /*
1944          * First we check for operations that are illegal -- namely those that
```

```
1945          * might result in integer division by zero, and abort if one is found.
1946          */
1947         if (rp->dn_kind == DT_NODE_INT && rp->dn_value == 0 &&
1948             (op == DT_TOK_MOD || op == DT_TOK_DIV ||
1949              op == DT_TOK_MOD_EQ || op == DT_TOK_DIV_EQ))
1950                 xyerror(D_DIV_ZERO, "expression contains division by zero\n");

1952         /*
1953          * If both children are immediate values, we can just perform inline
1954          * calculation and return a new immediate node with the result.
1955          */
1956         if (lp->dn_kind == DT_NODE_INT && rp->dn_kind == DT_NODE_INT) {
1957                 uintmax_t l = lp->dn_value;
1958                 uintmax_t r = rp->dn_value;

1960                 dnp = dt_node_int(0); /* allocate new integer node for result */

1962                 switch (op) {
1963                 case DT_TOK_LOR:
1964                         dnp->dn_value = l || r;
1965                         dt_node_type_assign(dnp,
1966                             DT_INT_CTFP(dtp), DT_INT_TYPE(dtp), B_FALSE);
1253                         DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
1967                         break;
1968                 case DT_TOK_LXOR:
1969                         dnp->dn_value = (l != 0) ^ (r != 0);
1970                         dt_node_type_assign(dnp,
1971                             DT_INT_CTFP(dtp), DT_INT_TYPE(dtp), B_FALSE);
1258                         DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
1972                         break;
1973                 case DT_TOK_LAND:
1974                         dnp->dn_value = l && r;
1975                         dt_node_type_assign(dnp,
1976                             DT_INT_CTFP(dtp), DT_INT_TYPE(dtp), B_FALSE);
1263                         DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
1977                         break;
1978                 case DT_TOK_BOR:
1979                         dnp->dn_value = l | r;
1980                         dt_node_promote(lp, rp, dnp);
1981                         break;
1982                 case DT_TOK_XOR:
1983                         dnp->dn_value = l ^ r;
1984                         dt_node_promote(lp, rp, dnp);
1985                         break;
1986                 case DT_TOK_BAND:
1987                         dnp->dn_value = l & r;
1988                         dt_node_promote(lp, rp, dnp);
1989                         break;
1990                 case DT_TOK_EQU:
1991                         dnp->dn_value = l == r;
1992                         dt_node_type_assign(dnp,
1993                             DT_INT_CTFP(dtp), DT_INT_TYPE(dtp), B_FALSE);
1280                         DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
1994                         break;
1995                 case DT_TOK_NEQ:
1996                         dnp->dn_value = l != r;
1997                         dt_node_type_assign(dnp,
1998                             DT_INT_CTFP(dtp), DT_INT_TYPE(dtp), B_FALSE);
1285                         DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
1999                         break;
2000                 case DT_TOK_LT:
2001                         dt_node_promote(lp, rp, dnp);
2002                         if (dnp->dn_flags & DT_NF_SIGNED)
2003                                 dnp->dn_value = (intmax_t)l < (intmax_t)r;
2004                         else
2005                                 dnp->dn_value = l < r;
```



```
2006                         dt_node_type_assign(dnp,
2007                             DT_INT_CTFP(dtp), DT_INT_TYPE(dtp), B_FALSE);
1294                         DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
2008                         break;
2009                 case DT_TOK_LE:
2010                         dt_node_promote(lp, rp, dnp);
2011                         if (dnp->dn_flags & DT_NF_SIGNED)
2012                                 dnp->dn_value = (intmax_t)l <= (intmax_t)r;
2013                         else
2014                                 dnp->dn_value = l <= r;
2015                         dt_node_type_assign(dnp,
2016                             DT_INT_CTFP(dtp), DT_INT_TYPE(dtp), B_FALSE);
1303                         DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
2017                         break;
2018                 case DT_TOK_GT:
2019                         dt_node_promote(lp, rp, dnp);
2020                         if (dnp->dn_flags & DT_NF_SIGNED)
2021                                 dnp->dn_value = (intmax_t)l > (intmax_t)r;
2022                         else
2023                                 dnp->dn_value = l > r;
2024                         dt_node_type_assign(dnp,
2025                             DT_INT_CTFP(dtp), DT_INT_TYPE(dtp), B_FALSE);
1312                         DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
2026                         break;
2027                 case DT_TOK_GE:
2028                         dt_node_promote(lp, rp, dnp);
2029                         if (dnp->dn_flags & DT_NF_SIGNED)
2030                                 dnp->dn_value = (intmax_t)l >= (intmax_t)r;
2031                         else
2032                                 dnp->dn_value = l >= r;
2033                         dt_node_type_assign(dnp,
2034                             DT_INT_CTFP(dtp), DT_INT_TYPE(dtp), B_FALSE);
1321                         DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
2035                         break;
2036                 case DT_TOK_LSH:
2037                         dnp->dn_value = l << r;
2038                         dt_node_type_propagate(lp, dnp);
2039                         dt_node_attr_assign(rp,
2040                             dt_attr_min(lp->dn_attr, rp->dn_attr));
2041                         break;
2042                 case DT_TOK_RSH:
2043                         dnp->dn_value = l >> r;
2044                         dt_node_type_propagate(lp, dnp);
2045                         dt_node_attr_assign(rp,
2046                             dt_attr_min(lp->dn_attr, rp->dn_attr));
2047                         break;
2048                 case DT_TOK_ADD:
2049                         dnp->dn_value = l + r;
2050                         dt_node_promote(lp, rp, dnp);
2051                         break;
2052                 case DT_TOK_SUB:
2053                         dnp->dn_value = l - r;
2054                         dt_node_promote(lp, rp, dnp);
2055                         break;
2056                 case DT_TOK_MUL:
2057                         dnp->dn_value = l * r;
2058                         dt_node_promote(lp, rp, dnp);
2059                         break;
2060                 case DT_TOK_DIV:
2061                         dt_node_promote(lp, rp, dnp);
2062                         if (dnp->dn_flags & DT_NF_SIGNED)
2063                                 dnp->dn_value = (intmax_t)l / (intmax_t)r;
2064                         else
2065                                 dnp->dn_value = l / r;
2066                         break;
2067                 case DT_TOK_MOD:
```

```
2068                        dt_node_promote(lp, rp, dnp);
2069                        if (dnp->dn_flags & DT_NF_SIGNED)
2070                                dnp->dn_value = (intmax_t)l % (intmax_t)r;
2071                        else
2072                                dnp->dn_value = l % r;
2073                        break;
2074                default:
2075                        dt_node_free(dnp);
2076                        dnp = NULL;
2077                }

2079                if (dnp != NULL) {
2080                        dt_node_free(lp);
2081                        dt_node_free(rp);
2082                        return (dnp);
2083                }
2084        }

2086        if (op == DT_TOK_LPAR && rp->dn_kind == DT_NODE_INT &&
2087            dt_node_is_integer(lp)) {
2088                dt_cast(lp, rp);
2089                dt_node_type_propagate(lp, rp);
2090                dt_node_attr_assign(rp, dt_attr_min(lp->dn_attr, rp->dn_attr));
2091                dt_node_free(lp);

2093                return (rp);
2094        }

2096        /*
2097         * If no immediate optimizations are available, create an new OP2 node
2098         * and glue the left and right children into place and return.
2099         */
2100        dnp = dt_node_alloc(DT_NODE_OP2);
2101        assert(op <= USHRT_MAX);
2102        dnp->dn_op = (ushort_t)op;
2103        dnp->dn_left = lp;
2104        dnp->dn_right = rp;

2106        return (dnp);
2107 }
_____unchanged_portion_omitted_

2217 dt_node_t *
2218 dt_node_inline(dt_node_t *expr)
2219 {
2220        dtrace_hdl_t *dtp = yypcb->pcb_hdl;
2221        dt_scope_t *dsp = &yypcb->pcb_dstack;
2222        dt_decl_t *ddp = dt_decl_top();

2224        char n[DT_TYPE_NAMELEN];
2225        dtrace_typeinfo_t dtt;

2227        dt_ident_t *idp, *rdp;
2228        dt_idnode_t *inp;
2229        dt_node_t *dnp;

2231        if (dt_decl_type(ddp, &dtt) != 0)
2232                longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);

2234        if (dsp->ds_class != DT_DC_DEFAULT) {
2235                xyerror(D_DECL_BADCLASS, "specified storage class not "
2236                    "appropriate for inline declaration\n");
2237        }

2239        if (dsp->ds_ident == NULL)
2240                xyerror(D_DECL_USELESS, "inline declaration requires a name\n");
```



```
2242        if ((idp = dt_idstack_lookup(
2243            &yypcb->pcb_globals, dsp->ds_ident)) != NULL) {
2244                xyerror(D_DECL_IDRED, "identifier redefined: %s\n\t current: "
2245                    "inline definition\n\tprevious: %s %s\n",
2246                    idp->di_name, dt_idkind_name(idp->di_kind),
2247                    (idp->di_flags & DT_IDFLG_INLINE) ? "inline" : "");
2248        }

2250        /*
2251         * If we are declaring an inlined array, verify that we have a tuple
2252         * signature, and then recompute 'dtt' as the array's value type.
2253         */
2254        if (ddp->dd_kind == CTF_K_ARRAY) {
2255                if (ddp->dd_node == NULL) {
2256                        xyerror(D_DECL_ARRNULL, "inline declaration requires "
2257                            "array tuple signature: %s\n", dsp->ds_ident);
2258                }

2260                if (ddp->dd_node->dn_kind != DT_NODE_TYPE) {
2261                        xyerror(D_DECL_ARRNULL, "inline declaration cannot be "
2262                            "of scalar array type: %s\n", dsp->ds_ident);
2263                }

2265                if (dt_decl_type(ddp->dd_next, &dtt) != 0)
2266                        longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);
2267        }

2269        /*
2270         * If the inline identifier is not defined, then create it with the
2271         * orphan flag set.  We do not insert the identifier into dt_globals
2272         * until we have successfully cooked the right-hand expression, below.
2273         */
2274        dnp = dt_node_alloc(DT_NODE_INLINE);
2275        dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type, B_FALSE);
1562        dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type);
2276        dt_node_attr_assign(dnp, _dtrace_defattr);

2278        if (dt_node_is_void(dnp)) {
2279                xyerror(D_DECL_VOIDOBJ,
2280                    "cannot declare void inline: %s\n", dsp->ds_ident);
2281        }

2283        if (ctf_type_kind(dnp->dn_ctfp, ctf_type_resolve(
2284            dnp->dn_ctfp, dnp->dn_type)) == CTF_K_FORWARD) {
2285                xyerror(D_DECL_INCOMPLETE,
2286                    "incomplete struct/union/enum %s: %s\n",
2287                    dt_node_type_name(dnp, n, sizeof (n)), dsp->ds_ident);
2288        }

2290        if ((inp = malloc(sizeof (dt_idnode_t))) == NULL)
2291                longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2293        bzero(inp, sizeof (dt_idnode_t));

2295        idp = dnp->dn_ident = dt_ident_create(dsp->ds_ident,
2296            ddp->dd_kind == CTF_K_ARRAY ? DT_IDENT_ARRAY : DT_IDENT_SCALAR,
2297            DT_IDFLG_INLINE | DT_IDFLG_REF | DT_IDFLG_DECL | DT_IDFLG_ORPHAN, 0,
2298            _dtrace_defattr, 0, &dt_idops_inline, inp, dtp->dt_gen);

2300        if (idp == NULL) {
2301                free(inp);
2302                longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
2303        }

2305        /*
```

```
2306          * If we're inlining an associative array, create a private identifier
2307          * hash containing the named parameters and store it in inp->din_hash.
2308          * We then push this hash on to the top of the pcb_globals stack.
2309          */
2310         if (ddp->dd_kind == CTF_K_ARRAY) {
2311                 dt_idnode_t *pinp;
2312                 dt_ident_t *pidp;
2313                 dt_node_t *pnp;
2314                 uint_t i = 0;

2316                 for (pnp = ddp->dd_node; pnp != NULL; pnp = pnp->dn_list)
2317                         i++; /* count up parameters for din_argv[] */

2319                 inp->din_hash = dt_idhash_create("inline args", NULL, 0, 0);
2320                 inp->din_argv = calloc(i, sizeof (dt_ident_t *));

2322                 if (inp->din_hash == NULL || inp->din_argv == NULL)
2323                         longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2325                 /*
2326                  * Create an identifier for each parameter as a scalar inline,
2327                  * and store it in din_hash and in position in din_argv[].  The
2328                  * parameter identifiers also use dt_idops_inline, but we leave
2329                  * the dt_idnode_t argument 'pinp' zeroed.  This will be filled
2330                  * in by the code generation pass with references to the args.
2331                  */
2332                 for (i = 0, pnp = ddp->dd_node;
2333                     pnp != NULL; pnp = pnp->dn_list, i++) {

2335                         if (pnp->dn_string == NULL)
2336                                 continue; /* ignore anonymous parameters */

2338                         if ((pinp = malloc(sizeof (dt_idnode_t))) == NULL)
2339                                 longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2341                         pidp = dt_idhash_insert(inp->din_hash, pnp->dn_string,
2342                             DT_IDENT_SCALAR, DT_IDFLG_DECL | DT_IDFLG_INLINE, 0,
2343                             _dtrace_defattr, 0, &dt_idops_inline,
2344                             pinp, dtp->dt_gen);

2346                         if (pidp == NULL) {
2347                                 free(pinp);
2348                                 longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);
2349                         }

2351                         inp->din_argv[i] = pidp;
2352                         bzero(pinp, sizeof (dt_idnode_t));
2353                         dt_ident_type_assign(pidp, pnp->dn_ctfp, pnp->dn_type);
2354                 }

2356                 dt_idstack_push(&yypcb->pcb_globals, inp->din_hash);
2357         }

2359         /*
2360          * Unlike most constructors, we need to explicitly cook the right-hand
2361          * side of the inline definition immediately to prevent recursion.  If
2362          * the right-hand side uses the inline itself, the cook will fail.
2363          */
2364         expr = dt_node_cook(expr, DT_IDFLG_REF);

2366         if (ddp->dd_kind == CTF_K_ARRAY)
2367                 dt_idstack_pop(&yypcb->pcb_globals, inp->din_hash);

2369         /*
2370          * Set the type, attributes, and flags for the inline.  If the right-
2371          * hand expression has an identifier, propagate its flags.  Then cook
```



```
2372          * the identifier to fully initialize it: if we're declaring an inline
2373          * associative array this will construct a type signature from 'ddp'.
2374          */
2375         if (dt_node_is_dynamic(expr))
2376                 rdp = dt_ident_resolve(expr->dn_ident);
2377         else if (expr->dn_kind == DT_NODE_VAR || expr->dn_kind == DT_NODE_SYM)
2378                 rdp = expr->dn_ident;
2379         else
2380                 rdp = NULL;

2382         if (rdp != NULL) {
2383                 idp->di_flags |= (rdp->di_flags &
2384                     (DT_IDFLG_WRITE | DT_IDFLG_USER | DT_IDFLG_PRIM));
2385         }

2387         idp->di_attr = dt_attr_min(_dtrace_defattr, expr->dn_attr);
2388         dt_ident_type_assign(idp, dtt.dtt_ctfp, dtt.dtt_type);
2389         (void) dt_ident_cook(dnp, idp, &ddp->dd_node);

2391         /*
2392          * Store the parse tree nodes for 'expr' inside of idp->di_data ('inp')
2393          * so that they will be preserved with this identifier.  Then pop the
2394          * inline declaration from the declaration stack and restore the lexer.
2395          */
2396         inp->din_list = yypcb->pcb_list;
2397         inp->din_root = expr;

2399         dt_decl_free(dt_decl_pop());
2400         yybegin(YYS_CLAUSE);

2402         /*
2403          * Finally, insert the inline identifier into dt_globals to make it
2404          * visible, and then cook 'dnp' to check its type against 'expr'.
2405          */
2406         dt_idhash_xinsert(dtp->dt_globals, idp);
2407         return (dt_node_cook(dnp, DT_IDFLG_REF));
2408 }

2410 dt_node_t *
2411 dt_node_member(dt_decl_t *ddp, char *name, dt_node_t *expr)
2412 {
2413         dtrace_typeinfo_t dtt;
2414         dt_node_t *dnp;
2415         int err;

2417         if (ddp != NULL) {
2418                 err = dt_decl_type(ddp, &dtt);
2419                 dt_decl_free(ddp);

2421                 if (err != 0)
2422                         longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);
2423         }

2425         dnp = dt_node_alloc(DT_NODE_MEMBER);
2426         dnp->dn_membname = name;
2427         dnp->dn_membexpr = expr;

2429         if (ddp != NULL)
2430                 dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type,
2431                     dtt.dtt_flags);
1717                 dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type);

2433         return (dnp);
2434 }

2436 dt_node_t *
```

```
2437 dt_node_xlator(dt_decl_t *ddp, dt_decl_t *sdp, char *name, dt_node_t *members)
2438 {
2439         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
2440         dtrace_typeinfo_t src, dst;
2441         dt_node_t sn, dn;
2442         dt_xlator_t *dxp;
2443         dt_node_t *dnp;
2444         int edst, esrc;
2445         uint_t kind;

2447         char n1[DT_TYPE_NAMELEN];
2448         char n2[DT_TYPE_NAMELEN];

2450         edst = dt_decl_type(ddp, &dst);
2451         dt_decl_free(ddp);

2453         esrc = dt_decl_type(sdp, &src);
2454         dt_decl_free(sdp);

2456         if (edst != 0 || esrc != 0) {
2457                 free(name);
2458                 longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);
2459         }

2461         bzero(&sn, sizeof (sn));
2462         dt_node_type_assign(&sn, src.dtt_ctfp, src.dtt_type, B_FALSE);
1748         dt_node_type_assign(&sn, src.dtt_ctfp, src.dtt_type);

2464         bzero(&dn, sizeof (dn));
2465         dt_node_type_assign(&dn, dst.dtt_ctfp, dst.dtt_type, B_FALSE);
1751         dt_node_type_assign(&dn, dst.dtt_ctfp, dst.dtt_type);

2467         if (dt_xlator_lookup(dtp, &sn, &dn, DT_XLATE_EXACT) != NULL) {
2468                 xyerror(D_XLATE_REDECL,
2469                     "translator from %s to %s has already been declared\n",
2470                     dt_node_type_name(&sn, n1, sizeof (n1)),
2471                     dt_node_type_name(&dn, n2, sizeof (n2)));
2472         }

2474         kind = ctf_type_kind(dst.dtt_ctfp,
2475             ctf_type_resolve(dst.dtt_ctfp, dst.dtt_type));

2477         if (kind == CTF_K_FORWARD) {
2478                 xyerror(D_XLATE_SOU, "incomplete struct/union/enum %s\n",
2479                     dt_type_name(dst.dtt_ctfp, dst.dtt_type, n1, sizeof (n1)));
2480         }

2482         if (kind != CTF_K_STRUCT && kind != CTF_K_UNION) {
2483                 xyerror(D_XLATE_SOU,
2484                     "translator output type must be a struct or union\n");
2485         }

2487         dxp = dt_xlator_create(dtp, &src, &dst, name, members, yypcb->pcb_list);
2488         yybegin(YYS_CLAUSE);
2489         free(name);

2491         if (dxp == NULL)
2492                 longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2494         dnp = dt_node_alloc(DT_NODE_XLATOR);
2495         dnp->dn_xlator = dxp;
2496         dnp->dn_members = members;

2498         return (dt_node_cook(dnp, DT_IDFLG_REF));
2499 }
```

*unchanged_portion_omitted_*


```
2619 /*
2620  * This function provides the underlying implementation of cooking an
2621  * identifier given its node, a hash of dynamic identifiers, an identifier
2622  * kind, and a boolean flag indicating whether we are allowed to instantiate
2623  * a new identifier if the string is not found.  This function is either
2624  * called from dt_cook_ident(), below, or directly by the various cooking
2625  * routines that are allowed to instantiate identifiers (e.g. op2 TOK_ASGN).
2626  */
2627 static void
2628 dt_xcook_ident(dt_node_t *dnp, dt_idhash_t *dhp, uint_t idkind, int create)
2629 {
2630         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
2631         const char *sname = dt_idhash_name(dhp);
2632         int uref = 0;

2634         dtrace_attribute_t attr = _dtrace_defattr;
2635         dt_ident_t *idp;
2636         dtrace_syminfo_t dts;
2637         GElf_Sym sym;

2639         const char *scope, *mark;
2640         uchar_t dnkind;
2641         char *name;

2643         /*
2644          * Look for scoping marks in the identifier.  If one is found, set our
2645          * scope to either DTRACE_OBJ_KMODS or UMODS or to the first part of
2646          * the string that specifies the scope using an explicit module name.
2647          * If two marks in a row are found, set 'uref' (user symbol reference).
2648          * Otherwise we set scope to DTRACE_OBJ_EXEC, indicating that normal
2649          * scope is desired and we should search the specified idhash.
2650          */
2651         if ((name = strrchr(dnp->dn_string, '`')) != NULL) {
2652                 if (name > dnp->dn_string && name[-1] == '`') {
2653                         uref++;
2654                         name[-1] = '\0';
2655                 }

2657                 if (name == dnp->dn_string + uref)
2658                         scope = uref ? DTRACE_OBJ_UMODS : DTRACE_OBJ_KMODS;
2659                 else
2660                         scope = dnp->dn_string;

2662                 *name++ = '\0'; /* leave name pointing after scoping mark */
2663                 dnkind = DT_NODE_VAR;

2665         } else if (idkind == DT_IDENT_AGG) {
2666                 scope = DTRACE_OBJ_EXEC;
2667                 name = dnp->dn_string + 1;
2668                 dnkind = DT_NODE_AGG;
2669         } else {
2670                 scope = DTRACE_OBJ_EXEC;
2671                 name = dnp->dn_string;
2672                 dnkind = DT_NODE_VAR;
2673         }

2675         /*
2676          * If create is set to false, and we fail our idhash lookup, preset
2677          * the errno code to EDT_NOVAR for our final error message below.
2678          * If we end up calling dtrace_lookup_by_name(), it will reset the
2679          * errno appropriately and that error will be reported instead.
2680          */
2681         (void) dt_set_errno(dtp, EDT_NOVAR);
2682         mark = uref ? "``" : "`";
```

```
2684         if (scope == DTRACE_OBJ_EXEC && (
2685             (dhp != dtp->dt_globals &&
2686             (idp = dt_idhash_lookup(dhp, name)) != NULL) ||
2687             (dhp == dtp->dt_globals &&
2688             (idp = dt_idstack_lookup(&yypcb->pcb_globals, name)) != NULL))) {
2689                 /*
2690                  * Check that we are referencing the ident in the manner that
2691                  * matches its type if this is a global lookup.  In the TLS or
2692                  * local case, we don't know how the ident will be used until
2693                  * the time operator -> is seen; more parsing is needed.
2694                  */
2695                 if (idp->di_kind != idkind && dhp == dtp->dt_globals) {
2696                         xyerror(D_IDENT_BADREF, "%s '%s' may not be referenced "
2697                             "as %s\n", dt_idkind_name(idp->di_kind),
2698                             idp->di_name, dt_idkind_name(idkind));
2699                 }

2701                 /*
2702                  * Arrays and aggregations are not cooked individually. They
2703                  * have dynamic types and must be referenced using operator [].
2704                  * This is handled explicitly by the code for DT_TOK_LBRAC.
2705                  */
2706                 if (idp->di_kind != DT_IDENT_ARRAY &&
2707                     idp->di_kind != DT_IDENT_AGG)
2708                         attr = dt_ident_cook(dnp, idp, NULL);
2709                 else {
2710                         dt_node_type_assign(dnp,
2711                             DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp), B_FALSE);
1997                             DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp));
2712                         attr = idp->di_attr;
2713                 }

2715                 free(dnp->dn_string);
2716                 dnp->dn_string = NULL;
2717                 dnp->dn_kind = dnkind;
2718                 dnp->dn_ident = idp;
2719                 dnp->dn_flags |= DT_NF_LVALUE;

2721                 if (idp->di_flags & DT_IDFLG_WRITE)
2722                         dnp->dn_flags |= DT_NF_WRITABLE;

2724                 dt_node_attr_assign(dnp, attr);

2726         } else if (dhp == dtp->dt_globals && scope != DTRACE_OBJ_EXEC &&
2727             dtrace_lookup_by_name(dtp, scope, name, &sym, &dts) == 0) {

2729                 dt_module_t *mp = dt_module_lookup_by_name(dtp, dts.dts_object);
2730                 int umod = (mp->dm_flags & DT_DM_KERNEL) == 0;
2731                 static const char *const kunames[] = { "kernel", "user" };

2733                 dtrace_typeinfo_t dtt;
2734                 dtrace_syminfo_t *sip;

2736                 if (uref ^ umod) {
2737                         xyerror(D_SYM_BADREF, "%s module '%s' symbol '%s' may "
2738                             "not be referenced as a %s symbol\n", kunames[umod],
2739                             dts.dts_object, dts.dts_name, kunames[uref]);
2740                 }

2742                 if (dtrace_symbol_type(dtp, &sym, &dts, &dtt) != 0) {
2743                         /*
2744                          * For now, we special-case EDT_DATAMODEL to clarify
2745                          * that mixed data models are not currently supported.
2746                          */
2747                         if (dtp->dt_errno == EDT_DATAMODEL) {
2748                                 xyerror(D_SYM_MODEL, "cannot use %s symbol "
```



```
2749                                     "%s%s%s in a %s D program\n",
2750                                     dt_module_modelname(mp),
2751                                     dts.dts_object, mark, dts.dts_name,
2752                                     dt_module_modelname(dtp->dt_ddefs));
2753                         }

2755                         xyerror(D_SYM_NOTYPES,
2756                             "no symbolic type information is available for "
2757                             "%s%s%s: %s\n", dts.dts_object, mark, dts.dts_name,
2758                             dtrace_errmsg(dtp, dtrace_errno(dtp)));
2759                 }

2761                 idp = dt_ident_create(name, DT_IDENT_SYMBOL, 0, 0,
2762                     _dtrace_symattr, 0, &dt_idops_thaw, NULL, dtp->dt_gen);

2764                 if (idp == NULL)
2765                         longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2767                 if (mp->dm_flags & DT_DM_PRIMARY)
2768                         idp->di_flags |= DT_IDFLG_PRIM;

2770                 idp->di_next = dtp->dt_externs;
2771                 dtp->dt_externs = idp;

2773                 if ((sip = malloc(sizeof (dtrace_syminfo_t))) == NULL)
2774                         longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2776                 bcopy(&dts, sip, sizeof (dtrace_syminfo_t));
2777                 idp->di_data = sip;
2778                 idp->di_ctfp = dtt.dtt_ctfp;
2779                 idp->di_type = dtt.dtt_type;

2781                 free(dnp->dn_string);
2782                 dnp->dn_string = NULL;
2783                 dnp->dn_kind = DT_NODE_SYM;
2784                 dnp->dn_ident = idp;
2785                 dnp->dn_flags |= DT_NF_LVALUE;

2787                 dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type,
2788                     dtt.dtt_flags);
2073                 dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type);
2789                 dt_node_attr_assign(dnp, _dtrace_symattr);

2791                 if (uref) {
2792                         idp->di_flags |= DT_IDFLG_USER;
2793                         dnp->dn_flags |= DT_NF_USERLAND;
2794                 }

2796         } else if (scope == DTRACE_OBJ_EXEC && create == B_TRUE) {
2797                 uint_t flags = DT_IDFLG_WRITE;
2798                 uint_t id;

2800                 if (dt_idhash_nextid(dhp, &id) == -1) {
2801                         xyerror(D_ID_OFLOW, "cannot create %s: limit on number "
2802                             "of %s variables exceeded\n", name, sname);
2803                 }

2805                 if (dhp == yypcb->pcb_locals)
2806                         flags |= DT_IDFLG_LOCAL;
2807                 else if (dhp == dtp->dt_tls)
2808                         flags |= DT_IDFLG_TLS;

2810                 dt_dprintf("create %s %s variable %s, id=%u\n",
2811                     sname, dt_idkind_name(idkind), name, id);

2813                 if (idkind == DT_IDENT_ARRAY || idkind == DT_IDENT_AGG) {
```

```
2814                        idp = dt_idhash_insert(dhp, name,
2815                            idkind, flags, id, _dtrace_defattr, 0,
2816                            &dt_idops_assc, NULL, dtp->dt_gen);
2817                } else {
2818                        idp = dt_idhash_insert(dhp, name,
2819                            idkind, flags, id, _dtrace_defattr, 0,
2820                            &dt_idops_thaw, NULL, dtp->dt_gen);
2821                }

2823                if (idp == NULL)
2824                        longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2826                /*
2827                 * Arrays and aggregations are not cooked individually. They
2828                 * have dynamic types and must be referenced using operator [].
2829                 * This is handled explicitly by the code for DT_TOK_LBRAC.
2830                 */
2831                if (idp->di_kind != DT_IDENT_ARRAY &&
2832                    idp->di_kind != DT_IDENT_AGG)
2833                        attr = dt_ident_cook(dnp, idp, NULL);
2834                else {
2835                        dt_node_type_assign(dnp,
2836                            DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp), B_FALSE);
2121                        DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp));
2837                        attr = idp->di_attr;
2838                }

2840                free(dnp->dn_string);
2841                dnp->dn_string = NULL;
2842                dnp->dn_kind = dnkind;
2843                dnp->dn_ident = idp;
2844                dnp->dn_flags |= DT_NF_LVALUE | DT_NF_WRITABLE;

2846                dt_node_attr_assign(dnp, attr);

2848        } else if (scope != DTRACE_OBJ_EXEC) {
2849                xyerror(D_IDENT_UNDEF, "failed to resolve %s%s%s: %s\n",
2850                    dnp->dn_string, mark, name,
2851                    dtrace_errmsg(dtp, dtrace_errno(dtp)));
2852        } else {
2853                xyerror(D_IDENT_UNDEF, "failed to resolve %s: %s\n",
2854                    dnp->dn_string, dtrace_errmsg(dtp, dtrace_errno(dtp)));
2855        }
2856 }
```

unchanged_portion_omitted_

```
2904 static dt_node_t *
2905 dt_cook_op1(dt_node_t *dnp, uint_t idflags)
2906 {
2907        dtrace_hdl_t *dtp = yypcb->pcb_hdl;
2908        dt_node_t *cp = dnp->dn_child;

2910        char n[DT_TYPE_NAMELEN];
2911        dtrace_typeinfo_t dtt;
2912        dt_ident_t *idp;

2914        ctf_encoding_t e;
2915        ctf_arinfo_t r;
2916        ctf_id_t type, base;
2917        uint_t kind;

2919        if (dnp->dn_op == DT_TOK_PREINC || dnp->dn_op == DT_TOK_POSTINC ||
2920            dnp->dn_op == DT_TOK_PREDEC || dnp->dn_op == DT_TOK_POSTDEC)
2921                idflags = DT_IDFLG_REF | DT_IDFLG_MOD;
2922        else
2923                idflags = DT_IDFLG_REF;
```



```
2925        /*
2926         * We allow the unary ++ and -- operators to instantiate new scalar
2927         * variables if applied to an identifier; otherwise just cook as usual.
2928         */
2929        if (cp->dn_kind == DT_NODE_IDENT && (idflags & DT_IDFLG_MOD))
2930                dt_xcook_ident(cp, dtp->dt_globals, DT_IDENT_SCALAR, B_TRUE);

2932        cp = dnp->dn_child = dt_node_cook(cp, 0); /* don't set idflags yet */

2934        if (cp->dn_kind == DT_NODE_VAR && dt_ident_unref(cp->dn_ident)) {
2935                if (dt_type_lookup("int64_t", &dtt) != 0)
2936                        xyerror(D_TYPE_ERR, "failed to lookup int64_t\n");

2938                dt_ident_type_assign(cp->dn_ident, dtt.dtt_ctfp, dtt.dtt_type);
2939                dt_node_type_assign(cp, dtt.dtt_ctfp, dtt.dtt_type,
2940                    dtt.dtt_flags);
2224                dt_node_type_assign(cp, dtt.dtt_ctfp, dtt.dtt_type);
2941        }

2943        if (cp->dn_kind == DT_NODE_VAR)
2944                cp->dn_ident->di_flags |= idflags;

2946        switch (dnp->dn_op) {
2947        case DT_TOK_DEREF:
2948                /*
2949                 * If the deref operator is applied to a translated pointer,
2950                 * we set our output type to the output of the translation.
2951                 */
2952                if ((idp = dt_node_resolve(cp, DT_IDENT_XLPTR)) != NULL) {
2953                        dt_xlator_t *dxp = idp->di_data;

2955                        dnp->dn_ident = &dxp->dx_souid;
2956                        dt_node_type_assign(dnp,
2957                            dnp->dn_ident->di_ctfp, dnp->dn_ident->di_type,
2958                            cp->dn_flags & DT_NF_USERLAND);
2241                        dnp->dn_ident->di_ctfp, dnp->dn_ident->di_type);
2959                        break;
2960                }

2962                type = ctf_type_resolve(cp->dn_ctfp, cp->dn_type);
2963                kind = ctf_type_kind(cp->dn_ctfp, type);

2965                if (kind == CTF_K_ARRAY) {
2966                        if (ctf_array_info(cp->dn_ctfp, type, &r) != 0) {
2967                                dtp->dt_ctferr = ctf_errno(cp->dn_ctfp);
2968                                longjmp(yypcb->pcb_jmpbuf, EDT_CTF);
2969                        } else
2970                                type = r.ctr_contents;
2971                } else if (kind == CTF_K_POINTER) {
2972                        type = ctf_type_reference(cp->dn_ctfp, type);
2973                } else {
2974                        xyerror(D_DEREF_NONPTR,
2975                            "cannot dereference non-pointer type\n");
2976                }

2978                dt_node_type_assign(dnp, cp->dn_ctfp, type,
2979                    cp->dn_flags & DT_NF_USERLAND);
2261                dt_node_type_assign(dnp, cp->dn_ctfp, type);
2980                base = ctf_type_resolve(cp->dn_ctfp, type);
2981                kind = ctf_type_kind(cp->dn_ctfp, base);

2983                if (kind == CTF_K_INTEGER && ctf_type_encoding(cp->dn_ctfp,
2984                    base, &e) == 0 && IS_VOID(e)) {
2985                        xyerror(D_DEREF_VOID,
2986                            "cannot dereference pointer to void\n");
```

```
2987                }

2989                if (kind == CTF_K_FUNCTION) {
2990                        xyerror(D_DEREF_FUNC,
2991                            "cannot dereference pointer to function\n");
2992                }

2994                if (kind != CTF_K_ARRAY || dt_node_is_string(dnp))
2995                        dnp->dn_flags |= DT_NF_LVALUE; /* see K&R[A7.4.3] */

2997                /*
2998                 * If we propagated the l-value bit and the child operand was
2999                 * a writable D variable or a binary operation of the form
3000                 * a + b where a is writable, then propagate the writable bit.
3001                 * This is necessary to permit assignments to scalar arrays,
3002                 * which are converted to expressions of the form *(a + i).
3003                 */
3004                if ((cp->dn_flags & DT_NF_WRITABLE) ||
3005                    (cp->dn_kind == DT_NODE_OP2 && cp->dn_op == DT_TOK_ADD &&
3006                    (cp->dn_left->dn_flags & DT_NF_WRITABLE)))
3007                        dnp->dn_flags |= DT_NF_WRITABLE;

3009                if ((cp->dn_flags & DT_NF_USERLAND) &&
3010                    (kind == CTF_K_POINTER || (dnp->dn_flags & DT_NF_REF)))
3011                        dnp->dn_flags |= DT_NF_USERLAND;
3012                break;

3014        case DT_TOK_IPOS:
3015        case DT_TOK_INEG:
3016                if (!dt_node_is_arith(cp)) {
3017                        xyerror(D_OP_ARITH, "operator %s requires an operand "
3018                            "of arithmetic type\n", opstr(dnp->dn_op));
3019                }
3020                dt_node_type_propagate(cp, dnp); /* see K&R[A7.4.4-6] */
3021                break;

3023        case DT_TOK_BNEG:
3024                if (!dt_node_is_integer(cp)) {
3025                        xyerror(D_OP_INT, "operator %s requires an operand of "
3026                            "integral type\n", opstr(dnp->dn_op));
3027                }
3028                dt_node_type_propagate(cp, dnp); /* see K&R[A7.4.4-6] */
3029                break;

3031        case DT_TOK_LNEG:
3032                if (!dt_node_is_scalar(cp)) {
3033                        xyerror(D_OP_SCALAR, "operator %s requires an operand "
3034                            "of scalar type\n", opstr(dnp->dn_op));
3035                }
3036                dt_node_type_assign(dnp, DT_INT_CTFP(dtp), DT_INT_TYPE(dtp),
3037                    B_FALSE);
2318                dt_node_type_assign(dnp, DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
3038                break;

3040        case DT_TOK_ADDROF:
3041                if (cp->dn_kind == DT_NODE_VAR || cp->dn_kind == DT_NODE_AGG) {
3042                        xyerror(D_ADDROF_VAR,
3043                            "cannot take address of dynamic variable\n");
3044                }

3046                if (dt_node_is_dynamic(cp)) {
3047                        xyerror(D_ADDROF_VAR,
3048                            "cannot take address of dynamic object\n");
3049                }

3051                if (!(cp->dn_flags & DT_NF_LVALUE)) {
```



```
3052                        xyerror(D_ADDROF_LVAL, /* see K&R[A7.4.2] */
3053                            "unacceptable operand for unary & operator\n");
3054                }

3056                if (cp->dn_flags & DT_NF_BITFIELD) {
3057                        xyerror(D_ADDROF_BITFIELD,
3058                            "cannot take address of bit-field\n");
3059                }

3061                dtt.dtt_object = NULL;
3062                dtt.dtt_ctfp = cp->dn_ctfp;
3063                dtt.dtt_type = cp->dn_type;

3065                if (dt_type_pointer(&dtt) == -1) {
3066                        xyerror(D_TYPE_ERR, "cannot find type for \"&\": %s*\n",
3067                            dt_node_type_name(cp, n, sizeof (n)));
3068                }

3070                dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type,
3071                    cp->dn_flags & DT_NF_USERLAND);
2351                dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type);

2353                if (cp->dn_flags & DT_NF_USERLAND)
2354                        dnp->dn_flags |= DT_NF_USERLAND;
3072                break;

3074        case DT_TOK_SIZEOF:
3075                if (cp->dn_flags & DT_NF_BITFIELD) {
3076                        xyerror(D_SIZEOF_BITFIELD,
3077                            "cannot apply sizeof to a bit-field\n");
3078                }

3080                if (dt_node_sizeof(cp) == 0) {
3081                        xyerror(D_SIZEOF_TYPE, "cannot apply sizeof to an "
3082                            "operand of unknown size\n");
3083                }

3085                dt_node_type_assign(dnp, dtp->dt_ddefs->dm_ctfp,
3086                    ctf_lookup_by_name(dtp->dt_ddefs->dm_ctfp, "size_t"),
3087                    B_FALSE);
2369                    ctf_lookup_by_name(dtp->dt_ddefs->dm_ctfp, "size_t"));
3088                break;

3090        case DT_TOK_STRINGOF:
3091                if (!dt_node_is_scalar(cp) && !dt_node_is_pointer(cp) &&
3092                    !dt_node_is_strcompat(cp)) {
3093                        xyerror(D_STRINGOF_TYPE,
3094                            "cannot apply stringof to a value of type %s\n",
3095                            dt_node_type_name(cp, n, sizeof (n)));
3096                }
3097                dt_node_type_assign(dnp, DT_STR_CTFP(dtp), DT_STR_TYPE(dtp),
3098                    cp->dn_flags & DT_NF_USERLAND);
2379                dt_node_type_assign(dnp, DT_STR_CTFP(dtp), DT_STR_TYPE(dtp));
3099                break;

3101        case DT_TOK_PREINC:
3102        case DT_TOK_POSTINC:
3103        case DT_TOK_PREDEC:
3104        case DT_TOK_POSTDEC:
3105                if (dt_node_is_scalar(cp) == 0) {
3106                        xyerror(D_OP_SCALAR, "operator %s requires operand of "
3107                            "scalar type\n", opstr(dnp->dn_op));
3108                }

3110                if (dt_node_is_vfptr(cp)) {
3111                        xyerror(D_OP_VFPTR, "operator %s requires an operand "
```

```
3112                                "of known size\n", opstr(dnp->dn_op));
3113                }

3115                if (!(cp->dn_flags & DT_NF_LVALUE)) {
3116                        xyerror(D_OP_LVAL, "operator %s requires modifiable "
3117                            "lvalue as an operand\n", opstr(dnp->dn_op));
3118                }

3120                if (!(cp->dn_flags & DT_NF_WRITABLE)) {
3121                        xyerror(D_OP_WRITE, "operator %s can only be applied "
3122                            "to a writable variable\n", opstr(dnp->dn_op));
3123                }

3125                dt_node_type_propagate(cp, dnp); /* see K&R[A7.4.1] */
3126                break;

3128        default:
3129                xyerror(D_UNKNOWN, "invalid unary op %s\n", opstr(dnp->dn_op));
3130        }

3132        dt_node_attr_assign(dnp, cp->dn_attr);
3133        return (dnp);
3134 }
```

unchanged_portion_omitted_

```
3161 static dt_node_t *
3162 dt_cook_op2(dt_node_t *dnp, uint_t idflags)
3163 {
3164        dtrace_hdl_t *dtp = yypcb->pcb_hdl;
3165        dt_node_t *lp = dnp->dn_left;
3166        dt_node_t *rp = dnp->dn_right;
3167        int op = dnp->dn_op;

3169        ctf_membinfo_t m;
3170        ctf_file_t *ctfp;
3171        ctf_id_t type;
3172        int kind, val, uref;
3173        dt_ident_t *idp;

3175        char n1[DT_TYPE_NAMELEN];
3176        char n2[DT_TYPE_NAMELEN];

3178        /*
3179         * The expression E1[E2] is identical by definition to *((E1)+(E2)) so
3180         * we convert "[" to "+" and glue on "*" at the end (see K&R[A7.3.1])
3181         * unless the left-hand side is an untyped D scalar, associative array,
3182         * or aggregation.  In these cases, we proceed to case DT_TOK_LBRAC and
3183         * handle associative array and aggregation references there.
3184         */
3185        if (op == DT_TOK_LBRAC) {
3186                if (lp->dn_kind == DT_NODE_IDENT) {
3187                        dt_idhash_t *dhp;
3188                        uint_t idkind;

3190                        if (lp->dn_op == DT_TOK_AGG) {
3191                                dhp = dtp->dt_aggs;
3192                                idp = dt_idhash_lookup(dhp, lp->dn_string + 1);
3193                                idkind = DT_IDENT_AGG;
3194                        } else {
3195                                dhp = dtp->dt_globals;
3196                                idp = dt_idstack_lookup(
3197                                    &yypcb->pcb_globals, lp->dn_string);
3198                                idkind = DT_IDENT_ARRAY;
3199                        }

3201                        if (idp == NULL || dt_ident_unref(idp))
```



```
3202                                dt_xcook_ident(lp, dhp, idkind, B_TRUE);
3203                        else
3204                                dt_xcook_ident(lp, dhp, idp->di_kind, B_FALSE);
3205                } else
3206                        lp = dnp->dn_left = dt_node_cook(lp, 0);

3208                /*
3209                 * Switch op to '+' for *(E1 + E2) array mode in these cases:
3210                 * (a) lp is a DT_IDENT_ARRAY variable that has already been
3211                 *     referenced using [] notation (dn_args != NULL).
3212                 * (b) lp is a non-ARRAY variable that has already been given
3213                 *     a type by assignment or declaration (!dt_ident_unref())
3214                 * (c) lp is neither a variable nor an aggregation
3215                 */
3216                if (lp->dn_kind == DT_NODE_VAR) {
3217                        if (lp->dn_ident->di_kind == DT_IDENT_ARRAY) {
3218                                if (lp->dn_args != NULL)
3219                                        op = DT_TOK_ADD;
3220                        } else if (!dt_ident_unref(lp->dn_ident))
3221                                op = DT_TOK_ADD;
3222                } else if (lp->dn_kind != DT_NODE_AGG)
3223                        op = DT_TOK_ADD;
3224        }

3226        switch (op) {
3227        case DT_TOK_BAND:
3228        case DT_TOK_XOR:
3229        case DT_TOK_BOR:
3230                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3231                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3233                if (!dt_node_is_integer(lp) || !dt_node_is_integer(rp)) {
3234                        xyerror(D_OP_INT, "operator %s requires operands of "
3235                            "integral type\n", opstr(op));
3236                }

3238                dt_node_promote(lp, rp, dnp); /* see K&R[A7.11-13] */
3239                break;

3241        case DT_TOK_LSH:
3242        case DT_TOK_RSH:
3243                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3244                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3246                if (!dt_node_is_integer(lp) || !dt_node_is_integer(rp)) {
3247                        xyerror(D_OP_INT, "operator %s requires operands of "
3248                            "integral type\n", opstr(op));
3249                }

3251                dt_node_type_propagate(lp, dnp); /* see K&R[A7.8] */
3252                dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));
3253                break;

3255        case DT_TOK_MOD:
3256                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3257                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3259                if (!dt_node_is_integer(lp) || !dt_node_is_integer(rp)) {
3260                        xyerror(D_OP_INT, "operator %s requires operands of "
3261                            "integral type\n", opstr(op));
3262                }

3264                dt_node_promote(lp, rp, dnp); /* see K&R[A7.6] */
3265                break;

3267        case DT_TOK_MUL:
```

```
3268         case DT_TOK_DIV:
3269                 lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3270                 rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3272                 if (!dt_node_is_arith(lp) || !dt_node_is_arith(rp)) {
3273                         xyerror(D_OP_ARITH, "operator %s requires operands of "
3274                             "arithmetic type\n", opstr(op));
3275                 }

3277                 dt_node_promote(lp, rp, dnp); /* see K&R[A7.6] */
3278                 break;

3280         case DT_TOK_LAND:
3281         case DT_TOK_LXOR:
3282         case DT_TOK_LOR:
3283                 lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3284                 rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3286                 if (!dt_node_is_scalar(lp) || !dt_node_is_scalar(rp)) {
3287                         xyerror(D_OP_SCALAR, "operator %s requires operands "
3288                             "of scalar type\n", opstr(op));
3289                 }

3291                 dt_node_type_assign(dnp, DT_INT_CTFP(dtp), DT_INT_TYPE(dtp),
3292                     B_FALSE);
2572                 dt_node_type_assign(dnp, DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
3293                 dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));
3294                 break;

3296         case DT_TOK_LT:
3297         case DT_TOK_LE:
3298         case DT_TOK_GT:
3299         case DT_TOK_GE:
3300         case DT_TOK_EQU:
3301         case DT_TOK_NEQ:
3302                 /*
3303                  * The D comparison operators provide the ability to transform
3304                  * a right-hand identifier into a corresponding enum tag value
3305                  * if the left-hand side is an enum type.  To do this, we cook
3306                  * the left-hand side, and then see if the right-hand side is
3307                  * an unscoped identifier defined in the enum.  If so, we
3308                  * convert into an integer constant node with the tag's value.
3309                  */
3310                 lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);

3312                 kind = ctf_type_kind(lp->dn_ctfp,
3313                     ctf_type_resolve(lp->dn_ctfp, lp->dn_type));

3315                 if (kind == CTF_K_ENUM && rp->dn_kind == DT_NODE_IDENT &&
3316                     strchr(rp->dn_string, '`') == NULL && ctf_enum_value(
3317                     lp->dn_ctfp, lp->dn_type, rp->dn_string, &val) == 0) {

3319                         if ((idp = dt_idstack_lookup(&yypcb->pcb_globals,
3320                             rp->dn_string)) != NULL) {
3321                                 xyerror(D_IDENT_AMBIG,
3322                                     "ambiguous use of operator %s: %s is "
3323                                     "both a %s enum tag and a global %s\n",
3324                                     opstr(op), rp->dn_string,
3325                                     dt_node_type_name(lp, n1, sizeof (n1)),
3326                                     dt_idkind_name(idp->di_kind));
3327                         }

3329                         free(rp->dn_string);
3330                         rp->dn_string = NULL;
3331                         rp->dn_kind = DT_NODE_INT;
3332                         rp->dn_flags |= DT_NF_COOKED;
```



```
3333                         rp->dn_op = DT_TOK_INT;
3334                         rp->dn_value = (intmax_t)val;

3336                         dt_node_type_assign(rp, lp->dn_ctfp, lp->dn_type,
3337                             B_FALSE);
2616                         dt_node_type_assign(rp, lp->dn_ctfp, lp->dn_type);
3338                         dt_node_attr_assign(rp, _dtrace_symattr);
3339                 }

3341                 rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3343                 /*
3344                  * The rules for type checking for the relational operators are
3345                  * described in the ANSI-C spec (see K&R[A7.9-10]).  We perform
3346                  * the various tests in order from least to most expensive.  We
3347                  * also allow derived strings to be compared as a first-class
3348                  * type (resulting in a strcmp(3C)-style comparison), and we
3349                  * slightly relax the A7.9 rules to permit void pointer
3350                  * comparisons as in A7.10.  Our users won't be confused by
3351                  * this since they understand pointers are just numbers, and
3352                  * relaxing this constraint simplifies the implementation.
3353                  */
3354                 if (ctf_type_compat(lp->dn_ctfp, lp->dn_type,
3355                     rp->dn_ctfp, rp->dn_type))
3356                         /*EMPTY*/;
3357                 else if (dt_node_is_integer(lp) && dt_node_is_integer(rp))
3358                         /*EMPTY*/;
3359                 else if (dt_node_is_strcompat(lp) && dt_node_is_strcompat(rp) &&
3360                     (dt_node_is_string(lp) || dt_node_is_string(rp)))
3361                         /*EMPTY*/;
3362                 else if (dt_node_is_ptrcompat(lp, rp, NULL, NULL) == 0) {
3363                         xyerror(D_OP_INCOMPAT, "operands have "
3364                             "incompatible types: \"%s\" %s \"%s\"\n",
3365                             dt_node_type_name(lp, n1, sizeof (n1)), opstr(op),
3366                             dt_node_type_name(rp, n2, sizeof (n2)));
3367                 }

3369                 dt_node_type_assign(dnp, DT_INT_CTFP(dtp), DT_INT_TYPE(dtp),
3370                     B_FALSE);
2648                 dt_node_type_assign(dnp, DT_INT_CTFP(dtp), DT_INT_TYPE(dtp));
3371                 dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));
3372                 break;

3374         case DT_TOK_ADD:
3375         case DT_TOK_SUB: {
3376                 /*
3377                  * The rules for type checking for the additive operators are
3378                  * described in the ANSI-C spec (see K&R[A7.7]).  Pointers and
3379                  * integers may be manipulated according to specific rules.  In
3380                  * these cases D permits strings to be treated as pointers.
3381                  */
3382                 int lp_is_ptr, lp_is_int, rp_is_ptr, rp_is_int;

3384                 lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3385                 rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3387                 lp_is_ptr = dt_node_is_string(lp) ||
3388                     (dt_node_is_pointer(lp) && !dt_node_is_vfptr(lp));
3389                 lp_is_int = dt_node_is_integer(lp);

3391                 rp_is_ptr = dt_node_is_string(rp) ||
3392                     (dt_node_is_pointer(rp) && !dt_node_is_vfptr(rp));
3393                 rp_is_int = dt_node_is_integer(rp);

3395                 if (lp_is_int && rp_is_int) {
3396                         dt_type_promote(lp, rp, &ctfp, &type);
```

```
3397                    uref = 0;
3398            } else if (lp_is_ptr && rp_is_int) {
3399                    ctfp = lp->dn_ctfp;
3400                    type = lp->dn_type;
3401                    uref = lp->dn_flags & DT_NF_USERLAND;
3402            } else if (lp_is_int && rp_is_ptr && op == DT_TOK_ADD) {
3403                    ctfp = rp->dn_ctfp;
3404                    type = rp->dn_type;
3405                    uref = rp->dn_flags & DT_NF_USERLAND;
3406            } else if (lp_is_ptr && rp_is_ptr && op == DT_TOK_SUB &&
3407                dt_node_is_ptrcompat(lp, rp, NULL, NULL)) {
3408                    ctfp = dtp->dt_ddefs->dm_ctfp;
3409                    type = ctf_lookup_by_name(ctfp, "ptrdiff_t");
3410                    uref = 0;
3411            } else {
3412                    xyerror(D_OP_INCOMPAT, "operands have incompatible "
3413                        "types: \"%s\" %s \"%s\"\n",
3414                        dt_node_type_name(lp, n1, sizeof (n1)), opstr(op),
3415                        dt_node_type_name(rp, n2, sizeof (n2)));
3416            }

3418            dt_node_type_assign(dnp, ctfp, type, B_FALSE);
2696            dt_node_type_assign(dnp, ctfp, type);
3419            dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));

3421            if (uref)
3422                    dnp->dn_flags |= DT_NF_USERLAND;
3423            break;
3424    }

3426    case DT_TOK_OR_EQ:
3427    case DT_TOK_XOR_EQ:
3428    case DT_TOK_AND_EQ:
3429    case DT_TOK_LSH_EQ:
3430    case DT_TOK_RSH_EQ:
3431    case DT_TOK_MOD_EQ:
3432            if (lp->dn_kind == DT_NODE_IDENT) {
3433                    dt_xcook_ident(lp, dtp->dt_globals,
3434                        DT_IDENT_SCALAR, B_TRUE);
3435            }

3437            lp = dnp->dn_left =
3438                dt_node_cook(lp, DT_IDFLG_REF | DT_IDFLG_MOD);

3440            rp = dnp->dn_right =
3441                dt_node_cook(rp, DT_IDFLG_REF | DT_IDFLG_MOD);

3443            if (!dt_node_is_integer(lp) || !dt_node_is_integer(rp)) {
3444                    xyerror(D_OP_INT, "operator %s requires operands of "
3445                        "integral type\n", opstr(op));
3446            }
3447            goto asgn_common;

3449    case DT_TOK_MUL_EQ:
3450    case DT_TOK_DIV_EQ:
3451            if (lp->dn_kind == DT_NODE_IDENT) {
3452                    dt_xcook_ident(lp, dtp->dt_globals,
3453                        DT_IDENT_SCALAR, B_TRUE);
3454            }

3456            lp = dnp->dn_left =
3457                dt_node_cook(lp, DT_IDFLG_REF | DT_IDFLG_MOD);

3459            rp = dnp->dn_right =
3460                dt_node_cook(rp, DT_IDFLG_REF | DT_IDFLG_MOD);
```



```
3462            if (!dt_node_is_arith(lp) || !dt_node_is_arith(rp)) {
3463                    xyerror(D_OP_ARITH, "operator %s requires operands of "
3464                        "arithmetic type\n", opstr(op));
3465            }
3466            goto asgn_common;

3468    case DT_TOK_ASGN:
3469            /*
3470             * If the left-hand side is an identifier, attempt to resolve
3471             * it as either an aggregation or scalar variable.  We pass
3472             * B_TRUE to dt_xcook_ident to indicate that a new variable can
3473             * be created if no matching variable exists in the namespace.
3474             */
3475            if (lp->dn_kind == DT_NODE_IDENT) {
3476                    if (lp->dn_op == DT_TOK_AGG) {
3477                            dt_xcook_ident(lp, dtp->dt_aggs,
3478                                DT_IDENT_AGG, B_TRUE);
3479                    } else {
3480                            dt_xcook_ident(lp, dtp->dt_globals,
3481                                DT_IDENT_SCALAR, B_TRUE);
3482                    }
3483            }

3485            lp = dnp->dn_left = dt_node_cook(lp, 0); /* don't set mod yet */
3486            rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3488            /*
3489             * If the left-hand side is an aggregation, verify that we are
3490             * assigning it the result of an aggregating function.  Once
3491             * we've done so, hide the func node in the aggregation and
3492             * return the aggregation itself up to the parse tree parent.
3493             * This transformation is legal since the assigned function
3494             * cannot change identity across disjoint cooking passes and
3495             * the argument list subtree is retained for later cooking.
3496             */
3497            if (lp->dn_kind == DT_NODE_AGG) {
3498                    const char *aname = lp->dn_ident->di_name;
3499                    dt_ident_t *oid = lp->dn_ident->di_iarg;

3501                    if (rp->dn_kind != DT_NODE_FUNC ||
3502                        rp->dn_ident->di_kind != DT_IDENT_AGGFUNC) {
3503                            xyerror(D_AGG_FUNC,
3504                                "@%s must be assigned the result of "
3505                                "an aggregating function\n", aname);
3506                    }

3508                    if (oid != NULL && oid != rp->dn_ident) {
3509                            xyerror(D_AGG_REDEF,
3510                                "aggregation redefined: @%s\n\t "
3511                                "current: @%s = %s( )\n\tprevious: @%s = "
3512                                "%s( ) : line %d\n", aname, aname,
3513                                rp->dn_ident->di_name, aname, oid->di_name,
3514                                lp->dn_ident->di_lineno);
3515                    } else if (oid == NULL)
3516                            lp->dn_ident->di_iarg = rp->dn_ident;

3518                    /*
3519                     * Do not allow multiple aggregation assignments in a
3520                     * single statement, e.g. (@a = count()) = count();
3521                     * We produce a message as if the result of aggregating
3522                     * function does not propagate DT_NF_LVALUE.
3523                     */
3524                    if (lp->dn_aggfun != NULL) {
3525                            xyerror(D_OP_LVAL, "operator = requires "
3526                                "modifiable lvalue as an operand\n");
3527                    }
```

```
3529                    lp->dn_aggfun = rp;
3530                    lp = dt_node_cook(lp, DT_IDFLG_MOD);

3532                    dnp->dn_left = dnp->dn_right = NULL;
3533                    dt_node_free(dnp);

3535                    return (lp);
3536            }

3538            /*
3539             * If the right-hand side is a dynamic variable that is the
3540             * output of a translator, our result is the translated type.
3541             */
3542            if ((idp = dt_node_resolve(rp, DT_IDENT_XLSOU)) != NULL) {
3543                    ctfp = idp->di_ctfp;
3544                    type = idp->di_type;
3545                    uref = idp->di_flags & DT_IDFLG_USER;
3546            } else {
3547                    ctfp = rp->dn_ctfp;
3548                    type = rp->dn_type;
3549                    uref = rp->dn_flags & DT_NF_USERLAND;
3550            }

3552            /*
3553             * If the left-hand side of an assignment statement is a virgin
3554             * variable created by this compilation pass, reset the type of
3555             * this variable to the type of the right-hand side.
3556             */
3557            if (lp->dn_kind == DT_NODE_VAR &&
3558                dt_ident_unref(lp->dn_ident)) {
3559                    dt_node_type_assign(lp, ctfp, type, B_FALSE);
2837                    dt_node_type_assign(lp, ctfp, type);
3560                    dt_ident_type_assign(lp->dn_ident, ctfp, type);

3562                    if (uref) {
3563                            lp->dn_flags |= DT_NF_USERLAND;
3564                            lp->dn_ident->di_flags |= DT_IDFLG_USER;
3565                    }
3566            }

3568            if (lp->dn_kind == DT_NODE_VAR)
3569                    lp->dn_ident->di_flags |= DT_IDFLG_MOD;

3571            /*
3572             * The rules for type checking for the assignment operators are
3573             * described in the ANSI-C spec (see K&R[A7.17]).  We share
3574             * most of this code with the argument list checking code.
3575             */
3576            if (!dt_node_is_string(lp)) {
3577                    kind = ctf_type_kind(lp->dn_ctfp,
3578                        ctf_type_resolve(lp->dn_ctfp, lp->dn_type));

3580                    if (kind == CTF_K_ARRAY || kind == CTF_K_FUNCTION) {
3581                            xyerror(D_OP_ARRFUN, "operator %s may not be "
3582                                "applied to operand of type \"%s\"\n",
3583                                opstr(op),
3584                                dt_node_type_name(lp, n1, sizeof (n1)));
3585                    }
3586            }

3588            if (idp != NULL && idp->di_kind == DT_IDENT_XLSOU &&
3589                ctf_type_compat(lp->dn_ctfp, lp->dn_type, ctfp, type))
3590                    goto asgn_common;

3592            if (dt_node_is_argcompat(lp, rp))
```



```
3593                    goto asgn_common;

3595            xyerror(D_OP_INCOMPAT,
3596                "operands have incompatible types: \"%s\" %s \"%s\"\n",
3597                dt_node_type_name(lp, n1, sizeof (n1)), opstr(op),
3598                dt_node_type_name(rp, n2, sizeof (n2)));
3599            /*NOTREACHED*/

3601        case DT_TOK_ADD_EQ:
3602        case DT_TOK_SUB_EQ:
3603            if (lp->dn_kind == DT_NODE_IDENT) {
3604                    dt_xcook_ident(lp, dtp->dt_globals,
3605                        DT_IDENT_SCALAR, B_TRUE);
3606            }

3608            lp = dnp->dn_left =
3609                dt_node_cook(lp, DT_IDFLG_REF | DT_IDFLG_MOD);

3611            rp = dnp->dn_right =
3612                dt_node_cook(rp, DT_IDFLG_REF | DT_IDFLG_MOD);

3614            if (dt_node_is_string(lp) || dt_node_is_string(rp)) {
3615                    xyerror(D_OP_INCOMPAT, "operands have "
3616                        "incompatible types: \"%s\" %s \"%s\"\n",
3617                        dt_node_type_name(lp, n1, sizeof (n1)), opstr(op),
3618                        dt_node_type_name(rp, n2, sizeof (n2)));
3619            }

3621            /*
3622             * The rules for type checking for the assignment operators are
3623             * described in the ANSI-C spec (see K&R[A7.17]).  To these
3624             * rules we add that only writable D nodes can be modified.
3625             */
3626            if (dt_node_is_integer(lp) == 0 ||
3627                dt_node_is_integer(rp) == 0) {
3628                    if (!dt_node_is_pointer(lp) || dt_node_is_vfptr(lp)) {
3629                            xyerror(D_OP_VFPTR,
3630                                "operator %s requires left-hand scalar "
3631                                "operand of known size\n", opstr(op));
3632                    } else if (dt_node_is_integer(rp) == 0 &&
3633                        dt_node_is_ptrcompat(lp, rp, NULL, NULL) == 0) {
3634                            xyerror(D_OP_INCOMPAT, "operands have "
3635                                "incompatible types: \"%s\" %s \"%s\"\n",
3636                                dt_node_type_name(lp, n1, sizeof (n1)),
3637                                opstr(op),
3638                                dt_node_type_name(rp, n2, sizeof (n2)));
3639                    }
3640            }
3641    asgn_common:
3642            dt_assign_common(dnp);
3643            break;

3645        case DT_TOK_PTR:
3646                /*
3647                 * If the left-hand side of operator -> is the name "self",
3648                 * then we permit a TLS variable to be created or referenced.
3649                 */
3650                if (lp->dn_kind == DT_NODE_IDENT &&
3651                    strcmp(lp->dn_string, "self") == 0) {
3652                        if (rp->dn_kind != DT_NODE_VAR) {
3653                                dt_xcook_ident(rp, dtp->dt_tls,
3654                                    DT_IDENT_SCALAR, B_TRUE);
3655                        }

3657                        if (idflags != 0)
3658                                rp = dt_node_cook(rp, idflags);
```

```
3660                            dnp->dn_right = dnp->dn_left; /* avoid freeing rp */
3661                            dt_node_free(dnp);
3662                            return (rp);
3663                    }

3665                    /*
3666                     * If the left-hand side of operator -> is the name "this",
3667                     * then we permit a local variable to be created or referenced.
3668                     */
3669                    if (lp->dn_kind == DT_NODE_IDENT &&
3670                        strcmp(lp->dn_string, "this") == 0) {
3671                            if (rp->dn_kind != DT_NODE_VAR) {
3672                                    dt_xcook_ident(rp, yypcb->pcb_locals,
3673                                        DT_IDENT_SCALAR, B_TRUE);
3674                            }

3676                            if (idflags != 0)
3677                                    rp = dt_node_cook(rp, idflags);

3679                            dnp->dn_right = dnp->dn_left; /* avoid freeing rp */
3680                            dt_node_free(dnp);
3681                            return (rp);
3682                    }

3684                    /*FALLTHRU*/

3686            case DT_TOK_DOT:
3687                    lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);

3689                    if (rp->dn_kind != DT_NODE_IDENT) {
3690                            xyerror(D_OP_IDENT, "operator %s must be followed by "
3691                                "an identifier\n", opstr(op));
3692                    }

3694                    if ((idp = dt_node_resolve(lp, DT_IDENT_XLSOU)) != NULL ||
3695                        (idp = dt_node_resolve(lp, DT_IDENT_XLPTR)) != NULL) {
3696                            /*
3697                             * If the left-hand side is a translated struct or ptr,
3698                             * the type of the left is the translation output type.
3699                             */
3700                            dt_xlator_t *dxp = idp->di_data;

3702                            if (dt_xlator_member(dxp, rp->dn_string) == NULL) {
3703                                    xyerror(D_XLATE_NOCONV,
3704                                        "translator does not define conversion "
3705                                        "for member: %s\n", rp->dn_string);
3706                            }

3708                            ctfp = idp->di_ctfp;
3709                            type = ctf_type_resolve(ctfp, idp->di_type);
3710                            uref = idp->di_flags & DT_IDFLG_USER;
3711                    } else {
3712                            ctfp = lp->dn_ctfp;
3713                            type = ctf_type_resolve(ctfp, lp->dn_type);
3714                            uref = lp->dn_flags & DT_NF_USERLAND;
3715                    }

3717                    kind = ctf_type_kind(ctfp, type);

3719                    if (op == DT_TOK_PTR) {
3720                            if (kind != CTF_K_POINTER) {
3721                                    xyerror(D_OP_PTR, "operator %s must be "
3722                                        "applied to a pointer\n", opstr(op));
3723                            }
3724                            type = ctf_type_reference(ctfp, type);
```



```
3725                            type = ctf_type_resolve(ctfp, type);
3726                            kind = ctf_type_kind(ctfp, type);
3727                    }

3729                    /*
3730                     * If we follow a reference to a forward declaration tag,
3731                     * search the entire type space for the actual definition.
3732                     */
3733                    while (kind == CTF_K_FORWARD) {
3734                            char *tag = ctf_type_name(ctfp, type, n1, sizeof (n1));
3735                            dtrace_typeinfo_t dtt;

3737                            if (tag != NULL && dt_type_lookup(tag, &dtt) == 0 &&
3738                                (dtt.dtt_ctfp != ctfp || dtt.dtt_type != type)) {
3739                                    ctfp = dtt.dtt_ctfp;
3740                                    type = ctf_type_resolve(ctfp, dtt.dtt_type);
3741                                    kind = ctf_type_kind(ctfp, type);
3742                            } else {
3743                                    xyerror(D_OP_INCOMPLETE,
3744                                        "operator %s cannot be applied to a "
3745                                        "forward declaration: no %s definition "
3746                                        "is available\n", opstr(op), tag);
3747                            }
3748                    }

3750                    if (kind != CTF_K_STRUCT && kind != CTF_K_UNION) {
3751                            if (op == DT_TOK_PTR) {
3752                                    xyerror(D_OP_SOU, "operator -> cannot be "
3753                                        "applied to pointer to type \"%s\"; must "
3754                                        "be applied to a struct or union pointer\n",
3755                                        ctf_type_name(ctfp, type, n1, sizeof (n1)));
3756                            } else {
3757                                    xyerror(D_OP_SOU, "operator %s cannot be "
3758                                        "applied to type \"%s\"; must be applied "
3759                                        "to a struct or union\n", opstr(op),
3760                                        ctf_type_name(ctfp, type, n1, sizeof (n1)));
3761                            }
3762                    }

3764                    if (ctf_member_info(ctfp, type, rp->dn_string, &m) == CTF_ERR) {
3765                            xyerror(D_TYPE_MEMBER,
3766                                "%s is not a member of %s\n", rp->dn_string,
3767                                ctf_type_name(ctfp, type, n1, sizeof (n1)));
3768                    }

3770                    type = ctf_type_resolve(ctfp, m.ctm_type);
3771                    kind = ctf_type_kind(ctfp, type);

3773                    dt_node_type_assign(dnp, ctfp, m.ctm_type, B_FALSE);
3051                    dt_node_type_assign(dnp, ctfp, m.ctm_type);
3774                    dt_node_attr_assign(dnp, lp->dn_attr);

3776                    if (op == DT_TOK_PTR && (kind != CTF_K_ARRAY ||
3777                        dt_node_is_string(dnp)))
3778                            dnp->dn_flags |= DT_NF_LVALUE; /* see K&R[A7.3.3] */

3780                    if (op == DT_TOK_DOT && (lp->dn_flags & DT_NF_LVALUE) &&
3781                        (kind != CTF_K_ARRAY || dt_node_is_string(dnp)))
3782                            dnp->dn_flags |= DT_NF_LVALUE; /* see K&R[A7.3.3] */

3784                    if (lp->dn_flags & DT_NF_WRITABLE)
3785                            dnp->dn_flags |= DT_NF_WRITABLE;

3787                    if (uref && (kind == CTF_K_POINTER ||
3788                        (dnp->dn_flags & DT_NF_REF)))
3789                            dnp->dn_flags |= DT_NF_USERLAND;
```

```
3790                 break;
3791
3792         case DT_TOK_LBRAC: {
3793                 /*
3794                  * If op is DT_TOK_LBRAC, we know from the special-case code at
3795                  * the top that lp is either a D variable or an aggregation.
3796                  */
3797                 dt_node_t *lnp;
3798
3799                 /*
3800                  * If the left-hand side is an aggregation, just set dn_aggtup
3801                  * to the right-hand side and return the cooked aggregation.
3802                  * This transformation is legal since we are just collapsing
3803                  * nodes to simplify later processing, and the entire aggtup
3804                  * parse subtree is retained for subsequent cooking passes.
3805                  */
3806                 if (lp->dn_kind == DT_NODE_AGG) {
3807                         if (lp->dn_aggtup != NULL) {
3808                                 xyerror(D_AGG_MDIM, "improper attempt to "
3809                                     "reference @%s as a multi-dimensional "
3810                                     "array\n", lp->dn_ident->di_name);
3811                         }
3812
3813                         lp->dn_aggtup = rp;
3814                         lp = dt_node_cook(lp, 0);
3815
3816                         dnp->dn_left = dnp->dn_right = NULL;
3817                         dt_node_free(dnp);
3818
3819                         return (lp);
3820                 }
3821
3822                 assert(lp->dn_kind == DT_NODE_VAR);
3823                 idp = lp->dn_ident;
3824
3825                 /*
3826                  * If the left-hand side is a non-global scalar that hasn't yet
3827                  * been referenced or modified, it was just created by self->
3828                  * or this-> and we can convert it from scalar to assoc array.
3829                  */
3830                 if (idp->di_kind == DT_IDENT_SCALAR && dt_ident_unref(idp) &&
3831                     (idp->di_flags & (DT_IDFLG_LOCAL | DT_IDFLG_TLS)) != 0) {
3832
3833                         if (idp->di_flags & DT_IDFLG_LOCAL) {
3834                                 xyerror(D_ARR_LOCAL,
3835                                     "local variables may not be used as "
3836                                     "associative arrays: %s\n", idp->di_name);
3837                         }
3838
3839                         dt_dprintf("morph variable %s (id %u) from scalar to "
3840                             "array\n", idp->di_name, idp->di_id);
3841
3842                         dt_ident_morph(idp, DT_IDENT_ARRAY,
3843                             &dt_idops_assc, NULL);
3844                 }
3845
3846                 if (idp->di_kind != DT_IDENT_ARRAY) {
3847                         xyerror(D_IDENT_BADREF, "%s '%s' may not be referenced "
3848                             "as %s\n", dt_idkind_name(idp->di_kind),
3849                             idp->di_name, dt_idkind_name(DT_IDENT_ARRAY));
3850                 }
3851
3852                 /*
3853                  * Now that we've confirmed our left-hand side is a DT_NODE_VAR
3854                  * of idkind DT_IDENT_ARRAY, we need to splice the [ node from
3855                  * the parse tree and leave a cooked DT_NODE_VAR in its place
```



```
3856                  * where dn_args for the VAR node is the right-hand 'rp' tree,
3857                  * as shown in the parse tree diagram below:
3858                  *
3859                  *        /                       /
3860                  * [ OP2 "[" ]=dnp             [ VAR ]=dnp
3861                  *       /   \          =>       |
3862                  *      /     \                  +- dn_args -> [ ??? ]=rp
3863                  * [ VAR ]=lp  [ ??? ]=rp
3864                  *
3865                  * Since the final dt_node_cook(dnp) can fail using longjmp we
3866                  * must perform the transformations as a group first by over-
3867                  * writing 'dnp' to become the VAR node, so that the parse tree
3868                  * is guaranteed to be in a consistent state if the cook fails.
3869                  */
3870                 assert(lp->dn_kind == DT_NODE_VAR);
3871                 assert(lp->dn_args == NULL);
3872
3873                 lnp = dnp->dn_link;
3874                 bcopy(lp, dnp, sizeof (dt_node_t));
3875                 dnp->dn_link = lnp;
3876
3877                 dnp->dn_args = rp;
3878                 dnp->dn_list = NULL;
3879
3880                 dt_node_free(lp);
3881                 return (dt_node_cook(dnp, idflags));
3882         }
3883
3884         case DT_TOK_XLATE: {
3885                 dt_xlator_t *dxp;
3886
3887                 assert(lp->dn_kind == DT_NODE_TYPE);
3888                 rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);
3889                 dxp = dt_xlator_lookup(dtp, rp, lp, DT_XLATE_FUZZY);
3890
3891                 if (dxp == NULL) {
3892                         xyerror(D_XLATE_NONE,
3893                             "cannot translate from \"%s\" to \"%s\"\n",
3894                             dt_node_type_name(rp, n1, sizeof (n1)),
3895                             dt_node_type_name(lp, n2, sizeof (n2)));
3896                 }
3897
3898                 dnp->dn_ident = dt_xlator_ident(dxp, lp->dn_ctfp, lp->dn_type);
3899                 dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp),
3900                     B_FALSE);
3177                 dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp));
3901                 dt_node_attr_assign(dnp,
3902                     dt_attr_min(rp->dn_attr, dnp->dn_ident->di_attr));
3903                 break;
3904         }
3905
3906         case DT_TOK_LPAR: {
3907                 ctf_id_t ltype, rtype;
3908                 uint_t lkind, rkind;
3909
3910                 assert(lp->dn_kind == DT_NODE_TYPE);
3911                 rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);
3912
3913                 ltype = ctf_type_resolve(lp->dn_ctfp, lp->dn_type);
3914                 lkind = ctf_type_kind(lp->dn_ctfp, ltype);
3915
3916                 rtype = ctf_type_resolve(rp->dn_ctfp, rp->dn_type);
3917                 rkind = ctf_type_kind(rp->dn_ctfp, rtype);
3918
3919                 /*
3920                  * The rules for casting are loosely explained in K&R[A7.5]
```

```
3921                  * and K&R[A6].  Basically, we can cast to the same type or
3922                  * same base type, between any kind of scalar values, from
3923                  * arrays to pointers, and we can cast anything to void.
3924                  * To these rules D adds casts from scalars to strings.
3925                  */
3926                 if (ctf_type_compat(lp->dn_ctfp, lp->dn_type,
3927                     rp->dn_ctfp, rp->dn_type))
3928                         /*EMPTY*/;
3929                 else if (dt_node_is_scalar(lp) &&
3930                     (dt_node_is_scalar(rp) || rkind == CTF_K_FUNCTION))
3931                         /*EMPTY*/;
3932                 else if (dt_node_is_void(lp))
3933                         /*EMPTY*/;
3934                 else if (lkind == CTF_K_POINTER && dt_node_is_pointer(rp))
3935                         /*EMPTY*/;
3936                 else if (dt_node_is_string(lp) && (dt_node_is_scalar(rp) ||
3937                     dt_node_is_pointer(rp) || dt_node_is_strcompat(rp)))
3938                         /*EMPTY*/;
3939                 else {
3940                         xyerror(D_CAST_INVAL,
3941                             "invalid cast expression: \"%s\" to \"%s\"\n",
3942                             dt_node_type_name(rp, n1, sizeof (n1)),
3943                             dt_node_type_name(lp, n2, sizeof (n2)));
3944                 }

3946                 dt_node_type_propagate(lp, dnp); /* see K&R[A7.5] */
3947                 dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));

3949                 /*
3950                  * If it's a pointer then should be able to (attempt to)
3951                  * assign to it.
3952                  */
3953                 if (lkind == CTF_K_POINTER)
3954                         dnp->dn_flags |= DT_NF_WRITABLE;

3956                 break;
3957         }

3959         case DT_TOK_COMMA:
3960                 lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3961                 rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3963                 if (dt_node_is_dynamic(lp) || dt_node_is_dynamic(rp)) {
3964                         xyerror(D_OP_DYN, "operator %s operands "
3965                             "cannot be of dynamic type\n", opstr(op));
3966                 }

3968                 if (dt_node_is_actfunc(lp) || dt_node_is_actfunc(rp)) {
3969                         xyerror(D_OP_ACT, "operator %s operands "
3970                             "cannot be actions\n", opstr(op));
3971                 }

3973                 dt_node_type_propagate(rp, dnp); /* see K&R[A7.18] */
3974                 dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));
3975                 break;

3977         default:
3978                 xyerror(D_UNKNOWN, "invalid binary op %s\n", opstr(op));
3979         }

3981         /*
3982          * Complete the conversion of E1[E2] to *((E1)+(E2)) that we started
3983          * at the top of our switch() above (see K&R[A7.3.1]).  Since E2 is
3984          * parsed as an argument_expression_list by dt_grammar.y, we can
3985          * end up with a comma-separated list inside of a non-associative
3986          * array reference.  We check for this and report an appropriate error.
```



```
3987          */
3988         if (dnp->dn_op == DT_TOK_LBRAC && op == DT_TOK_ADD) {
3989                 dt_node_t *pnp;

3991                 if (rp->dn_list != NULL) {
3992                         xyerror(D_ARR_BADREF,
3993                             "cannot access %s as an associative array\n",
3994                             dt_node_name(lp, n1, sizeof (n1)));
3995                 }

3997                 dnp->dn_op = DT_TOK_ADD;
3998                 pnp = dt_node_op1(DT_TOK_DEREF, dnp);

4000                 /*
4001                  * Cook callbacks are not typically permitted to allocate nodes.
4002                  * When we do, we must insert them in the middle of an existing
4003                  * allocation list rather than having them appended to the pcb
4004                  * list because the sub-expression may be part of a definition.
4005                  */
4006                 assert(yypcb->pcb_list == pnp);
4007                 yypcb->pcb_list = pnp->dn_link;

4009                 pnp->dn_link = dnp->dn_link;
4010                 dnp->dn_link = pnp;

4012                 return (dt_node_cook(pnp, DT_IDFLG_REF));
4013         }

4015         return (dnp);
4016 }

4018 /*ARGSUSED*/
4019 static dt_node_t *
4020 dt_cook_op3(dt_node_t *dnp, uint_t idflags)
4021 {
4022         dt_node_t *lp, *rp;
4023         ctf_file_t *ctfp;
4024         ctf_id_t type;

4026         dnp->dn_expr = dt_node_cook(dnp->dn_expr, DT_IDFLG_REF);
4027         lp = dnp->dn_left = dt_node_cook(dnp->dn_left, DT_IDFLG_REF);
4028         rp = dnp->dn_right = dt_node_cook(dnp->dn_right, DT_IDFLG_REF);

4030         if (!dt_node_is_scalar(dnp->dn_expr)) {
4031                 xyerror(D_OP_SCALAR,
4032                     "operator ?: expression must be of scalar type\n");
4033         }

4035         if (dt_node_is_dynamic(lp) || dt_node_is_dynamic(rp)) {
4036                 xyerror(D_OP_DYN,
4037                     "operator ?: operands cannot be of dynamic type\n");
4038         }

4040         /*
4041          * The rules for type checking for the ternary operator are complex and
4042          * are described in the ANSI-C spec (see K&R[A7.16]).  We implement
4043          * the various tests in order from least to most expensive.
4044          */
4045         if (ctf_type_compat(lp->dn_ctfp, lp->dn_type,
4046             rp->dn_ctfp, rp->dn_type)) {
4047                 ctfp = lp->dn_ctfp;
4048                 type = lp->dn_type;
4049         } else if (dt_node_is_integer(lp) && dt_node_is_integer(rp)) {
4050                 dt_type_promote(lp, rp, &ctfp, &type);
4051         } else if (dt_node_is_strcompat(lp) && dt_node_is_strcompat(rp) &&
4052             (dt_node_is_string(lp) || dt_node_is_string(rp))) {
```

```
4053                ctfp = DT_STR_CTFP(yypcb->pcb_hdl);
4054                type = DT_STR_TYPE(yypcb->pcb_hdl);
4055        } else if (dt_node_is_ptrcompat(lp, rp, &ctfp, &type) == 0) {
4056                xyerror(D_OP_INCOMPAT,
4057                    "operator ?: operands must have compatible types\n");
4058        }

4060        if (dt_node_is_actfunc(lp) || dt_node_is_actfunc(rp)) {
4061                xyerror(D_OP_ACT, "action cannot be "
4062                    "used in a conditional context\n");
4063        }

4065        dt_node_type_assign(dnp, ctfp, type, B_FALSE);
3342        dt_node_type_assign(dnp, ctfp, type);
4066        dt_node_attr_assign(dnp, dt_attr_min(dnp->dn_expr->dn_attr,
4067            dt_attr_min(lp->dn_attr, rp->dn_attr)));

4069        return (dnp);
4070 }
______unchanged_portion_omitted_

4081 /*
4082  * If dn_aggfun is set, this node is a collapsed aggregation assignment (see
4083  * the special case code for DT_TOK_ASGN in dt_cook_op2() above), in which
4084  * case we cook both the tuple and the function call.  If dn_aggfun is NULL,
4085  * this node is just a reference to the aggregation's type and attributes.
4086  */
4087 /*ARGSUSED*/
4088 static dt_node_t *
4089 dt_cook_aggregation(dt_node_t *dnp, uint_t idflags)
4090 {
4091        dtrace_hdl_t *dtp = yypcb->pcb_hdl;

4093        if (dnp->dn_aggfun != NULL) {
4094                dnp->dn_aggfun = dt_node_cook(dnp->dn_aggfun, DT_IDFLG_REF);
4095                dt_node_attr_assign(dnp, dt_ident_cook(dnp,
4096                    dnp->dn_ident, &dnp->dn_aggtup));
4097        } else {
4098                dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp),
4099                    B_FALSE);
3375                dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp));
4100                dt_node_attr_assign(dnp, dnp->dn_ident->di_attr);
4101        }

4103        return (dnp);
4104 }
______unchanged_portion_omitted_

4270 /*ARGSUSED*/
4271 static dt_node_t *
4272 dt_cook_xlator(dt_node_t *dnp, uint_t idflags)
4273 {
4274        dtrace_hdl_t *dtp = yypcb->pcb_hdl;
4275        dt_xlator_t *dxp = dnp->dn_xlator;
4276        dt_node_t *mnp;

4278        char n1[DT_TYPE_NAMELEN];
4279        char n2[DT_TYPE_NAMELEN];

4281        dtrace_attribute_t attr = _dtrace_maxattr;
4282        ctf_membinfo_t ctm;

4284        /*
4285         * Before cooking each translator member, we push a reference to the
4286         * hash containing translator-local identifiers on to pcb_globals to
4287         * temporarily interpose these identifiers in front of other globals.
```



```
4288         */
4289        dt_idstack_push(&yypcb->pcb_globals, dxp->dx_locals);

4291        for (mnp = dnp->dn_members; mnp != NULL; mnp = mnp->dn_list) {
4292                if (ctf_member_info(dxp->dx_dst_ctfp, dxp->dx_dst_type,
4293                    mnp->dn_membname, &ctm) == CTF_ERR) {
4294                        xyerror(D_XLATE_MEMB,
4295                            "translator member %s is not a member of %s\n",
4296                            mnp->dn_membname, ctf_type_name(dxp->dx_dst_ctfp,
4297                            dxp->dx_dst_type, n1, sizeof (n1)));
4298                }

4300                (void) dt_node_cook(mnp, DT_IDFLG_REF);
4301                dt_node_type_assign(mnp, dxp->dx_dst_ctfp, ctm.ctm_type,
4302                    B_FALSE);
3577                dt_node_type_assign(mnp, dxp->dx_dst_ctfp, ctm.ctm_type);
4303                attr = dt_attr_min(attr, mnp->dn_attr);

4305                if (dt_node_is_argcompat(mnp, mnp->dn_membexpr) == 0) {
4306                        xyerror(D_XLATE_INCOMPAT,
4307                            "translator member %s definition uses "
4308                            "incompatible types: \"%s\" = \"%s\"\n",
4309                            mnp->dn_membname,
4310                            dt_node_type_name(mnp, n1, sizeof (n1)),
4311                            dt_node_type_name(mnp->dn_membexpr,
4312                            n2, sizeof (n2)));
4313                }
4314        }

4316        dt_idstack_pop(&yypcb->pcb_globals, dxp->dx_locals);

4318        dxp->dx_souid.di_attr = attr;
4319        dxp->dx_ptrid.di_attr = attr;

4321        dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp), B_FALSE);
3596        dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp));
4322        dt_node_attr_assign(dnp, _dtrace_defattr);

4324        return (dnp);
4325 }
______unchanged_portion_omitted_

4596 /*
4597  * Compute the DOF dtrace_diftype_t representation of a node's type.  This is
4598  * called from a variety of places in the library so it cannot assume yypcb
4599  * is valid: any references to handle-specific data must be made through 'dtp'.
4600  */
4601 void
4602 dt_node_diftype(dtrace_hdl_t *dtp, const dt_node_t *dnp, dtrace_diftype_t *tp)
4603 {
4604        if (dnp->dn_ctfp == DT_STR_CTFP(dtp) &&
4605            dnp->dn_type == DT_STR_TYPE(dtp)) {
4606                tp->dtdt_kind = DIF_TYPE_STRING;
4607                tp->dtdt_ckind = CTF_K_UNKNOWN;
4608        } else {
4609                tp->dtdt_kind = DIF_TYPE_CTF;
4610                tp->dtdt_ckind = ctf_type_kind(dnp->dn_ctfp,
4611                    ctf_type_resolve(dnp->dn_ctfp, dnp->dn_type));
4612        }

4614        tp->dtdt_flags = (dnp->dn_flags & DT_NF_REF) ?
4615            (dnp->dn_flags & DT_NF_USERLAND) ? DIF_TF_BYUREF :
4616            DIF_TF_BYREF : 0;
3889        tp->dtdt_flags = (dnp->dn_flags & DT_NF_REF) ? DIF_TF_BYREF : 0;
4617        tp->dtdt_pad = 0;
4618        tp->dtdt_size = ctf_type_size(dnp->dn_ctfp, dnp->dn_type);
```

```
4619 }
______unchanged_portion_omitted_
```

```
**********************************************************
   11453 Tue Jan 14 16:48:57 2014
new/usr/src/lib/libdtrace/common/dt_parser.h
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 * Copyright (c) 2013 Joyent, Inc. All rights reserved.
 */
#endif /* ! codereview */

#ifndef _DT_PARSER_H
#define _DT_PARSER_H

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

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

#include <libctf.h>
#include <stdarg.h>
#include <stdio.h>

#ifdef  __cplusplus
extern "C" {
#endif

#include <dt_errtags.h>
#include <dt_ident.h>
#include <dt_decl.h>
#include <dt_xlator.h>
#include <dt_list.h>

typedef struct dt_node {
        ctf_file_t *dn_ctfp;    /* CTF type container for node's type */
        ctf_id_t dn_type;       /* CTF type reference for node's type */
        uchar_t dn_kind;        /* node kind (DT_NODE_*, defined below) */
```



```
        uchar_t dn_flags;       /* node flags (DT_NF_*, defined below) */
        ushort_t dn_op;         /* operator (DT_TOK_*, defined by lex) */
        int dn_line;            /* line number for error messages */
        int dn_reg;             /* register allocated by cg */
        dtrace_attribute_t dn_attr; /* node stability attributes */

        /*
         * D compiler nodes, as is the usual style, contain a union of the
         * different sub-elements required by the various kinds of nodes.
         * These sub-elements are accessed using the macros defined below.
         */
        union {
                struct {
                        uintmax_t _value;       /* integer value */
                        char *_string;          /* string value */
                } _const;

                struct {
                        dt_ident_t *_ident;     /* identifier reference */
                        struct dt_node *_links[3]; /* child node pointers */
                } _nodes;

                struct {
                        struct dt_node *_descs; /* list of descriptions */
                        struct dt_node *_pred;  /* predicate expression */
                        struct dt_node *_acts;  /* action statement list */
                        dt_idhash_t *_locals;   /* local variable hash */
                        dtrace_attribute_t _attr; /* context attributes */
                } _clause;

                struct {
                        char *_spec;            /* specifier string (if any) */
                        dtrace_probedesc_t *_desc; /* final probe description */
                } _pdesc;

                struct {
                        char *_name;            /* string name of member */
                        struct dt_node *_expr;  /* expression node pointer */
                        dt_xlator_t *_xlator;   /* translator reference */
                        uint_t _id;             /* member identifier */
                } _member;

                struct {
                        dt_xlator_t *_xlator;   /* translator reference */
                        struct dt_node *_xmemb; /* individual xlator member */
                        struct dt_node *_membs; /* list of member nodes */
                } _xlator;

                struct {
                        char *_name;            /* string name of provider */
                        struct dt_provider *_pvp; /* provider references */
                        struct dt_node *_probes;  /* list of probe nodes */
                        int _redecl;            /* provider redeclared */
                } _provider;
        } dn_u;

        struct dt_node *dn_list; /* parse tree list link */
        struct dt_node *dn_link; /* allocation list link */
} dt_node_t;

#define dn_value        dn_u._const._value      /* DT_NODE_INT */
#define dn_string       dn_u._const._string     /* STRING, IDENT, TYPE */
#define dn_ident        dn_u._nodes._ident      /* VAR,SYM,FUN,AGG,INL,PROBE */
#define dn_args         dn_u._nodes._links[0]   /* DT_NODE_VAR, FUNC */
#define dn_child        dn_u._nodes._links[0]   /* DT_NODE_OP1 */
#define dn_left         dn_u._nodes._links[0]   /* DT_NODE_OP2, OP3 */
```

```
121 #define dn_right        dn_u._nodes._links[1]   /* DT_NODE_OP2, OP3 */
122 #define dn_expr         dn_u._nodes._links[2]   /* DT_NODE_OP3, DEXPR */
123 #define dn_aggfun       dn_u._nodes._links[0]   /* DT_NODE_AGG */
124 #define dn_aggtup       dn_u._nodes._links[1]   /* DT_NODE_AGG */
125 #define dn_pdescs       dn_u._clause._descs     /* DT_NODE_CLAUSE */
126 #define dn_pred         dn_u._clause._pred      /* DT_NODE_CLAUSE */
127 #define dn_acts         dn_u._clause._acts      /* DT_NODE_CLAUSE */
128 #define dn_locals       dn_u._clause._locals    /* DT_NODE_CLAUSE */
129 #define dn_ctxattr      dn_u._clause._attr      /* DT_NODE_CLAUSE */
130 #define dn_spec         dn_u._pdesc._spec       /* DT_NODE_PDESC */
131 #define dn_desc         dn_u._pdesc._desc       /* DT_NODE_PDESC */
132 #define dn_membname     dn_u._member._name      /* DT_NODE_MEMBER */
133 #define dn_membexpr     dn_u._member._expr      /* DT_NODE_MEMBER */
134 #define dn_membxlator   dn_u._member._xlator    /* DT_NODE_MEMBER */
135 #define dn_membid       dn_u._member._id        /* DT_NODE_MEMBER */
136 #define dn_xlator       dn_u._xlator._xlator    /* DT_NODE_XLATOR */
137 #define dn_xmember      dn_u._xlator._xmemb     /* DT_NODE_XLATOR */
138 #define dn_members      dn_u._xlator._membs     /* DT_NODE_XLATOR */
139 #define dn_provname     dn_u._provider._name    /* DT_NODE_PROVIDER */
140 #define dn_provider     dn_u._provider._pvp     /* DT_NODE_PROVIDER */
141 #define dn_provred      dn_u._provider._redecl  /* DT_NODE_PROVIDER */
142 #define dn_probes       dn_u._provider._probes  /* DT_NODE_PROVIDER */

144 #define DT_NODE_FREE    0       /* unused node (waiting to be freed) */
145 #define DT_NODE_INT     1       /* integer value */
146 #define DT_NODE_STRING  2       /* string value */
147 #define DT_NODE_IDENT   3       /* identifier */
148 #define DT_NODE_VAR     4       /* variable reference */
149 #define DT_NODE_SYM     5       /* symbol reference */
150 #define DT_NODE_TYPE    6       /* type reference or formal parameter */
151 #define DT_NODE_FUNC    7       /* function call */
152 #define DT_NODE_OP1     8       /* unary operator */
153 #define DT_NODE_OP2     9       /* binary operator */
154 #define DT_NODE_OP3     10      /* ternary operator */
155 #define DT_NODE_DEXPR   11      /* D expression action */
156 #define DT_NODE_DFUNC   12      /* D function action */
157 #define DT_NODE_AGG     13      /* aggregation */
158 #define DT_NODE_PDESC   14      /* probe description */
159 #define DT_NODE_CLAUSE  15      /* clause definition */
160 #define DT_NODE_INLINE  16      /* inline definition */
161 #define DT_NODE_MEMBER  17      /* member definition */
162 #define DT_NODE_XLATOR  18      /* translator definition */
163 #define DT_NODE_PROBE   19      /* probe definition */
164 #define DT_NODE_PROVIDER 20     /* provider definition */
165 #define DT_NODE_PROG    21      /* program translation unit */

167 #define DT_NF_SIGNED    0x01    /* data is a signed quantity (else unsigned) */
168 #define DT_NF_COOKED    0x02    /* data is a known type (else still cooking) */
169 #define DT_NF_REF       0x04    /* pass by reference (array, struct, union) */
170 #define DT_NF_LVALUE    0x08    /* node is an l-value according to ANSI-C */
171 #define DT_NF_WRITABLE  0x10    /* node is writable (can be modified) */
172 #define DT_NF_BITFIELD  0x20    /* node is an integer bitfield */
173 #define DT_NF_USERLAND  0x40    /* data is a userland address */

175 #define DT_TYPE_NAMELEN 128     /* reasonable size for ctf_type_name() */

177 extern int dt_node_is_integer(const dt_node_t *);
178 extern int dt_node_is_float(const dt_node_t *);
179 extern int dt_node_is_scalar(const dt_node_t *);
180 extern int dt_node_is_arith(const dt_node_t *);
181 extern int dt_node_is_vfptr(const dt_node_t *);
182 extern int dt_node_is_dynamic(const dt_node_t *);
183 extern int dt_node_is_stack(const dt_node_t *);
184 extern int dt_node_is_symaddr(const dt_node_t *);
185 extern int dt_node_is_usymaddr(const dt_node_t *);
186 extern int dt_node_is_string(const dt_node_t *);
```



```
187 extern int dt_node_is_strcompat(const dt_node_t *);
188 extern int dt_node_is_pointer(const dt_node_t *);
189 extern int dt_node_is_void(const dt_node_t *);
190 extern int dt_node_is_ptrcompat(const dt_node_t *, const dt_node_t *,
191         ctf_file_t **, ctf_id_t *);
192 extern int dt_node_is_argcompat(const dt_node_t *, const dt_node_t *);
193 extern int dt_node_is_posconst(const dt_node_t *);
194 extern int dt_node_is_actfunc(const dt_node_t *);

196 extern dt_node_t *dt_node_int(uintmax_t);
197 extern dt_node_t *dt_node_string(char *);
198 extern dt_node_t *dt_node_ident(char *);
199 extern dt_node_t *dt_node_type(dt_decl_t *);
200 extern dt_node_t *dt_node_vatype(void);
201 extern dt_node_t *dt_node_decl(void);
202 extern dt_node_t *dt_node_func(dt_node_t *, dt_node_t *);
203 extern dt_node_t *dt_node_offsetof(dt_decl_t *, char *);
204 extern dt_node_t *dt_node_op1(int, dt_node_t *);
205 extern dt_node_t *dt_node_op2(int, dt_node_t *, dt_node_t *);
206 extern dt_node_t *dt_node_op3(dt_node_t *, dt_node_t *, dt_node_t *);
207 extern dt_node_t *dt_node_statement(dt_node_t *);
208 extern dt_node_t *dt_node_pdesc_by_name(char *);
209 extern dt_node_t *dt_node_pdesc_by_id(uintmax_t);
210 extern dt_node_t *dt_node_clause(dt_node_t *, dt_node_t *, dt_node_t *);
211 extern dt_node_t *dt_node_inline(dt_node_t *);
212 extern dt_node_t *dt_node_member(dt_decl_t *, char *, dt_node_t *);
213 extern dt_node_t *dt_node_xlator(dt_decl_t *, dt_decl_t *, char *, dt_node_t *);
214 extern dt_node_t *dt_node_probe(char *, int, dt_node_t *, dt_node_t *);
215 extern dt_node_t *dt_node_provider(char *, dt_node_t *);
216 extern dt_node_t *dt_node_program(dt_node_t *);

218 extern dt_node_t *dt_node_link(dt_node_t *, dt_node_t *);
219 extern dt_node_t *dt_node_cook(dt_node_t *, uint_t);

221 extern dt_node_t *dt_node_xalloc(dtrace_hdl_t *, int);
222 extern void dt_node_free(dt_node_t *);

224 extern dtrace_attribute_t dt_node_list_cook(dt_node_t **, uint_t);
225 extern void dt_node_list_free(dt_node_t **);
226 extern void dt_node_link_free(dt_node_t **);

228 extern void dt_node_attr_assign(dt_node_t *, dtrace_attribute_t);
229 extern void dt_node_type_assign(dt_node_t *, ctf_file_t *, ctf_id_t, boolean_t);
222 extern void dt_node_type_assign(dt_node_t *, ctf_file_t *, ctf_id_t);
230 extern void dt_node_type_propagate(const dt_node_t *, dt_node_t *);
231 extern const char *dt_node_type_name(const dt_node_t *, char *, size_t);
232 extern size_t dt_node_type_size(const dt_node_t *);

234 extern dt_ident_t *dt_node_resolve(const dt_node_t *, uint_t);
235 extern size_t dt_node_sizeof(const dt_node_t *);
236 extern void dt_node_promote(dt_node_t *, dt_node_t *, dt_node_t *);

238 extern void dt_node_diftype(dtrace_hdl_t *,
239     const dt_node_t *, dtrace_diftype_t *);
240 extern void dt_node_printr(dt_node_t *, FILE *, int);
241 extern const char *dt_node_name(const dt_node_t *, char *, size_t);
242 extern int dt_node_root(dt_node_t *);

244 struct dtrace_typeinfo; /* see <dtrace.h> */
245 struct dt_pcb;          /* see <dt_impl.h> */

247 #define IS_CHAR(e) \
248         (((e).cte_format & (CTF_INT_CHAR | CTF_INT_SIGNED)) == \
249         (CTF_INT_CHAR | CTF_INT_SIGNED) && (e).cte_bits == NBBY)

251 #define IS_VOID(e) \
```

```
 252         ((e).cte_offset == 0 && (e).cte_bits == 0)

 254 extern int dt_type_lookup(const char *, struct dtrace_typeinfo *);
 255 extern int dt_type_pointer(struct dtrace_typeinfo *);
 256 extern const char *dt_type_name(ctf_file_t *, ctf_id_t, char *, size_t);

 258 typedef enum {
 259         YYS_CLAUSE,     /* lex/yacc state for finding program clauses */
 260         YYS_DEFINE,     /* lex/yacc state for parsing persistent definitions */
 261         YYS_EXPR,       /* lex/yacc state for parsing D expressions */
 262         YYS_DONE,       /* lex/yacc state for indicating parse tree is done */
 263         YYS_CONTROL     /* lex/yacc state for parsing control lines */
 264 } yystate_t;
```

*unchanged_portion_omitted_*

```
**********************************************************
   24082 Tue Jan 14 16:48:58 2014
new/usr/src/lib/libdtrace/common/dt_pid.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 */
/*
 * Copyright (c) 2013, Joyent, Inc.  All rights reserved.
 */
#endif /* ! codereview */

#include <assert.h>
#include <strings.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <alloca.h>
#include <libgen.h>
#include <stddef.h>
#include <sys/sysmacros.h>
#endif /* ! codereview */

#include <dt_impl.h>
#include <dt_program.h>
#include <dt_pid.h>
#include <dt_string.h>
#include <dt_module.h>
#endif /* ! codereview */

typedef struct dt_pid_probe {
        dtrace_hdl_t *dpp_dtp;
        dt_pcb_t *dpp_pcb;
        dt_proc_t *dpp_dpr;
        struct ps_prochandle *dpp_pr;
        const char *dpp_mod;
        char *dpp_func;
```



```
        const char *dpp_name;
        const char *dpp_obj;
        uintptr_t dpp_pc;
        size_t dpp_size;
        Lmid_t dpp_lmid;
        uint_t dpp_nmatches;
        uint64_t dpp_stret[4];
        GElf_Sym dpp_last;
        uint_t dpp_last_taken;
} dt_pid_probe_t;

/*
 * Compose the lmid and object name into the canonical representation. We
 * omit the lmid for the default link map for convenience.
 */
static void
dt_pid_objname(char *buf, size_t len, Lmid_t lmid, const char *obj)
{
        if (lmid == LM_ID_BASE)
                (void) strncpy(buf, obj, len);
        else
                (void) snprintf(buf, len, "LM%lx`%s", lmid, obj);
}

static int
dt_pid_error(dtrace_hdl_t *dtp, dt_pcb_t *pcb, dt_proc_t *dpr,
    fasttrap_probe_spec_t *ftp, dt_errtag_t tag, const char *fmt, ...)
{
        va_list ap;
        int len;

        if (ftp != NULL)
                dt_free(dtp, ftp);

        va_start(ap, fmt);
        if (pcb == NULL) {
                assert(dpr != NULL);
                len = vsnprintf(dpr->dpr_errmsg, sizeof (dpr->dpr_errmsg),
                    fmt, ap);
                assert(len >= 2);
                if (dpr->dpr_errmsg[len - 2] == '\n')
                        dpr->dpr_errmsg[len - 2] = '\0';
        } else {
                dt_set_errmsg(dtp, dt_errtag(tag), pcb->pcb_region,
                    pcb->pcb_filetag, pcb->pcb_fileptr ? yylineno : 0, fmt, ap);
        }
        va_end(ap);

        return (1);
}

static int
dt_pid_per_sym(dt_pid_probe_t *pp, const GElf_Sym *symp, const char *func)
{
        dtrace_hdl_t *dtp = pp->dpp_dtp;
        dt_pcb_t *pcb = pp->dpp_pcb;
        dt_proc_t *dpr = pp->dpp_dpr;
        fasttrap_probe_spec_t *ftp;
        uint64_t off;
        char *end;
        uint_t nmatches = 0;
        ulong_t sz;
        int glob, err;
        int isdash = strcmp("-", func) == 0;
        pid_t pid;
```

```
123         pid = Pstatus(pp->dpp_pr)->pr_pid;
124
125         dt_dprintf("creating probe pid%d:%s:%s:%s\n", (int)pid, pp->dpp_obj,
126             func, pp->dpp_name);
127
128         sz = sizeof (fasttrap_probe_spec_t) + (isdash ? 4 :
129             (symp->st_size - 1) * sizeof (ftp->ftps_offs[0]));
130
131         if ((ftp = dt_alloc(dtp, sz)) == NULL) {
132                 dt_dprintf("proc_per_sym: dt_alloc(%lu) failed\n", sz);
133                 return (1); /* errno is set for us */
134         }
135
136         ftp->ftps_pid = pid;
137         (void) strncpy(ftp->ftps_func, func, sizeof (ftp->ftps_func));
138
139         dt_pid_objname(ftp->ftps_mod, sizeof (ftp->ftps_mod), pp->dpp_lmid,
140             pp->dpp_obj);
141
142         if (!isdash && gmatch("return", pp->dpp_name)) {
143                 if (dt_pid_create_return_probe(pp->dpp_pr, dtp, ftp, symp,
144                     pp->dpp_stret) < 0) {
145                         return (dt_pid_error(dtp, pcb, dpr, ftp,
146                             D_PROC_CREATEFAIL, "failed to create return probe "
147                             "for '%s': %s", func,
148                             dtrace_errmsg(dtp, dtrace_errno(dtp))));
149                 }
150
151                 nmatches++;
152         }
153
154         if (!isdash && gmatch("entry", pp->dpp_name)) {
155                 if (dt_pid_create_entry_probe(pp->dpp_pr, dtp, ftp, symp) < 0) {
156                         return (dt_pid_error(dtp, pcb, dpr, ftp,
157                             D_PROC_CREATEFAIL, "failed to create entry probe "
158                             "for '%s': %s", func,
159                             dtrace_errmsg(dtp, dtrace_errno(dtp))));
160                 }
161
162                 nmatches++;
163         }
164
165         glob = strisglob(pp->dpp_name);
166         if (!glob && nmatches == 0) {
167                 off = strtoull(pp->dpp_name, &end, 16);
168                 if (*end != '\0') {
169                         return (dt_pid_error(dtp, pcb, dpr, ftp, D_PROC_NAME,
170                             "'%s' is an invalid probe name", pp->dpp_name));
171                 }
172
173                 if (off >= symp->st_size) {
174                         return (dt_pid_error(dtp, pcb, dpr, ftp, D_PROC_OFF,
175                             "offset 0x%llx outside of function '%s'",
176                             (u_longlong_t)off, func));
177                 }
178
179                 err = dt_pid_create_offset_probe(pp->dpp_pr, pp->dpp_dtp, ftp,
180                     symp, off);
181
182                 if (err == DT_PROC_ERR) {
183                         return (dt_pid_error(dtp, pcb, dpr, ftp,
184                             D_PROC_CREATEFAIL, "failed to create probe at "
185                             "'%s+0x%llx': %s", func, (u_longlong_t)off,
186                             dtrace_errmsg(dtp, dtrace_errno(dtp))));
187                 }
```



```
188
189                 if (err == DT_PROC_ALIGN) {
190                         return (dt_pid_error(dtp, pcb, dpr, ftp, D_PROC_ALIGN,
191                             "offset 0x%llx is not aligned on an instruction",
192                             (u_longlong_t)off));
193                 }
194
195                 nmatches++;
196
197         } else if (glob && !isdash) {
198                 if (dt_pid_create_glob_offset_probes(pp->dpp_pr,
199                     pp->dpp_dtp, ftp, symp, pp->dpp_name) < 0) {
200                         return (dt_pid_error(dtp, pcb, dpr, ftp,
201                             D_PROC_CREATEFAIL,
202                             "failed to create offset probes in '%s': %s", func,
203                             dtrace_errmsg(dtp, dtrace_errno(dtp))));
204                 }
205
206                 nmatches++;
207         }
208
209         pp->dpp_nmatches += nmatches;
210
211         dt_free(dtp, ftp);
212
213         return (0);
214 }
215
216 static int
217 dt_pid_sym_filt(void *arg, const GElf_Sym *symp, const char *func)
218 {
219         dt_pid_probe_t *pp = arg;
220
221         if (symp->st_shndx == SHN_UNDEF)
222                 return (0);
223
224         if (symp->st_size == 0) {
225                 dt_dprintf("st_size of %s is zero\n", func);
226                 return (0);
227         }
228
229         if (pp->dpp_last_taken == 0 ||
230             symp->st_value != pp->dpp_last.st_value ||
231             symp->st_size != pp->dpp_last.st_size) {
232                 /*
233                  * Due to 4524008, _init and _fini may have a bloated st_size.
234                  * While this bug has been fixed for a while, old binaries
235                  * may exist that still exhibit this problem. As a result, we
236                  * don't match _init and _fini though we allow users to
237                  * specify them explicitly.
238                  */
239                 if (strcmp(func, "_init") == 0 || strcmp(func, "_fini") == 0)
240                         return (0);
241
242                 if ((pp->dpp_last_taken = gmatch(func, pp->dpp_func)) != 0) {
243                         pp->dpp_last = *symp;
244                         return (dt_pid_per_sym(pp, symp, func));
245                 }
246         }
247
248         return (0);
249 }
250
251 static int
252 dt_pid_per_mod(void *arg, const prmap_t *pmp, const char *obj)
253 {
254         dt_pid_probe_t *pp = arg;
```

```
255         dtrace_hdl_t *dtp = pp->dpp_dtp;
256         dt_pcb_t *pcb = pp->dpp_pcb;
257         dt_proc_t *dpr = pp->dpp_dpr;
258         GElf_Sym sym;
259
260         if (obj == NULL)
261                 return (0);
262
263         (void) Plmid(pp->dpp_pr, pmp->pr_vaddr, &pp->dpp_lmid);
264
265         if ((pp->dpp_obj = strrchr(obj, '`')) == NULL)
266                 pp->dpp_obj = obj;
267         else
268                 pp->dpp_obj++;
269
270         if (Pxlookup_by_name(pp->dpp_pr, pp->dpp_lmid, obj, ".stret1", &sym,
271             NULL) == 0)
272                 pp->dpp_stret[0] = sym.st_value;
273         else
274                 pp->dpp_stret[0] = 0;
275
276         if (Pxlookup_by_name(pp->dpp_pr, pp->dpp_lmid, obj, ".stret2", &sym,
277             NULL) == 0)
278                 pp->dpp_stret[1] = sym.st_value;
279         else
280                 pp->dpp_stret[1] = 0;
281
282         if (Pxlookup_by_name(pp->dpp_pr, pp->dpp_lmid, obj, ".stret4", &sym,
283             NULL) == 0)
284                 pp->dpp_stret[2] = sym.st_value;
285         else
286                 pp->dpp_stret[2] = 0;
287
288         if (Pxlookup_by_name(pp->dpp_pr, pp->dpp_lmid, obj, ".stret8", &sym,
289             NULL) == 0)
290                 pp->dpp_stret[3] = sym.st_value;
291         else
292                 pp->dpp_stret[3] = 0;
293
294         dt_dprintf("%s stret %llx %llx %llx %llx\n", obj,
295             (u_longlong_t)pp->dpp_stret[0], (u_longlong_t)pp->dpp_stret[1],
296             (u_longlong_t)pp->dpp_stret[2], (u_longlong_t)pp->dpp_stret[3]);
297
298         /*
299          * If pp->dpp_func contains any globbing meta-characters, we need
300          * to iterate over the symbol table and compare each function name
301          * against the pattern.
302          */
303         if (!strisglob(pp->dpp_func)) {
304                 /*
305                  * If we fail to lookup the symbol, try interpreting the
306                  * function as the special "-" function that indicates that the
307                  * probe name should be interpreted as a absolute virtual
308                  * address. If that fails and we were matching a specific
309                  * function in a specific module, report the error, otherwise
310                  * just fail silently in the hopes that some other object will
311                  * contain the desired symbol.
312                  */
313                 if (Pxlookup_by_name(pp->dpp_pr, pp->dpp_lmid, obj,
314                     pp->dpp_func, &sym, NULL) != 0) {
315                         if (strcmp("-", pp->dpp_func) == 0) {
316                                 sym.st_name = 0;
317                                 sym.st_info =
318                                     GELF_ST_INFO(STB_LOCAL, STT_FUNC);
319                                 sym.st_other = 0;
320                                 sym.st_value = 0;
```



```
321                                 sym.st_size = Pstatus(pp->dpp_pr)->pr_dmodel ==
322                                     PR_MODEL_ILP32 ? -1U : -1ULL;
323
324                         } else if (!strisglob(pp->dpp_mod)) {
325                                 return (dt_pid_error(dtp, pcb, dpr, NULL,
326                                     D_PROC_FUNC,
327                                     "failed to lookup `%s` in module `%s`",
328                                     pp->dpp_func, pp->dpp_mod));
329                         } else {
330                                 return (0);
331                         }
332                 }
333
334                 /*
335                  * Only match defined functions of non-zero size.
336                  */
337                 if (GELF_ST_TYPE(sym.st_info) != STT_FUNC ||
338                     sym.st_shndx == SHN_UNDEF || sym.st_size == 0)
339                         return (0);
340
341                 /*
342                  * We don't instrument PLTs -- they're dynamically rewritten,
343                  * and, so, inherently dicey to instrument.
344                  */
345                 if (Ppltdest(pp->dpp_pr, sym.st_value) != NULL)
346                         return (0);
347
348                 (void) Plookup_by_addr(pp->dpp_pr, sym.st_value, pp->dpp_func,
349                     DTRACE_FUNCNAMELEN, &sym);
350
351                 return (dt_pid_per_sym(pp, &sym, pp->dpp_func));
352         } else {
353                 uint_t nmatches = pp->dpp_nmatches;
354
355                 if (Psymbol_iter_by_addr(pp->dpp_pr, obj, PR_SYMTAB,
356                     BIND_ANY | TYPE_FUNC, dt_pid_sym_filt, pp) == 1)
357                         return (1);
358
359                 if (nmatches == pp->dpp_nmatches) {
360                         /*
361                          * If we didn't match anything in the PR_SYMTAB, try
362                          * the PR_DYNSYM.
363                          */
364                         if (Psymbol_iter_by_addr(pp->dpp_pr, obj, PR_DYNSYM,
365                             BIND_ANY | TYPE_FUNC, dt_pid_sym_filt, pp) == 1)
366                                 return (1);
367                 }
368         }
369
370         return (0);
371 }
372
373 static int
374 dt_pid_mod_filt(void *arg, const prmap_t *pmp, const char *obj)
375 {
376         char name[DTRACE_MODNAMELEN];
377         dt_pid_probe_t *pp = arg;
378
379         if ((pp->dpp_obj = strrchr(obj, '`')) == NULL)
380                 pp->dpp_obj = obj;
381         else
382                 pp->dpp_obj++;
383
384         if (gmatch(pp->dpp_obj, pp->dpp_mod))
385                 return (dt_pid_per_mod(pp, pmp, obj));
```

```
387         (void) Plmid(pp->dpp_pr, pmp->pr_vaddr, &pp->dpp_lmid);

389         dt_pid_objname(name, sizeof (name), pp->dpp_lmid, pp->dpp_obj);

391         if (gmatch(name, pp->dpp_mod))
392                 return (dt_pid_per_mod(pp, pmp, obj));

394         return (0);
395 }

397 static const prmap_t *
398 dt_pid_fix_mod(dtrace_probedesc_t *pdp, struct ps_prochandle *P)
399 {
400         char m[MAXPATHLEN];
401         Lmid_t lmid = PR_LMID_EVERY;
402         const char *obj;
403         const prmap_t *pmp;

405         /*
406          * Pick apart the link map from the library name.
407          */
408         if (strchr(pdp->dtpd_mod, '`') != NULL) {
409                 char *end;

411                 if (strncmp(pdp->dtpd_mod, "LM", 2) != 0 ||
412                     !isdigit(pdp->dtpd_mod[2]))
413                         return (NULL);

415                 lmid = strtoul(&pdp->dtpd_mod[2], &end, 16);

417                 obj = end + 1;

419                 if (*end != '`' || strchr(obj, '`') != NULL)
420                         return (NULL);

422         } else {
423                 obj = pdp->dtpd_mod;
424         }

426         if ((pmp = Plmid_to_map(P, lmid, obj)) == NULL)
427                 return (NULL);

429         (void) Pobjname(P, pmp->pr_vaddr, m, sizeof (m));
430         if ((obj = strrchr(m, '/')) == NULL)
431                 obj = &m[0];
432         else
433                 obj++;

435         (void) Plmid(P, pmp->pr_vaddr, &lmid);
436         dt_pid_objname(pdp->dtpd_mod, sizeof (pdp->dtpd_mod), lmid, obj);

438         return (pmp);
439 }


442 static int
443 dt_pid_create_pid_probes(dtrace_probedesc_t *pdp, dtrace_hdl_t *dtp,
444     dt_pcb_t *pcb, dt_proc_t *dpr)
445 {
446         dt_pid_probe_t pp;
447         int ret = 0;

449         pp.dpp_dtp = dtp;
450         pp.dpp_dpr = dpr;
451         pp.dpp_pr = dpr->dpr_proc;
452         pp.dpp_pcb = pcb;
```



```
454         /*
455          * We can only trace dynamically-linked executables (since we've
456          * hidden some magic in ld.so.1 as well as libc.so.1).
457          */
458         if (Pname_to_map(pp.dpp_pr, PR_OBJ_LDSO) == NULL) {
459                 return (dt_pid_error(dtp, pcb, dpr, NULL, D_PROC_DYN,
460                     "process %s is not a dynamically-linked executable",
461                     &pdp->dtpd_provider[3]));
462         }

464         pp.dpp_mod = pdp->dtpd_mod[0] != '\0' ? pdp->dtpd_mod : "*";
465         pp.dpp_func = pdp->dtpd_func[0] != '\0' ? pdp->dtpd_func : "*";
466         pp.dpp_name = pdp->dtpd_name[0] != '\0' ? pdp->dtpd_name : "*";
467         pp.dpp_last_taken = 0;

469         if (strcmp(pp.dpp_func, "-") == 0) {
470                 const prmap_t *aout, *pmp;

472                 if (pdp->dtpd_mod[0] == '\0') {
473                         pp.dpp_mod = pdp->dtpd_mod;
474                         (void) strcpy(pdp->dtpd_mod, "a.out");
475                 } else if (strisglob(pp.dpp_mod) ||
476                     (aout = Pname_to_map(pp.dpp_pr, "a.out")) == NULL ||
477                     (pmp = Pname_to_map(pp.dpp_pr, pp.dpp_mod)) == NULL ||
478                     aout->pr_vaddr != pmp->pr_vaddr) {
479                         return (dt_pid_error(dtp, pcb, dpr, NULL, D_PROC_LIB,
480                             "only the a.out module is valid with the "
481                             "'-' function"));
482                 }

484                 if (strisglob(pp.dpp_name)) {
485                         return (dt_pid_error(dtp, pcb, dpr, NULL, D_PROC_NAME,
486                             "only individual addresses may be specified "
487                             "with the '-' function"));
488                 }
489         }

491         /*
492          * If pp.dpp_mod contains any globbing meta-characters, we need
493          * to iterate over each module and compare its name against the
494          * pattern. An empty module name is treated as '*'.
495          */
496         if (strisglob(pp.dpp_mod)) {
497                 ret = Pobject_iter(pp.dpp_pr, dt_pid_mod_filt, &pp);
498         } else {
499                 const prmap_t *pmp;
500                 char *obj;

502                 /*
503                  * If we can't find a matching module, don't sweat it -- either
504                  * we'll fail the enabling because the probes don't exist or
505                  * we'll wait for that module to come along.
506                  */
507                 if ((pmp = dt_pid_fix_mod(pdp, pp.dpp_pr)) != NULL) {
508                         if ((obj = strchr(pdp->dtpd_mod, '`')) == NULL)
509                                 obj = pdp->dtpd_mod;
510                         else
511                                 obj++;

513                         ret = dt_pid_per_mod(&pp, pmp, obj);
514                 }
515         }

517         return (ret);
518 }
```

```
520 static int
521 dt_pid_usdt_mapping(void *data, const prmap_t *pmp, const char *oname)
522 {
523         struct ps_prochandle *P = data;
524         GElf_Sym sym;
525         prsyminfo_t sip;
526         dof_helper_t dh;
527         GElf_Half e_type;
528         const char *mname;
529         const char *syms[] = { "___SUNW_dof", "__SUNW_dof" };
530         int i, fd = -1;

532         /*
533          * The symbol ___SUNW_dof is for lazy-loaded DOF sections, and
534          * __SUNW_dof is for actively-loaded DOF sections. We try to force
535          * in both types of DOF section since the process may not yet have
536          * run the code to instantiate these providers.
537          */
538         for (i = 0; i < 2; i++) {
539                 if (Pxlookup_by_name(P, PR_LMID_EVERY, oname, syms[i], &sym,
540                     &sip) != 0) {
541                         continue;
542                 }

544                 if ((mname = strrchr(oname, '/')) == NULL)
545                         mname = oname;
546                 else
547                         mname++;

549                 dt_dprintf("lookup of %s succeeded for %s\n", syms[i], mname);

551                 if (Pread(P, &e_type, sizeof (e_type), pmp->pr_vaddr +
552                     offsetof(Elf64_Ehdr, e_type)) != sizeof (e_type)) {
553                         dt_dprintf("read of ELF header failed");
554                         continue;
555                 }

557                 dh.dofhp_dof = sym.st_value;
558                 dh.dofhp_addr = (e_type == ET_EXEC) ? 0 : pmp->pr_vaddr;

560                 dt_pid_objname(dh.dofhp_mod, sizeof (dh.dofhp_mod),
561                     sip.prs_lmid, mname);

563                 if (fd == -1 &&
564                     (fd = pr_open(P, "/dev/dtrace/helper", O_RDWR, 0)) < 0) {
565                         dt_dprintf("pr_open of helper device failed: %s\n",
566                             strerror(errno));
567                         return (-1); /* errno is set for us */
568                 }

570                 if (pr_ioctl(P, fd, DTRACEHIOC_ADDDOF, &dh, sizeof (dh)) < 0)
571                         dt_dprintf("DOF was rejected for %s\n", dh.dofhp_mod);
572         }

574         if (fd != -1)
575                 (void) pr_close(P, fd);

577         return (0);
578 }

580 static int
581 dt_pid_create_usdt_probes(dtrace_probedesc_t *pdp, dtrace_hdl_t *dtp,
582     dt_pcb_t *pcb, dt_proc_t *dpr)
583 {
584         struct ps_prochandle *P = dpr->dpr_proc;
```



```
585         int ret = 0;

587         assert(MUTEX_HELD(&dpr->dpr_lock));

589         (void) Pupdate_maps(P);
590         if (Pobject_iter(P, dt_pid_usdt_mapping, P) != 0) {
591                 ret = -1;
592                 (void) dt_pid_error(dtp, pcb, dpr, NULL, D_PROC_USDT,
593                     "failed to instantiate probes for pid %d: %s",
594                     (int)Pstatus(P)->pr_pid, strerror(errno));
595         }

597         /*
598          * Put the module name in its canonical form.
599          */
600         (void) dt_pid_fix_mod(pdp, P);

602         return (ret);
603 }

605 static pid_t
606 dt_pid_get_pid(dtrace_probedesc_t *pdp, dtrace_hdl_t *dtp, dt_pcb_t *pcb,
607     dt_proc_t *dpr)
608 {
609         pid_t pid;
610         char *c, *last = NULL, *end;

612         for (c = &pdp->dtpd_provider[0]; *c != '\0'; c++) {
613                 if (!isdigit(*c))
614                         last = c;
615         }

617         if (last == NULL || (*(++last) == '\0')) {
618                 (void) dt_pid_error(dtp, pcb, dpr, NULL, D_PROC_BADPROV,
619                     "'%s' is not a valid provider", pdp->dtpd_provider);
620                 return (-1);
621         }

623         errno = 0;
624         pid = strtol(last, &end, 10);

626         if (errno != 0 || end == last || end[0] != '\0' || pid <= 0) {
627                 (void) dt_pid_error(dtp, pcb, dpr, NULL, D_PROC_BADPID,
628                     "'%s' does not contain a valid pid", pdp->dtpd_provider);
629                 return (-1);
630         }

632         return (pid);
633 }

635 int
636 dt_pid_create_probes(dtrace_probedesc_t *pdp, dtrace_hdl_t *dtp, dt_pcb_t *pcb)
637 {
638         char provname[DTRACE_PROVNAMELEN];
639         struct ps_prochandle *P;
640         dt_proc_t *dpr;
641         pid_t pid;
642         int err = 0;

644         assert(pcb != NULL);

646         if ((pid = dt_pid_get_pid(pdp, dtp, pcb, NULL)) == -1)
647                 return (-1);

649         if (dtp->dt_ftfd == -1) {
650                 if (dtp->dt_fterr == ENOENT) {
```

```
651                         (void) dt_pid_error(dtp, pcb, NULL, NULL, D_PROC_NODEV,
652                             "pid provider is not installed on this system");
653                 } else {
654                         (void) dt_pid_error(dtp, pcb, NULL, NULL, D_PROC_NODEV,
655                             "pid provider is not available: %s",
656                             strerror(dtp->dt_fterr));
657                 }

659                 return (-1);
660         }

662         (void) snprintf(provname, sizeof (provname), "pid%d", (int)pid);

664         if (gmatch(provname, pdp->dtpd_provider) != 0) {
665                 if ((P = dt_proc_grab(dtp, pid, PGRAB_RDONLY | PGRAB_FORCE,
666                     0)) == NULL) {
667                         (void) dt_pid_error(dtp, pcb, NULL, NULL, D_PROC_GRAB,
668                             "failed to grab process %d", (int)pid);
669                         return (-1);
670                 }

672                 dpr = dt_proc_lookup(dtp, P, 0);
673                 assert(dpr != NULL);
674                 (void) pthread_mutex_lock(&dpr->dpr_lock);

676                 if ((err = dt_pid_create_pid_probes(pdp, dtp, pcb, dpr)) == 0) {
677                         /*
678                          * Alert other retained enablings which may match
679                          * against the newly created probes.
680                          */
681                         (void) dt_ioctl(dtp, DTRACEIOC_ENABLE, NULL);
682                 }

684                 (void) pthread_mutex_unlock(&dpr->dpr_lock);
685                 dt_proc_release(dtp, P);
686         }

688         /*
689          * If it's not strictly a pid provider, we might match a USDT provider.
690          */
691         if (strcmp(provname, pdp->dtpd_provider) != 0) {
692                 if ((P = dt_proc_grab(dtp, pid, 0, 1)) == NULL) {
693                         (void) dt_pid_error(dtp, pcb, NULL, NULL, D_PROC_GRAB,
694                             "failed to grab process %d", (int)pid);
695                         return (-1);
696                 }

698                 dpr = dt_proc_lookup(dtp, P, 0);
699                 assert(dpr != NULL);
700                 (void) pthread_mutex_lock(&dpr->dpr_lock);

702                 if (!dpr->dpr_usdt) {
703                         err = dt_pid_create_usdt_probes(pdp, dtp, pcb, dpr);
704                         dpr->dpr_usdt = B_TRUE;
705                 }

707                 (void) pthread_mutex_unlock(&dpr->dpr_lock);
708                 dt_proc_release(dtp, P);
709         }

711         return (err ? -1 : 0);
712 }

714 int
715 dt_pid_create_probes_module(dtrace_hdl_t *dtp, dt_proc_t *dpr)
716 {
```



```
717         dtrace_prog_t *pgp;
718         dt_stmt_t *stp;
719         dtrace_probedesc_t *pdp, pd;
720         pid_t pid;
721         int ret = 0, found = B_FALSE;
722         char provname[DTRACE_PROVNAMELEN];

724         (void) snprintf(provname, sizeof (provname), "pid%d",
725             (int)dpr->dpr_pid);

727         for (pgp = dt_list_next(&dtp->dt_programs); pgp != NULL;
728             pgp = dt_list_next(pgp)) {

730                 for (stp = dt_list_next(&pgp->dp_stmts); stp != NULL;
731                     stp = dt_list_next(stp)) {

733                         pdp = &stp->ds_desc->dtsd_ecbdesc->dted_probe;
734                         pid = dt_pid_get_pid(pdp, dtp, NULL, dpr);
735                         if (pid != dpr->dpr_pid)
736                                 continue;

738                         found = B_TRUE;

740                         pd = *pdp;

742                         if (gmatch(provname, pdp->dtpd_provider) != 0 &&
743                             dt_pid_create_pid_probes(&pd, dtp, NULL, dpr) != 0)
744                                 ret = 1;

746                         /*
747                          * If it's not strictly a pid provider, we might match
748                          * a USDT provider.
749                          */
750                         if (strcmp(provname, pdp->dtpd_provider) != 0 &&
751                             dt_pid_create_usdt_probes(&pd, dtp, NULL, dpr) != 0)
752                                 ret = 1;
753                 }
754         }

756         if (found) {
757                 /*
758                  * Give DTrace a shot to the ribs to get it to check
759                  * out the newly created probes.
760                  */
761                 (void) dt_ioctl(dtp, DTRACEIOC_ENABLE, NULL);
762         }

764         return (ret);
765 }

767 /*
768  * libdtrace has a backroom deal with us to ask us for type information on
769  * behalf of pid provider probes when fasttrap doesn't return any type
770  * information. Instead we'll look up the module and see if there is type
771  * information available. However, if there is no type information available due
772  * to a lack of CTF data, then we want to make sure that DTrace still carries on
773  * in face of that. As such we don't have a meaningful exit code about failure.
774  * We emit information about why we failed to the dtrace debug log so someone
775  * can figure it out by asking nicely for DTRACE_DEBUG.
776  */
777 void
778 dt_pid_get_types(dtrace_hdl_t *dtp, const dtrace_probedesc_t *pdp,
779     dtrace_argdesc_t *adp, int *nargs)
780 {
781         dt_module_t *dmp;
782         ctf_file_t *fp;
```

```
783         ctf_funcinfo_t f;
784         ctf_id_t argv[32];
785         GElf_Sym sym;
786         prsyminfo_t si;
787         struct ps_prochandle *p;
788         int i, args;
789         char buf[DTRACE_ARGTYPELEN];
790         const char *mptr;
791         char *eptr;
792         int ret = 0;
793         int argc = sizeof (argv) / sizeof (ctf_id_t);
794         Lmid_t lmid;

796         /* Set up a potential outcome */
797         args = *nargs;
798         *nargs = 0;

800         /*
801          * If we don't have an entry or return probe then we can just stop right
802          * now as we don't have arguments for offset probes.
803          */
804         if (strcmp(pdp->dtpd_name, "entry") != 0 &&
805             strcmp(pdp->dtpd_name, "return") != 0)
806                 return;

808         dmp = dt_module_create(dtp, pdp->dtpd_provider);
809         if (dmp == NULL) {
810                 dt_dprintf("failed to find module for %s\n",
811                     pdp->dtpd_provider);
812                 return;
813         }
814         if (dt_module_load(dtp, dmp) != 0) {
815                 dt_dprintf("failed to load module for %s\n",
816                     pdp->dtpd_provider);
817                 return;
818         }

820         /*
821          * We may be working with a module that doesn't have ctf. If that's the
822          * case then we just return now and move on with life.
823          */
824         fp = dt_module_getctflib(dtp, dmp, pdp->dtpd_mod);
825         if (fp == NULL) {
826                 dt_dprintf("no ctf container for  %s\n",
827                     pdp->dtpd_mod);
828                 return;
829         }
830         p = dt_proc_grab(dtp, dmp->dm_pid, 0, PGRAB_RDONLY | PGRAB_FORCE);
831         if (p == NULL) {
832                 dt_dprintf("failed to grab pid\n");
833                 return;
834         }
835         dt_proc_lock(dtp, p);

837         /*
838          * Check to see if the D module has a link map ID and separate that out
839          * for properly interrogating libproc.
840          */
841         if ((mptr = strchr(pdp->dtpd_mod, '`')) != NULL) {
842                 if (strlen(pdp->dtpd_mod) < 3) {
843                         dt_dprintf("found weird modname with linkmap, "
844                             "aborting: %s\n", pdp->dtpd_mod);
845                         goto out;
846                 }
847                 if (pdp->dtpd_mod[0] != 'L' || pdp->dtpd_mod[1] != 'M') {
848                         dt_dprintf("missing leading 'LM', "
```



```
849                             "aborting: %s\n", pdp->dtpd_mod);
850                         goto out;
851                 }
852                 errno = 0;
853                 lmid = strtol(pdp->dtpd_mod + 2, &eptr, 16);
854                 if (errno == ERANGE || eptr != mptr) {
855                         dt_dprintf("failed to parse out lmid, aborting: %s\n",
856                             pdp->dtpd_mod);
857                         goto out;
858                 }
859                 mptr++;
860         } else {
861                 mptr = pdp->dtpd_mod;
862                 lmid = 0;
863         }

865         if (Pxlookup_by_name(p, lmid, mptr, pdp->dtpd_func,
866             &sym, &si) != 0) {
867                 dt_dprintf("failed to find function %s in %s`%s\n",
868                     pdp->dtpd_func, pdp->dtpd_provider, pdp->dtpd_mod);
869                 goto out;
870         }
871         if (ctf_func_info(fp, si.prs_id, &f) == CTF_ERR) {
872                 dt_dprintf("failed to get ctf information for %s in %s`%s\n",
873                     pdp->dtpd_func, pdp->dtpd_provider, pdp->dtpd_mod);
874                 goto out;
875         }

877         (void) snprintf(buf, sizeof (buf), "%s`%s", pdp->dtpd_provider,
878             pdp->dtpd_mod);

880         if (strcmp(pdp->dtpd_name, "return") == 0) {
881                 if (args < 2)
882                         goto out;

884                 bzero(adp, sizeof (dtrace_argdesc_t));
885                 adp->dtargd_ndx = 0;
886                 adp->dtargd_id = pdp->dtpd_id;
887                 adp->dtargd_mapping = adp->dtargd_ndx;
888                 /*
889                  * We explicitly leave out the library here, we only care that
890                  * it is some int. We are assuming that there is no ctf
891                  * container in here that is lying about what an int is.
892                  */
893                 (void) snprintf(adp->dtargd_native, DTRACE_ARGTYPELEN,
894                     "user %s`%s", pdp->dtpd_provider, "int");
895                 adp++;
896                 bzero(adp, sizeof (dtrace_argdesc_t));
897                 adp->dtargd_ndx = 1;
898                 adp->dtargd_id = pdp->dtpd_id;
899                 adp->dtargd_mapping = adp->dtargd_ndx;
900                 ret = snprintf(adp->dtargd_native, DTRACE_ARGTYPELEN,
901                     "userland ");
902                 (void) ctf_type_qname(fp, f.ctc_return, adp->dtargd_native +
903                     ret, DTRACE_ARGTYPELEN - ret, buf);
904                 *nargs = 2;
905         } else {
906                 if (ctf_func_args(fp, si.prs_id, argc, argv) == CTF_ERR)
907                         goto out;

909                 *nargs = MIN(args, f.ctc_argc);
910                 for (i = 0; i < *nargs; i++, adp++) {
911                         bzero(adp, sizeof (dtrace_argdesc_t));
912                         adp->dtargd_ndx = i;
913                         adp->dtargd_id = pdp->dtpd_id;
914                         adp->dtargd_mapping = adp->dtargd_ndx;
```

```
915                         ret = snprintf(adp->dtargd_native, DTRACE_ARGTYPELEN,
916                             "userland ");
917                         (void) ctf_type_qname(fp, argv[i], adp->dtargd_native +
918                             ret, DTRACE_ARGTYPELEN - ret, buf);
919                 }
920         }
921 out:
922         dt_proc_unlock(dtp, p);
923         dt_proc_release(dtp, p);
924 }
925 #endif /* ! codereview */
```

```
**********************************************************
    2080 Tue Jan 14 16:48:59 2014
new/usr/src/lib/libdtrace/common/dt_pid.h
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013, Joyent, Inc.  All rights reserved.
 */
#endif /* ! codereview */

#ifndef _DT_PID_H
#define _DT_PID_H

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

#include <libproc.h>
#include <sys/fasttrap.h>
#include <dt_impl.h>

#ifdef  __cplusplus
extern "C" {
#endif

#define DT_PROC_ERR     (-1)
#define DT_PROC_ALIGN   (-2)

extern int dt_pid_create_probes(dtrace_probedesc_t *, dtrace_hdl_t *,
    dt_pcb_t *pcb);
extern int dt_pid_create_probes_module(dtrace_hdl_t *, dt_proc_t *);

extern int dt_pid_create_entry_probe(struct ps_prochandle *, dtrace_hdl_t *,
    fasttrap_probe_spec_t *, const GElf_Sym *);

extern int dt_pid_create_return_probe(struct ps_prochandle *, dtrace_hdl_t *,
    fasttrap_probe_spec_t *, const GElf_Sym *, uint64_t *);
```



```
extern int dt_pid_create_offset_probe(struct ps_prochandle *, dtrace_hdl_t *,
    fasttrap_probe_spec_t *, const GElf_Sym *, ulong_t);

extern int dt_pid_create_glob_offset_probes(struct ps_prochandle *,
    dtrace_hdl_t *, fasttrap_probe_spec_t *, const GElf_Sym *, const char *);

extern void dt_pid_get_types(dtrace_hdl_t *, const dtrace_probedesc_t *,
    dtrace_argdesc_t *, int *);

#endif /* ! codereview */
#ifdef  __cplusplus
}
#endif

#endif  /* _DT_PID_H */
```

```
**********************************************************
   19075 Tue Jan 14 16:48:59 2014
new/usr/src/lib/libdtrace/common/dt_print.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
______unchanged_portion_omitted_

 640 /*
 641  * Main print function invoked by dt_consume_cpu().
 642  */
 643 int
 644 dtrace_print(dtrace_hdl_t *dtp, FILE *fp, const char *typename,
 645     caddr_t addr, size_t len)
 646 {
 647         const char *s;
 648         char *object;
 649         dt_printarg_t pa;
 650         ctf_id_t id;
 651         dt_module_t *dmp;
 652         ctf_file_t *ctfp;
 653         int libid;
 654 #endif /* ! codereview */

 656         /*
 657          * Split the fully-qualified type ID (module'id).  This should
 658          * always be the format, but if for some reason we don't find the
 659          * expected value, return 0 to fall back to the generic trace()
 660          * behavior. In the case of userland CTF modules this will actually be
 661          * of the format (module'lib'id). This is due to the fact that those
 662          * modules have multiple CTF containers which 'lib' identifies.
 652          * behavior.
 663          */
 664         for (s = typename; *s != '\0' && *s != ''; s++)
 665                 ;

 667         if (*s != '')
 668                 return (0);

 670         object = alloca(s - typename + 1);
 671         bcopy(typename, object, s - typename);
 672         object[s - typename] = '\0';
 673         dmp = dt_module_lookup_by_name(dtp, object);
 674         if (dmp == NULL)
 675                 return (0);

 677         if (dmp->dm_pid != 0) {
 678                 libid = atoi(s + 1);
 679                 s = strchr(s + 1, '');
 680                 if (s == NULL || libid > dmp->dm_nctflibs)
 681                         return (0);
 682                 ctfp = dmp->dm_libctfp[libid];
 683         } else {
 684                 ctfp = dt_module_getctf(dtp, dmp);
 685         }

 687 #endif /* ! codereview */
 688         id = atoi(s + 1);

 690         /*
 691          * Try to get the CTF kind for this id.  If something has gone horribly
 692          * wrong and we can't resolve the ID, bail out and let trace() do the
```



```
 693          * work.
 694          */
 695         if (ctfp == NULL || ctf_type_kind(ctfp, id) == CTF_ERR)
 663         dmp = dt_module_lookup_by_name(dtp, object);
 664         if (dmp == NULL || ctf_type_kind(dt_module_getctf(dtp, dmp),
 665             id) == CTF_ERR) {
 696                 return (0);
 667         }

 698         /* setup the print structure and kick off the main print routine */
 699         pa.pa_dtp = dtp;
 700         pa.pa_addr = addr;
 701         pa.pa_ctfp = ctfp;
 672         pa.pa_ctfp = dt_module_getctf(dtp, dmp);
 702         pa.pa_nest = 0;
 703         pa.pa_depth = 0;
 704         pa.pa_file = fp;
 705         (void) ctf_type_visit(pa.pa_ctfp, id, dt_print_member, &pa);

 707         dt_print_trailing_braces(&pa, 0);

 709         return (len);
 710 }
______unchanged_portion_omitted_
```

```
**********************************************************
   53312 Tue Jan 14 16:49:00 2014
new/usr/src/lib/libdtrace/common/dt_printf.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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, Joyent, Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2011, Joyent, Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

#include <sys/sysmacros.h>
#include <strings.h>
#include <stdlib.h>
#include <alloca.h>
#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <limits.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <arpa/nameser.h>

#include <dt_printf.h>
#include <dt_string.h>
#include <dt_impl.h>

/*ARGSUSED*/
static int
pfcheck_addr(dt_pfargv_t *pfv, dt_pfargd_t *pfd, dt_node_t *dnp)
{
        return (dt_node_is_pointer(dnp) || dt_node_is_integer(dnp));
}
```

_____*unchanged_portion_omitted_*

```
void
```



```
dt_printf_validate(dt_pfargv_t *pfv, uint_t flags,
    dt_ident_t *idp, int foff, dtrace_actkind_t kind, dt_node_t *dnp)
{
        dt_pfargd_t *pfd = pfv->pfv_argv;
        const char *func = idp->di_name;

        char n[DT_TYPE_NAMELEN];
        dtrace_typeinfo_t dtt;
        const char *aggtype;
        dt_node_t aggnode;
        int i, j;

        if (pfv->pfv_format[0] == '\0') {
                xyerror(D_PRINTF_FMT_EMPTY,
                    "%s( ) format string is empty\n", func);
        }

        pfv->pfv_flags = flags;

        /*
         * We fake up a parse node representing the type that can be used with
         * an aggregation result conversion, which -- for all but count() --
         * is a signed quantity.
         */
        if (kind != DTRACEAGG_COUNT)
                aggtype = "int64_t";
        else
                aggtype = "uint64_t";

        if (dt_type_lookup(aggtype, &dtt) != 0)
                xyerror(D_TYPE_ERR, "failed to lookup agg type %s\n", aggtype);

        bzero(&aggnode, sizeof (aggnode));
        dt_node_type_assign(&aggnode, dtt.dtt_ctfp, dtt.dtt_type, B_FALSE);
        dt_node_type_assign(&aggnode, dtt.dtt_ctfp, dtt.dtt_type);

        for (i = 0, j = 0; i < pfv->pfv_argc; i++, pfd = pfd->pfd_next) {
                const dt_pfconv_t *pfc = pfd->pfd_conv;
                const char *dyns[2];
                int dync = 0;

                char vname[64];
                dt_node_t *vnp;

                if (pfc == NULL)
                        continue; /* no checking if argd is just a prefix */

                if (pfc->pfc_print == &pfprint_pct) {
                        (void) strcat(pfd->pfd_fmt, pfc->pfc_ofmt);
                        continue;
                }

                if (pfd->pfd_flags & DT_PFCONV_DYNPREC)
                        dyns[dync++] = ".*";
                if (pfd->pfd_flags & DT_PFCONV_DYNWIDTH)
                        dyns[dync++] = "*";

                for (; dync != 0; dync--) {
                        if (dnp == NULL) {
                                xyerror(D_PRINTF_DYN_PROTO,
                                    "%s( ) prototype mismatch: conversion "
                                    "#%d (%%%s) is missing a corresponding "
                                    "\"%s\" argument\n", func, i + 1,
                                    pfc->pfc_name, dyns[dync - 1]);
                        }
```

```
1083                        if (dt_node_is_integer(dnp) == 0) {
1084                                xyerror(D_PRINTF_DYN_TYPE,
1085                                    "%s( ) argument #%d is incompatible "
1086                                    "with conversion #%d prototype:\n"
1087                                    "\tconversion: %% %s %s\n"
1088                                    "\t prototype: int\n\t  argument: %s\n",
1089                                    func, j + foff + 1, i + 1,
1090                                    dyns[dync - 1], pfc->pfc_name,
1091                                    dt_node_type_name(dnp, n, sizeof (n)));
1092                        }

1094                        dnp = dnp->dn_list;
1095                        j++;
1096                }

1098                /*
1099                 * If this conversion is consuming the aggregation data, set
1100                 * the value node pointer (vnp) to a fake node based on the
1101                 * aggregating function result type.  Otherwise assign vnp to
1102                 * the next parse node in the argument list, if there is one.
1103                 */
1104                if (pfd->pfd_flags & DT_PFCONV_AGG) {
1105                        if (!(flags & DT_PRINTF_AGGREGATION)) {
1106                                xyerror(D_PRINTF_AGG_CONV,
1107                                    "%%@ conversion requires an aggregation"
1108                                    " and is not for use with %s( )\n", func);
1109                        }
1110                        (void) strlcpy(vname, "aggregating action",
1111                            sizeof (vname));
1112                        vnp = &aggnode;
1113                } else if (dnp == NULL) {
1114                        xyerror(D_PRINTF_ARG_PROTO,
1115                            "%s( ) prototype mismatch: conversion #%d (%%"
1116                            "%s) is missing a corresponding value argument\n",
1117                            func, i + 1, pfc->pfc_name);
1118                } else {
1119                        (void) snprintf(vname, sizeof (vname),
1120                            "argument #%d", j + foff + 1);
1121                        vnp = dnp;
1122                        dnp = dnp->dn_list;
1123                        j++;
1124                }

1126                /*
1127                 * Fill in the proposed final format string by prepending any
1128                 * size-related prefixes to the pfconv's format string.  The
1129                 * pfc_check() function below may optionally modify the format
1130                 * as part of validating the type of the input argument.
1131                 */
1132                if (pfc->pfc_print == &pfprint_sint ||
1133                    pfc->pfc_print == &pfprint_uint ||
1134                    pfc->pfc_print == &pfprint_dint) {
1135                        if (dt_node_type_size(vnp) == sizeof (uint64_t))
1136                                (void) strcpy(pfd->pfd_fmt, "ll");
1137                } else if (pfc->pfc_print == &pfprint_fp) {
1138                        if (dt_node_type_size(vnp) == sizeof (long double))
1139                                (void) strcpy(pfd->pfd_fmt, "L");
1140                }

1142                (void) strcat(pfd->pfd_fmt, pfc->pfc_ofmt);

1144                /*
1145                 * Validate the format conversion against the value node type.
1146                 * If the conversion is good, create the descriptor format
1147                 * string by concatenating together any required printf(3C)
1148                 * size prefixes with the conversion's native format string.
```



```
1149                 */
1150                if (pfc->pfc_check(pfv, pfd, vnp) == 0) {
1151                        xyerror(D_PRINTF_ARG_TYPE,
1152                            "%s( ) %s is incompatible with "
1153                            "conversion #%d prototype:\n\tconversion: %%%s\n"
1154                            "\t prototype: %s\n\t  argument: %s\n", func,
1155                            vname, i + 1, pfc->pfc_name, pfc->pfc_tstr,
1156                            dt_node_type_name(vnp, n, sizeof (n)));
1157                }
1158        }

1160        if ((flags & DT_PRINTF_EXACTLEN) && dnp != NULL) {
1161                xyerror(D_PRINTF_ARG_EXTRA,
1162                    "%s( ) prototype mismatch: only %d arguments "
1163                    "required by this format string\n", func, j);
1164        }
1165 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   23597 Tue Jan 14 16:49:00 2014
new/usr/src/lib/libdtrace/common/dt_provider.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 */

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

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

#include <assert.h>
#include <limits.h>
#include <strings.h>
#include <stdlib.h>
#include <alloca.h>
#include <unistd.h>
#include <errno.h>

#include <dt_provider.h>
#include <dt_module.h>
#include <dt_string.h>
#include <dt_list.h>
#include <dt_pid.h>
#include <dtrace.h>
#endif /* ! codereview */

static dt_provider_t *
dt_provider_insert(dtrace_hdl_t *dtp, dt_provider_t *pvp, uint_t h)
{
        dt_list_append(&dtp->dt_provlist, pvp);

        pvp->pv_next = dtp->dt_provs[h];
```



```
        dtp->dt_provs[h] = pvp;
        dtp->dt_nprovs++;

        return (pvp);
}

dt_provider_t *
dt_provider_lookup(dtrace_hdl_t *dtp, const char *name)
{
        uint_t h = dt_strtab_hash(name, NULL) % dtp->dt_provbuckets;
        dtrace_providerdesc_t desc;
        dt_provider_t *pvp;

        for (pvp = dtp->dt_provs[h]; pvp != NULL; pvp = pvp->pv_next) {
                if (strcmp(pvp->pv_desc.dtvd_name, name) == 0)
                        return (pvp);
        }

        if (strisglob(name) || name[0] == '\0') {
                (void) dt_set_errno(dtp, EDT_NOPROV);
                return (NULL);
        }

        bzero(&desc, sizeof (desc));
        (void) strlcpy(desc.dtvd_name, name, DTRACE_PROVNAMELEN);

        if (dt_ioctl(dtp, DTRACEIOC_PROVIDER, &desc) == -1) {
                (void) dt_set_errno(dtp, errno == ESRCH ? EDT_NOPROV : errno);
                return (NULL);
        }

        if ((pvp = dt_provider_create(dtp, name)) == NULL)
                return (NULL); /* dt_errno is set for us */

        bcopy(&desc, &pvp->pv_desc, sizeof (desc));
        pvp->pv_flags |= DT_PROVIDER_IMPL;
        return (pvp);
}

dt_provider_t *
dt_provider_create(dtrace_hdl_t *dtp, const char *name)
{
        dt_provider_t *pvp;

        if ((pvp = dt_zalloc(dtp, sizeof (dt_provider_t))) == NULL)
                return (NULL);

        (void) strlcpy(pvp->pv_desc.dtvd_name, name, DTRACE_PROVNAMELEN);
        pvp->pv_probes = dt_idhash_create(pvp->pv_desc.dtvd_name, NULL, 0, 0);
        pvp->pv_gen = dtp->dt_gen;
        pvp->pv_hdl = dtp;

        if (pvp->pv_probes == NULL) {
                dt_free(dtp, pvp);
                (void) dt_set_errno(dtp, EDT_NOMEM);
                return (NULL);
        }

        pvp->pv_desc.dtvd_attr.dtpa_provider = _dtrace_prvattr;
        pvp->pv_desc.dtvd_attr.dtpa_mod = _dtrace_prvattr;
        pvp->pv_desc.dtvd_attr.dtpa_func = _dtrace_prvattr;
        pvp->pv_desc.dtvd_attr.dtpa_name = _dtrace_prvattr;
        pvp->pv_desc.dtvd_attr.dtpa_args = _dtrace_prvattr;

        return (dt_provider_insert(dtp, pvp,
            dt_strtab_hash(name, NULL) % dtp->dt_provbuckets));
```

```
121 }
123 void
124 dt_provider_destroy(dtrace_hdl_t *dtp, dt_provider_t *pvp)
125 {
126         dt_provider_t **pp;
127         uint_t h;
129         assert(pvp->pv_hdl == dtp);
131         h = dt_strtab_hash(pvp->pv_desc.dtvd_name, NULL) % dtp->dt_provbuckets;
132         pp = &dtp->dt_provs[h];
134         while (*pp != NULL && *pp != pvp)
135                 pp = &(*pp)->pv_next;
137         assert(*pp != NULL && *pp == pvp);
138         *pp = pvp->pv_next;
140         dt_list_delete(&dtp->dt_provlist, pvp);
141         dtp->dt_nprovs--;
143         if (pvp->pv_probes != NULL)
144                 dt_idhash_destroy(pvp->pv_probes);
146         dt_node_link_free(&pvp->pv_nodes);
147         dt_free(dtp, pvp->pv_xrefs);
148         dt_free(dtp, pvp);
149 }
151 int
152 dt_provider_xref(dtrace_hdl_t *dtp, dt_provider_t *pvp, id_t id)
153 {
154         size_t oldsize = BT_SIZEOFMAP(pvp->pv_xrmax);
155         size_t newsize = BT_SIZEOFMAP(dtp->dt_xlatorid);
157         assert(id >= 0 && id < dtp->dt_xlatorid);
159         if (newsize > oldsize) {
160                 ulong_t *xrefs = dt_zalloc(dtp, newsize);
162                 if (xrefs == NULL)
163                         return (-1);
165                 bcopy(pvp->pv_xrefs, xrefs, oldsize);
166                 dt_free(dtp, pvp->pv_xrefs);
168                 pvp->pv_xrefs = xrefs;
169                 pvp->pv_xrmax = dtp->dt_xlatorid;
170         }
172         BT_SET(pvp->pv_xrefs, id);
173         return (0);
174 }
176 static uint8_t
177 dt_probe_argmap(dt_node_t *xnp, dt_node_t *nnp)
178 {
179         uint8_t i;
181         for (i = 0; nnp != NULL; i++) {
182                 if (nnp->dn_string != NULL &&
183                     strcmp(nnp->dn_string, xnp->dn_string) == 0)
184                         break;
185                 else
186                         nnp = nnp->dn_list;
```



```
187         }
189         return (i);
190 }
192 static dt_node_t *
193 dt_probe_alloc_args(dt_provider_t *pvp, int argc)
194 {
195         dt_node_t *args = NULL, *pnp = NULL, *dnp;
196         int i;
198         for (i = 0; i < argc; i++, pnp = dnp) {
199                 if ((dnp = dt_node_xalloc(pvp->pv_hdl, DT_NODE_TYPE)) == NULL)
200                         return (NULL);
202                 dnp->dn_link = pvp->pv_nodes;
203                 pvp->pv_nodes = dnp;
205                 if (args == NULL)
206                         args = dnp;
207                 else
208                         pnp->dn_list = dnp;
209         }
211         return (args);
212 }
214 static size_t
215 dt_probe_keylen(const dtrace_probedesc_t *pdp)
216 {
217         return (strlen(pdp->dtpd_mod) + 1 +
218             strlen(pdp->dtpd_func) + 1 + strlen(pdp->dtpd_name) + 1);
219 }
221 static char *
222 dt_probe_key(const dtrace_probedesc_t *pdp, char *s)
223 {
224         (void) snprintf(s, INT_MAX, "%s:%s:%s",
225             pdp->dtpd_mod, pdp->dtpd_func, pdp->dtpd_name);
226         return (s);
227 }
229 /*
230  * If a probe was discovered from the kernel, ask dtrace(7D) for a description
231  * of each of its arguments, including native and translated types.
232  */
233 static dt_probe_t *
234 dt_probe_discover(dt_provider_t *pvp, const dtrace_probedesc_t *pdp)
235 {
236         dtrace_hdl_t *dtp = pvp->pv_hdl;
237         char *name = dt_probe_key(pdp, alloca(dt_probe_keylen(pdp)));
239         dt_node_t *xargs, *nargs;
240         dt_ident_t *idp;
241         dt_probe_t *prp;
243         dtrace_typeinfo_t dtt;
244         int i, nc, xc;
246         int adc = _dtrace_argmax;
247         dtrace_argdesc_t *adv = alloca(sizeof (dtrace_argdesc_t) * adc);
248         dtrace_argdesc_t *adp = adv;
250         assert(strcmp(pvp->pv_desc.dtvd_name, pdp->dtpd_provider) == 0);
251         assert(pdp->dtpd_id != DTRACE_IDNONE);
```

```
253         dt_dprintf("discovering probe %s:%s id=%d\n",
254             pvp->pv_desc.dtvd_name, name, pdp->dtpd_id);
255
256         for (nc = -1, i = 0; i < adc; i++, adp++) {
257                 bzero(adp, sizeof (dtrace_argdesc_t));
258                 adp->dtargd_ndx = i;
259                 adp->dtargd_id = pdp->dtpd_id;
260
261                 if (dt_ioctl(dtp, DTRACEIOC_PROBEARG, adp) != 0) {
262                         (void) dt_set_errno(dtp, errno);
263                         return (NULL);
264                 }
265
266                 if (adp->dtargd_ndx == DTRACE_ARGNONE)
267                         break; /* all argument descs have been retrieved */
268
269                 nc = MAX(nc, adp->dtargd_mapping);
270         }
271
272         xc = i;
273         nc++;
274
275         /*
276          * The pid provider believes in giving the kernel a break. No reason to
277          * give the kernel all the ctf containers that we're keeping ourselves
278          * just to get it back from it. So if we're coming from a pid provider
279          * probe and the kernel gave us no argument information we'll get some
280          * here. If for some crazy reason the kernel knows about our userland
281          * types then we just ignore this.
282          */
283         if (xc == 0 && nc == 0 &&
284             strncmp(pvp->pv_desc.dtvd_name, "pid", 3) == 0) {
285                 nc = adc;
286                 dt_pid_get_types(dtp, pdp, adv, &nc);
287                 xc = nc;
288         }
289
290         /*
291 #endif /* ! codereview */
292          * Now that we have discovered the number of native and translated
293          * arguments from the argument descriptions, allocate a new probe ident
294          * and corresponding dt_probe_t and hash it into the provider.
295          */
296         xargs = dt_probe_alloc_args(pvp, xc);
297         nargs = dt_probe_alloc_args(pvp, nc);
298
299         if ((xc != 0 && xargs == NULL) || (nc != 0 && nargs == NULL))
300                 return (NULL); /* dt_errno is set for us */
301
302         idp = dt_ident_create(name, DT_IDENT_PROBE,
303             DT_IDFLG_ORPHAN, pdp->dtpd_id, _dtrace_defattr, 0,
304             &dt_idops_probe, NULL, dtp->dt_gen);
305
306         if (idp == NULL) {
307                 (void) dt_set_errno(dtp, EDT_NOMEM);
308                 return (NULL);
309         }
310
311         if ((prp = dt_probe_create(dtp, idp, 2,
312             nargs, nc, xargs, xc)) == NULL) {
313                 dt_ident_destroy(idp);
314                 return (NULL);
315         }
316
317         dt_probe_declare(pvp, prp);
```



```
319         /*
320          * Once our new dt_probe_t is fully constructed, iterate over the
321          * cached argument descriptions and assign types to prp->pr_nargv[]
322          * and prp->pr_xargv[] and assign mappings to prp->pr_mapping[].
323          */
324         for (adp = adv, i = 0; i < xc; i++, adp++) {
325                 if (dtrace_type_strcompile(dtp,
326                     adp->dtargd_native, &dtt) != 0) {
327                         dt_dprintf("failed to resolve input type %s "
328                             "for %s:%s arg #%d: %s\n", adp->dtargd_native,
329                             pvp->pv_desc.dtvd_name, name, i + 1,
330                             dtrace_errmsg(dtp, dtrace_errno(dtp)));
331
332                         dtt.dtt_object = NULL;
333                         dtt.dtt_ctfp = NULL;
334                         dtt.dtt_type = CTF_ERR;
335                 } else {
336                         dt_node_type_assign(prp->pr_nargv[adp->dtargd_mapping],
337                             dtt.dtt_ctfp, dtt.dtt_type,
338                             dtt.dtt_flags & DTT_FL_USER ? B_TRUE : B_FALSE);
 44                          dtt.dtt_ctfp, dtt.dtt_type);
339                 }
340
341                 if (dtt.dtt_type != CTF_ERR && (adp->dtargd_xlate[0] == '\0' ||
342                     strcmp(adp->dtargd_native, adp->dtargd_xlate) == 0)) {
343                         dt_node_type_propagate(prp->pr_nargv[
344                             adp->dtargd_mapping], prp->pr_xargv[i]);
345                 } else if (dtrace_type_strcompile(dtp,
346                     adp->dtargd_xlate, &dtt) != 0) {
347                         dt_dprintf("failed to resolve output type %s "
348                             "for %s:%s arg #%d: %s\n", adp->dtargd_xlate,
349                             pvp->pv_desc.dtvd_name, name, i + 1,
350                             dtrace_errmsg(dtp, dtrace_errno(dtp)));
351
352                         dtt.dtt_object = NULL;
353                         dtt.dtt_ctfp = NULL;
354                         dtt.dtt_type = CTF_ERR;
355                 } else {
356                         dt_node_type_assign(prp->pr_xargv[i],
357                             dtt.dtt_ctfp, dtt.dtt_type, B_FALSE);
 63                          dtt.dtt_ctfp, dtt.dtt_type);
358                 }
359
360                 prp->pr_mapping[i] = adp->dtargd_mapping;
361                 prp->pr_argv[i] = dtt;
362         }
363
364         return (prp);
365 }
_____unchanged_portion_omitted_

621 /*
622  * Lookup the dynamic translator type tag for the specified probe argument and
623  * assign the type to the specified node.  If the type is not yet defined, add
624  * it to the "D" module's type container as a typedef for an unknown type.
625  */
626 dt_node_t *
627 dt_probe_tag(dt_probe_t *prp, uint_t argn, dt_node_t *dnp)
628 {
629         dtrace_hdl_t *dtp = prp->pr_pvp->pv_hdl;
630         dtrace_typeinfo_t dtt;
631         size_t len;
632         char *tag;
633
634         len = snprintf(NULL, 0, "__dtrace_%s___%s_arg%u",
635             prp->pr_pvp->pv_desc.dtvd_name, prp->pr_name, argn);
```

```
637         tag = alloca(len + 1);

639         (void) snprintf(tag, len + 1, "__dtrace_%s___%s_arg%u",
640             prp->pr_pvp->pv_desc.dtvd_name, prp->pr_name, argn);

642         if (dtrace_lookup_by_type(dtp, DTRACE_OBJ_DDEFS, tag, &dtt) != 0) {
643                 dtt.dtt_object = DTRACE_OBJ_DDEFS;
644                 dtt.dtt_ctfp = DT_DYN_CTFP(dtp);
645                 dtt.dtt_type = ctf_add_typedef(DT_DYN_CTFP(dtp),
646                     CTF_ADD_ROOT, tag, DT_DYN_TYPE(dtp));

648                 if (dtt.dtt_type == CTF_ERR ||
649                     ctf_update(dtt.dtt_ctfp) == CTF_ERR) {
650                         xyerror(D_UNKNOWN, "cannot define type %s: %s\n",
651                             tag, ctf_errmsg(ctf_errno(dtt.dtt_ctfp)));
652                 }
653         }

655         bzero(dnp, sizeof (dt_node_t));
656         dnp->dn_kind = DT_NODE_TYPE;

658         dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type, B_FALSE);
364         dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type);
659         dt_node_attr_assign(dnp, _dtrace_defattr);

661         return (dnp);
662 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   11557 Tue Jan 14 16:49:01 2014
new/usr/src/lib/libdtrace/common/dt_xlator.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 * Copyright (c) 2013 Joyent, Inc. All rights reserved.
 */

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

#include <strings.h>
#include <assert.h>

#include <dt_xlator.h>
#include <dt_parser.h>
#include <dt_grammar.h>
#include <dt_module.h>
#include <dt_impl.h>

/*
 * Create a member node corresponding to one of the output members of a dynamic
 * translator.  We set the member's dn_membexpr to a DT_NODE_XLATOR node that
 * has dn_op set to DT_TOK_XLATE and refers back to the translator itself.  The
 * code generator will then use this as the indicator for dynamic translation.
 */
/*ARGSUSED*/
static int
dt_xlator_create_member(const char *name, ctf_id_t type, ulong_t off, void *arg)
{
        dt_xlator_t *dxp = arg;
        dtrace_hdl_t *dtp = dxp->dx_hdl;
        dt_node_t *enp, *mnp;

        if ((enp = dt_node_xalloc(dtp, DT_NODE_XLATOR)) == NULL)
```



```
                return (dt_set_errno(dtp, EDT_NOMEM));

        enp->dn_link = dxp->dx_nodes;
        dxp->dx_nodes = enp;

        if ((mnp = dt_node_xalloc(dtp, DT_NODE_MEMBER)) == NULL)
                return (dt_set_errno(dtp, EDT_NOMEM));

        mnp->dn_link = dxp->dx_nodes;
        dxp->dx_nodes = mnp;

        /*
         * For the member expression, we use a DT_NODE_XLATOR/TOK_XLATE whose
         * xlator refers back to the translator and whose dn_xmember refers to
         * the current member.  These refs will be used by dt_cg.c and dt_as.c.
         */
        enp->dn_op = DT_TOK_XLATE;
        enp->dn_xlator = dxp;
        enp->dn_xmember = mnp;
        dt_node_type_assign(enp, dxp->dx_dst_ctfp, type, B_FALSE);
        dt_node_type_assign(enp, dxp->dx_dst_ctfp, type);

        /*
         * For the member itself, we use a DT_NODE_MEMBER as usual with the
         * appropriate name, output type, and member expression set to 'enp'.
         */
        if (dxp->dx_members != NULL) {
                assert(enp->dn_link->dn_kind == DT_NODE_MEMBER);
                enp->dn_link->dn_list = mnp;
        } else
                dxp->dx_members = mnp;

        mnp->dn_membname = strdup(name);
        mnp->dn_membexpr = enp;
        dt_node_type_assign(mnp, dxp->dx_dst_ctfp, type, B_FALSE);
        dt_node_type_assign(mnp, dxp->dx_dst_ctfp, type);

        if (mnp->dn_membname == NULL)
                return (dt_set_errno(dtp, EDT_NOMEM));

        return (0);
}
```

*unchanged_portion_omitted_*

```
dt_xlator_t *
dt_xlator_lookup(dtrace_hdl_t *dtp, dt_node_t *src, dt_node_t *dst, int flags)
{
        ctf_file_t *src_ctfp = src->dn_ctfp;
        ctf_id_t src_type = src->dn_type;
        ctf_id_t src_base = ctf_type_resolve(src_ctfp, src_type);

        ctf_file_t *dst_ctfp = dst->dn_ctfp;
        ctf_id_t dst_type = dst->dn_type;
        ctf_id_t dst_base = ctf_type_resolve(dst_ctfp, dst_type);
        uint_t dst_kind = ctf_type_kind(dst_ctfp, dst_base);

        int ptr = dst_kind == CTF_K_POINTER;
        dtrace_typeinfo_t src_dtt, dst_dtt;
        dt_node_t xn = { 0 };
        dt_xlator_t *dxp = NULL;

        if (src_base == CTF_ERR || dst_base == CTF_ERR)
                return (NULL); /* fail if these are unresolvable types */

        /*
         * Translators are always defined using a struct or union type, so if
```

```
279          * we are attempting to translate to type "T *", we internally look
280          * for a translation to type "T" by following the pointer reference.
281          */
282         if (ptr) {
283                 dst_type = ctf_type_reference(dst_ctfp, dst_type);
284                 dst_base = ctf_type_resolve(dst_ctfp, dst_type);
285                 dst_kind = ctf_type_kind(dst_ctfp, dst_base);
286         }

288         if (dst_kind != CTF_K_UNION && dst_kind != CTF_K_STRUCT)
289                 return (NULL); /* fail if the output isn't a struct or union */

291         /*
292          * In order to find a matching translator, we iterate over the set of
293          * available translators in three passes.  First, we look for a
294          * translation from the exact source type to the resolved destination.
295          * Second, we look for a translation from the resolved source type to
296          * the resolved destination.  Third, we look for a translation from a
297          * compatible source type (using the same rules as parameter formals)
298          * to the resolved destination.  If all passes fail, return NULL.
299          */
300         for (dxp = dt_list_next(&dtp->dt_xlators); dxp != NULL;
301             dxp = dt_list_next(dxp)) {
302                 if (ctf_type_compat(dxp->dx_src_ctfp, dxp->dx_src_type,
303                     src_ctfp, src_type) &&
304                     ctf_type_compat(dxp->dx_dst_ctfp, dxp->dx_dst_base,
305                     dst_ctfp, dst_base))
306                         goto out;
307         }

309         if (flags & DT_XLATE_EXACT)
310                 goto out; /* skip remaining passes if exact match required */

312         for (dxp = dt_list_next(&dtp->dt_xlators); dxp != NULL;
313             dxp = dt_list_next(dxp)) {
314                 if (ctf_type_compat(dxp->dx_src_ctfp, dxp->dx_src_base,
315                     src_ctfp, src_type) &&
316                     ctf_type_compat(dxp->dx_dst_ctfp, dxp->dx_dst_base,
317                     dst_ctfp, dst_base))
318                         goto out;
319         }

321         for (dxp = dt_list_next(&dtp->dt_xlators); dxp != NULL;
322             dxp = dt_list_next(dxp)) {
323                 dt_node_type_assign(&xn, dxp->dx_src_ctfp, dxp->dx_src_type,
324                     B_FALSE);
321                 dt_node_type_assign(&xn, dxp->dx_src_ctfp, dxp->dx_src_type);
325                 if (ctf_type_compat(dxp->dx_dst_ctfp, dxp->dx_dst_base,
326                     dst_ctfp, dst_base) && dt_node_is_argcompat(src, &xn))
327                         goto out;
328         }

330 out:
331         if (ptr && dxp != NULL && dxp->dx_ptrid.di_type == CTF_ERR)
332                 return (NULL);  /* no translation available to pointer type */

334         if (dxp != NULL || !(flags & DT_XLATE_EXTERN) ||
335             dtp->dt_xlatemode == DT_XL_STATIC)
336                 return (dxp);   /* we succeeded or not allowed to extern */

338         /*
339          * If we get here, then we didn't find an existing translator, but the
340          * caller and xlatemode permit us to create an extern to a dynamic one.
341          */
342         src_dtt.dtt_object = dt_module_lookup_by_ctf(dtp, src_ctfp)->dm_name;
343         src_dtt.dtt_ctfp = src_ctfp;
```



```
344         src_dtt.dtt_type = src_type;

346         dst_dtt.dtt_object = dt_module_lookup_by_ctf(dtp, dst_ctfp)->dm_name;
347         dst_dtt.dtt_ctfp = dst_ctfp;
348         dst_dtt.dtt_type = dst_type;

350         return (dt_xlator_create(dtp, &src_dtt, &dst_dtt, NULL, NULL, NULL));
351 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   23420 Tue Jan 14 16:49:01 2014
new/usr/src/lib/libdtrace/common/dtrace.h
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 * Copyright (c) 2011, Joyent, Inc. All rights reserved.
 */

#ifndef _DTRACE_H
#define _DTRACE_H

#include <sys/dtrace.h>
#include <stdarg.h>
#include <stdio.h>
#include <gelf.h>

#ifdef __cplusplus
extern "C" {
#endif

/*
 * DTrace Dynamic Tracing Software: Library Interfaces
 *
 * Note: The contents of this file are private to the implementation of the
 * Solaris system and DTrace subsystem and are subject to change at any time
 * without notice.  Applications and drivers using these interfaces will fail
 * to run on future releases.  These interfaces should not be used for any
 * purpose except those expressly outlined in dtrace(7D) and libdtrace(3LIB).
 * Please refer to the "Solaris Dynamic Tracing Guide" for more information.
 */

#define DTRACE_VERSION  3               /* library ABI interface version */
```



```
struct ps_prochandle;
typedef struct dtrace_hdl dtrace_hdl_t;
typedef struct dtrace_prog dtrace_prog_t;
typedef struct dtrace_vector dtrace_vector_t;
typedef struct dtrace_aggdata dtrace_aggdata_t;

#define DTRACE_O_NODEV          0x01    /* do not open dtrace(7D) device */
#define DTRACE_O_NOSYS          0x02    /* do not load /system/object modules */
#define DTRACE_O_LP64           0x04    /* force D compiler to be LP64 */
#define DTRACE_O_ILP32          0x08    /* force D compiler to be ILP32 */
#define DTRACE_O_MASK           0x0f    /* mask of valid flags to dtrace_open */

extern dtrace_hdl_t *dtrace_open(int, int, int *);
extern dtrace_hdl_t *dtrace_vopen(int, int, int *,
    const dtrace_vector_t *, void *);

extern int dtrace_go(dtrace_hdl_t *);
extern int dtrace_stop(dtrace_hdl_t *);
extern void dtrace_sleep(dtrace_hdl_t *);
extern void dtrace_close(dtrace_hdl_t *);

extern int dtrace_errno(dtrace_hdl_t *);
extern const char *dtrace_errmsg(dtrace_hdl_t *, int);
extern const char *dtrace_faultstr(dtrace_hdl_t *, int);
extern const char *dtrace_subrstr(dtrace_hdl_t *, int);

extern int dtrace_setopt(dtrace_hdl_t *, const char *, const char *);
extern int dtrace_getopt(dtrace_hdl_t *, const char *, dtrace_optval_t *);

extern void dtrace_update(dtrace_hdl_t *);
extern int dtrace_ctlfd(dtrace_hdl_t *);

/*
 * DTrace Program Interface
 *
 * DTrace programs can be created by compiling ASCII text files containing
 * D programs or by compiling in-memory C strings that specify a D program.
 * Once created, callers can examine the list of program statements and
 * enable the probes and actions described by these statements.
 */

typedef struct dtrace_proginfo {
        dtrace_attribute_t dpi_descattr; /* minimum probedesc attributes */
        dtrace_attribute_t dpi_stmtattr; /* minimum statement attributes */
        uint_t dpi_aggregates;  /* number of aggregates specified in program */
        uint_t dpi_recgens;     /* number of record generating probes in prog */
        uint_t dpi_matches;     /* number of probes matched by program */
        uint_t dpi_speculations; /* number of speculations specified in prog */
} dtrace_proginfo_t;
______unchanged_portion_omitted_

extern int dtrace_lookup_by_name(dtrace_hdl_t *, const char *, const char *,
    GElf_Sym *, dtrace_syminfo_t *);

extern int dtrace_lookup_by_addr(dtrace_hdl_t *, GElf_Addr addr,
    GElf_Sym *, dtrace_syminfo_t *);

typedef struct dtrace_typeinfo {
        const char *dtt_object;                 /* object containing type */
        ctf_file_t *dtt_ctfp;                   /* CTF container handle */
        ctf_id_t dtt_type;                      /* CTF type identifier */
        uint_t dtt_flags;                       /* Misc. flags */
#endif /* ! codereview */
} dtrace_typeinfo_t;
```

```
499 #define DTT_FL_USER     0x1                     /* user type */

501 #endif /* ! codereview */
502 extern int dtrace_lookup_by_type(dtrace_hdl_t *, const char *, const char *,
503     dtrace_typeinfo_t *);

505 extern int dtrace_symbol_type(dtrace_hdl_t *, const GElf_Sym *,
506     const dtrace_syminfo_t *, dtrace_typeinfo_t *);

508 extern int dtrace_type_strcompile(dtrace_hdl_t *,
509     const char *, dtrace_typeinfo_t *);

511 extern int dtrace_type_fcompile(dtrace_hdl_t *,
512     FILE *, dtrace_typeinfo_t *);

514 /*
515  * DTrace Probe Interface
516  *
517  * Library clients can use these functions to iterate over the set of available
518  * probe definitions and inquire as to their attributes.  The probe iteration
519  * interfaces report probes that are declared as well as those from dtrace(7D).
520  */
521 typedef struct dtrace_probeinfo {
522         dtrace_attribute_t dtp_attr;            /* name attributes */
523         dtrace_attribute_t dtp_arga;            /* arg attributes */
524         const dtrace_typeinfo_t *dtp_argv;      /* arg types */
525         int dtp_argc;                           /* arg count */
526 } dtrace_probeinfo_t;

528 typedef int dtrace_probe_f(dtrace_hdl_t *, const dtrace_probedesc_t *, void *);

530 extern int dtrace_probe_iter(dtrace_hdl_t *,
531     const dtrace_probedesc_t *pdp, dtrace_probe_f *, void *);

533 extern int dtrace_probe_info(dtrace_hdl_t *,
534     const dtrace_probedesc_t *, dtrace_probeinfo_t *);

536 /*
537  * DTrace Vector Interface
538  *
539  * The DTrace library normally speaks directly to dtrace(7D).  However,
540  * this communication may be vectored elsewhere.  Consumers who wish to
541  * perform a vectored open must fill in the vector, and use the dtrace_vopen()
542  * entry point to obtain a library handle.
543  */
544 struct dtrace_vector {
545         int (*dtv_ioctl)(void *, int, void *);
546         int (*dtv_lookup_by_addr)(void *, GElf_Addr, GElf_Sym *,
547             dtrace_syminfo_t *);
548         int (*dtv_status)(void *, processorid_t);
549         long (*dtv_sysconf)(void *, int);
550 };

552 /*
553  * DTrace Utility Functions
554  *
555  * Library clients can use these functions to convert addresses strings, to
556  * convert between string and integer probe descriptions and the
557  * dtrace_probedesc_t representation, and to perform similar conversions on
558  * stability attributes.
559  */
560 extern int dtrace_addr2str(dtrace_hdl_t *, uint64_t, char *, int);
561 extern int dtrace_uaddr2str(dtrace_hdl_t *, pid_t, uint64_t, char *, int);

563 extern int dtrace_xstr2desc(dtrace_hdl_t *, dtrace_probespec_t,
564     const char *, int, char *const [], dtrace_probedesc_t *);
```



```
566 extern int dtrace_str2desc(dtrace_hdl_t *, dtrace_probespec_t,
567     const char *, dtrace_probedesc_t *);

569 extern int dtrace_id2desc(dtrace_hdl_t *, dtrace_id_t, dtrace_probedesc_t *);

571 #define DTRACE_DESC2STR_MAX     1024    /* min buf size for dtrace_desc2str() */

573 extern char *dtrace_desc2str(const dtrace_probedesc_t *, char *, size_t);

575 #define DTRACE_ATTR2STR_MAX     64      /* min buf size for dtrace_attr2str() */

577 extern char *dtrace_attr2str(dtrace_attribute_t, char *, size_t);
578 extern int dtrace_str2attr(const char *, dtrace_attribute_t *);

580 extern const char *dtrace_stability_name(dtrace_stability_t);
581 extern const char *dtrace_class_name(dtrace_class_t);

583 extern int dtrace_provider_modules(dtrace_hdl_t *, const char **, int);

585 extern const char *const _dtrace_version;
586 extern int _dtrace_debug;

588 #ifdef  __cplusplus
589 }
590 #endif

592 #endif  /* _DTRACE_H */
```

```
**********************************************************
  122351 Tue Jan 14 16:49:01 2014
new/usr/src/pkg/manifests/system-dtrace-tests.mf
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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 (c) 2010, Oracle and/or its affiliates. All rights reserved.
  24 # Copyright (c) 2012 by Delphix. All rights reserved.
  25 #

  27 set name=pkg.fmri value=pkg:/system/dtrace/tests@$(PKGVERS)
  28 set name=pkg.description value="DTrace Test Suite Internal Distribution"
  29 set name=pkg.summary value="DTrace Test Suite"
  30 set name=info.classification \
  31     value=org.opensolaris.category.2008:Development/System
  32 set name=variant.arch value=$(ARCH)
  33 dir path=opt/SUNWdtrt group=sys
  34 dir path=opt/SUNWdtrt/bin
  35 dir path=opt/SUNWdtrt/bin/$(ARCH32)
  36 dir path=opt/SUNWdtrt/bin/$(ARCH64)
  37 dir path=opt/SUNWdtrt/lib
  38 dir path=opt/SUNWdtrt/lib/java
  39 dir path=opt/SUNWdtrt/tst
  40 dir path=opt/SUNWdtrt/tst/$(ARCH)
  41 dir path=opt/SUNWdtrt/tst/$(ARCH)/arrays
  42 $(i386_ONLY)dir path=opt/SUNWdtrt/tst/$(ARCH)/funcs
  43 dir path=opt/SUNWdtrt/tst/$(ARCH)/pid
  44 $(sparc_ONLY)dir path=opt/SUNWdtrt/tst/$(ARCH)/usdt
  45 dir path=opt/SUNWdtrt/tst/$(ARCH)/ustack
  46 dir path=opt/SUNWdtrt/tst/common
  47 dir path=opt/SUNWdtrt/tst/common/aggs
  48 dir path=opt/SUNWdtrt/tst/common/arithmetic
  49 dir path=opt/SUNWdtrt/tst/common/arrays
  50 dir path=opt/SUNWdtrt/tst/common/assocs
  51 dir path=opt/SUNWdtrt/tst/common/begin
  52 dir path=opt/SUNWdtrt/tst/common/bitfields
  53 dir path=opt/SUNWdtrt/tst/common/buffering
  54 dir path=opt/SUNWdtrt/tst/common/builtinvar
  55 dir path=opt/SUNWdtrt/tst/common/cg
  56 dir path=opt/SUNWdtrt/tst/common/clauses
```



```
  57 dir path=opt/SUNWdtrt/tst/common/cpc
  58 dir path=opt/SUNWdtrt/tst/common/decls
  59 dir path=opt/SUNWdtrt/tst/common/drops
  60 dir path=opt/SUNWdtrt/tst/common/dtraceUtil
  61 dir path=opt/SUNWdtrt/tst/common/end
  62 dir path=opt/SUNWdtrt/tst/common/enum
  63 dir path=opt/SUNWdtrt/tst/common/env
  64 dir path=opt/SUNWdtrt/tst/common/error
  65 dir path=opt/SUNWdtrt/tst/common/exit
  66 dir path=opt/SUNWdtrt/tst/common/fbtprovider
  67 dir path=opt/SUNWdtrt/tst/common/funcs
  68 dir path=opt/SUNWdtrt/tst/common/grammar
  69 dir path=opt/SUNWdtrt/tst/common/include
  70 dir path=opt/SUNWdtrt/tst/common/inline
  71 dir path=opt/SUNWdtrt/tst/common/io
  72 dir path=opt/SUNWdtrt/tst/common/ip
  73 dir path=opt/SUNWdtrt/tst/common/java_api
  74 dir path=opt/SUNWdtrt/tst/common/json
  75 dir path=opt/SUNWdtrt/tst/common/lexer
  76 dir path=opt/SUNWdtrt/tst/common/llquantize
  77 dir path=opt/SUNWdtrt/tst/common/mdb
  78 dir path=opt/SUNWdtrt/tst/common/mib
  79 dir path=opt/SUNWdtrt/tst/common/misc
  80 dir path=opt/SUNWdtrt/tst/common/multiaggs
  81 dir path=opt/SUNWdtrt/tst/common/nfs
  82 dir path=opt/SUNWdtrt/tst/common/offsetof
  83 dir path=opt/SUNWdtrt/tst/common/operators
  84 dir path=opt/SUNWdtrt/tst/common/pid
  85 dir path=opt/SUNWdtrt/tst/common/plockstat
  86 dir path=opt/SUNWdtrt/tst/common/pointers
  87 dir path=opt/SUNWdtrt/tst/common/pragma
  88 dir path=opt/SUNWdtrt/tst/common/predicates
  89 dir path=opt/SUNWdtrt/tst/common/preprocessor
  90 dir path=opt/SUNWdtrt/tst/common/print
  91 dir path=opt/SUNWdtrt/tst/common/printa
  92 dir path=opt/SUNWdtrt/tst/common/printf
  93 dir path=opt/SUNWdtrt/tst/common/privs
  94 dir path=opt/SUNWdtrt/tst/common/probes
  95 dir path=opt/SUNWdtrt/tst/common/proc
  96 dir path=opt/SUNWdtrt/tst/common/profile-n
  97 dir path=opt/SUNWdtrt/tst/common/providers
  98 dir path=opt/SUNWdtrt/tst/common/raise
  99 dir path=opt/SUNWdtrt/tst/common/rates
 100 dir path=opt/SUNWdtrt/tst/common/safety
 101 dir path=opt/SUNWdtrt/tst/common/scalars
 102 dir path=opt/SUNWdtrt/tst/common/sched
 103 dir path=opt/SUNWdtrt/tst/common/scripting
 104 dir path=opt/SUNWdtrt/tst/common/sdt
 105 dir path=opt/SUNWdtrt/tst/common/sizeof
 106 dir path=opt/SUNWdtrt/tst/common/speculation
 107 dir path=opt/SUNWdtrt/tst/common/stability
 108 dir path=opt/SUNWdtrt/tst/common/stack
 109 dir path=opt/SUNWdtrt/tst/common/stackdepth
 110 dir path=opt/SUNWdtrt/tst/common/stop
 111 dir path=opt/SUNWdtrt/tst/common/strlen
 112 dir path=opt/SUNWdtrt/tst/common/strtoll
 113 dir path=opt/SUNWdtrt/tst/common/struct
 114 dir path=opt/SUNWdtrt/tst/common/syscall
 115 dir path=opt/SUNWdtrt/tst/common/sysevent
 116 dir path=opt/SUNWdtrt/tst/common/tick-n
 117 dir path=opt/SUNWdtrt/tst/common/trace
 118 dir path=opt/SUNWdtrt/tst/common/tracemem
 119 dir path=opt/SUNWdtrt/tst/common/translators
 120 dir path=opt/SUNWdtrt/tst/common/typedef
 121 dir path=opt/SUNWdtrt/tst/common/types
 122 dir path=opt/SUNWdtrt/tst/common/uctf
```

```
#endif /* ! codereview */
dir path=opt/SUNWdtrt/tst/common/union
dir path=opt/SUNWdtrt/tst/common/usdt
dir path=opt/SUNWdtrt/tst/common/ustack
dir path=opt/SUNWdtrt/tst/common/vars
dir path=opt/SUNWdtrt/tst/common/version
$(i386_ONLY)dir path=opt/SUNWdtrt/tst/i86xpv
$(i386_ONLY)dir path=opt/SUNWdtrt/tst/i86xpv/xdt
file path=opt/SUNWdtrt/README mode=0444
file path=opt/SUNWdtrt/bin/$(ARCH32)/chkargs mode=0555
file path=opt/SUNWdtrt/bin/$(ARCH64)/chkargs mode=0555
file path=opt/SUNWdtrt/bin/baddof mode=0555
file path=opt/SUNWdtrt/bin/badioctl mode=0555
file path=opt/SUNWdtrt/bin/chkargs mode=0555
file path=opt/SUNWdtrt/bin/dstyle mode=0555
file path=opt/SUNWdtrt/bin/dtest mode=0555
file path=opt/SUNWdtrt/bin/dtfailures mode=0555
file path=opt/SUNWdtrt/bin/exception.lst mode=0444
file path=opt/SUNWdtrt/bin/jdtrace mode=0555
file path=opt/SUNWdtrt/lib/java/jdtrace.jar
file path=opt/SUNWdtrt/tst/$(ARCH)/arrays/tst.uregsarray.d mode=0444
$(i386_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/funcs/tst.badcopyin.d mode=0444
$(i386_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/funcs/tst.badcopyinstr.d \
    mode=0444
$(i386_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/funcs/tst.badcopyout.d \
    mode=0444
$(i386_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/funcs/tst.badcopyoutstr.d \
    mode=0444
$(sparc_ONLY)file \
    path=opt/SUNWdtrt/tst/$(ARCH)/pid/err.D_PROC_ALIGN.misaligned.d mode=0444
$(sparc_ONLY)file \
    path=opt/SUNWdtrt/tst/$(ARCH)/pid/err.D_PROC_ALIGN.misaligned.exe \
    mode=0555
$(i386_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.badinstr.d mode=0444
$(i386_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.badinstr.exe mode=0555
$(sparc_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.br.d mode=0444
$(sparc_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.br.d.out mode=0444
$(sparc_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.br.exe mode=0555
file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.branch.d mode=0444
file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.branch.exe mode=0555
file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.embedded.d mode=0444
file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.embedded.exe mode=0555
$(i386_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.ret.d mode=0444
$(i386_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.ret.exe mode=0555
$(i386_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.retlist.exe mode=0555
$(i386_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/pid/tst.retlist.ksh mode=0444
$(sparc_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/usdt/tst.tailcall.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/$(ARCH)/ustack/tst.annotated.d mode=0444
file path=opt/SUNWdtrt/tst/$(ARCH)/ustack/tst.annotated.d.out mode=0444
file path=opt/SUNWdtrt/tst/$(ARCH)/ustack/tst.annotated.exe mode=0555
file path=opt/SUNWdtrt/tst/$(ARCH)/ustack/tst.circstack.d mode=0444
file path=opt/SUNWdtrt/tst/$(ARCH)/ustack/tst.circstack.exe mode=0555
file path=opt/SUNWdtrt/tst/$(ARCH)/ustack/tst.helper.d mode=0444
file path=opt/SUNWdtrt/tst/$(ARCH)/ustack/tst.helper.d.out mode=0444
file path=opt/SUNWdtrt/tst/$(ARCH)/ustack/tst.helper.exe mode=0555
$(sparc_ONLY)file path=opt/SUNWdtrt/tst/$(ARCH)/ustack/tst.trapstat.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_AGG_FUNC.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_AGG_MDIM.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_AGG_NULL.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_AGG_REDEF.redef.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_AGG_SCALAR.avgtoofew.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_AGG_SCALAR.maxnoarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_AGG_SCALAR.mintoofew.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_AGG_SCALAR.quantizetoofew.d \
```



```
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_AGG_SCALAR.stddevtoofew.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_AGG_SCALAR.sumtoofew.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_CLEAR_AGGARG.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_CLEAR_PROTO.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_FUNC_IDENT.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_FUNC_UNDEF.badaggfunc.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_IDENT_UNDEF.badexpr.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_IDENT_UNDEF.badkey3.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_IDENT_UNDEF.noeffect.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_KEY_TYPE.badkey1.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_KEY_TYPE.badkey2.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_KEY_TYPE.badkey4.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_BASETYPE.lqbad1.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_BASETYPE.lqshort.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_BASEVAL.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_LIMTYPE.lqbad1.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_LIMVAL.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_MATCHBASE.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_MATCHBASE.order.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_MATCHLIM.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_MATCHLIM.order.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_MATCHSTEP.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_MISMATCH.lqbadarg.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_STEPLARGE.lqtoofew.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_STEPSMALL.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_STEPTYPE.lqbadinc.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_LQUANT_STEPVAL.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_NORMALIZE_AGGARG.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_NORMALIZE_PROTO.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_NORMALIZE_SCALAR.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_ARG.lquantizetoofew.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.avgnoarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.avgtoomany.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.counttoomany.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.lquantizenoarg.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.lquantizetoomany.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.maxnoarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.maxtoomany.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.minnoarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.mintoomany.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.quantizenoarg.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.quantizetoomany.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.stddevnoarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.stddevtoomany.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.sumnoarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_PROTO_LEN.sumtoomany.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_TRUNC_AGGARG.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_TRUNC_PROTO.badmany.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_TRUNC_PROTO.badnone.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/err.D_TRUNC_SCALAR.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/aggs/tst.allquant.d mode=0444
```

```
255 file path=opt/SUNWdtrt/tst/common/aggs/tst.allquant.d.out mode=0444
256 file path=opt/SUNWdtrt/tst/common/aggs/tst.avg.d mode=0444
257 file path=opt/SUNWdtrt/tst/common/aggs/tst.avg.d.out mode=0444
258 file path=opt/SUNWdtrt/tst/common/aggs/tst.avg_neg.d mode=0444
259 file path=opt/SUNWdtrt/tst/common/aggs/tst.avg_neg.d.out mode=0444
260 file path=opt/SUNWdtrt/tst/common/aggs/tst.clear.d mode=0444
261 file path=opt/SUNWdtrt/tst/common/aggs/tst.clear.d.out mode=0444
262 file path=opt/SUNWdtrt/tst/common/aggs/tst.clearavg.d mode=0444
263 file path=opt/SUNWdtrt/tst/common/aggs/tst.clearavg.d.out mode=0444
264 file path=opt/SUNWdtrt/tst/common/aggs/tst.clearavg2.d mode=0444
265 file path=opt/SUNWdtrt/tst/common/aggs/tst.clearavg2.d.out mode=0444
266 file path=opt/SUNWdtrt/tst/common/aggs/tst.cleardenormalize.d mode=0444
267 file path=opt/SUNWdtrt/tst/common/aggs/tst.cleardenormalize.d.out mode=0444
268 file path=opt/SUNWdtrt/tst/common/aggs/tst.clearlquantize.d mode=0444
269 file path=opt/SUNWdtrt/tst/common/aggs/tst.clearlquantize.d.out mode=0444
270 file path=opt/SUNWdtrt/tst/common/aggs/tst.clearnormalize.d mode=0444
271 file path=opt/SUNWdtrt/tst/common/aggs/tst.clearnormalize.d.out mode=0444
272 file path=opt/SUNWdtrt/tst/common/aggs/tst.clearstddev.d mode=0444
273 file path=opt/SUNWdtrt/tst/common/aggs/tst.clearstddev.d.out mode=0444
274 file path=opt/SUNWdtrt/tst/common/aggs/tst.count.d mode=0444
275 file path=opt/SUNWdtrt/tst/common/aggs/tst.count.d.out mode=0444
276 file path=opt/SUNWdtrt/tst/common/aggs/tst.count2.d mode=0444
277 file path=opt/SUNWdtrt/tst/common/aggs/tst.count2.d.out mode=0444
278 file path=opt/SUNWdtrt/tst/common/aggs/tst.count3.d mode=0444
279 file path=opt/SUNWdtrt/tst/common/aggs/tst.denormalize.d mode=0444
280 file path=opt/SUNWdtrt/tst/common/aggs/tst.denormalize.d.out mode=0444
281 file path=opt/SUNWdtrt/tst/common/aggs/tst.denormalizeonly.d mode=0444
282 file path=opt/SUNWdtrt/tst/common/aggs/tst.denormalizeonly.d.out mode=0444
283 file path=opt/SUNWdtrt/tst/common/aggs/tst.fmtnormalize.d mode=0444
284 file path=opt/SUNWdtrt/tst/common/aggs/tst.fmtnormalize.d.out mode=0444
285 file path=opt/SUNWdtrt/tst/common/aggs/tst.forms.d mode=0444
286 file path=opt/SUNWdtrt/tst/common/aggs/tst.forms.d.out mode=0444
287 file path=opt/SUNWdtrt/tst/common/aggs/tst.goodkey.d mode=0444
288 file path=opt/SUNWdtrt/tst/common/aggs/tst.keysort.d mode=0444
289 file path=opt/SUNWdtrt/tst/common/aggs/tst.keysort.d.out mode=0444
290 file path=opt/SUNWdtrt/tst/common/aggs/tst.lquantize.d mode=0444
291 file path=opt/SUNWdtrt/tst/common/aggs/tst.lquantize.d.out mode=0444
292 file path=opt/SUNWdtrt/tst/common/aggs/tst.lquantnormal.d mode=0444
293 file path=opt/SUNWdtrt/tst/common/aggs/tst.lquantnormal.d.out mode=0444
294 file path=opt/SUNWdtrt/tst/common/aggs/tst.lquantrange.d mode=0444
295 file path=opt/SUNWdtrt/tst/common/aggs/tst.lquantrange.d.out mode=0444
296 file path=opt/SUNWdtrt/tst/common/aggs/tst.lquantround.d mode=0444
297 file path=opt/SUNWdtrt/tst/common/aggs/tst.lquantround.d.out mode=0444
298 file path=opt/SUNWdtrt/tst/common/aggs/tst.lquantzero.d mode=0444
299 file path=opt/SUNWdtrt/tst/common/aggs/tst.lquantzero.d.out mode=0444
300 file path=opt/SUNWdtrt/tst/common/aggs/tst.max.d mode=0444
301 file path=opt/SUNWdtrt/tst/common/aggs/tst.max.d.out mode=0444
302 file path=opt/SUNWdtrt/tst/common/aggs/tst.max_neg.d mode=0444
303 file path=opt/SUNWdtrt/tst/common/aggs/tst.max_neg.d.out mode=0444
304 file path=opt/SUNWdtrt/tst/common/aggs/tst.min.d mode=0444
305 file path=opt/SUNWdtrt/tst/common/aggs/tst.min.d.out mode=0444
306 file path=opt/SUNWdtrt/tst/common/aggs/tst.min_neg.d mode=0444
307 file path=opt/SUNWdtrt/tst/common/aggs/tst.min_neg.d.out mode=0444
308 file path=opt/SUNWdtrt/tst/common/aggs/tst.multiaggs1.d mode=0444
309 file path=opt/SUNWdtrt/tst/common/aggs/tst.multiaggs2.d mode=0444
310 file path=opt/SUNWdtrt/tst/common/aggs/tst.multiaggs2.d.out mode=0444
311 file path=opt/SUNWdtrt/tst/common/aggs/tst.multiaggs3.d mode=0444
312 file path=opt/SUNWdtrt/tst/common/aggs/tst.multiaggs3.d.out mode=0444
313 file path=opt/SUNWdtrt/tst/common/aggs/tst.multinormalize.d mode=0444
314 file path=opt/SUNWdtrt/tst/common/aggs/tst.multinormalize.d.out mode=0444
315 file path=opt/SUNWdtrt/tst/common/aggs/tst.neglquant.d mode=0444
316 file path=opt/SUNWdtrt/tst/common/aggs/tst.neglquant.d.out mode=0444
317 file path=opt/SUNWdtrt/tst/common/aggs/tst.negorder.d mode=0444
318 file path=opt/SUNWdtrt/tst/common/aggs/tst.negorder.d.out mode=0444
319 file path=opt/SUNWdtrt/tst/common/aggs/tst.negquant.d mode=0444
320 file path=opt/SUNWdtrt/tst/common/aggs/tst.negquant.d.out mode=0444
```



```
321 file path=opt/SUNWdtrt/tst/common/aggs/tst.negtrunc.d mode=0444
322 file path=opt/SUNWdtrt/tst/common/aggs/tst.negtrunc.d.out mode=0444
323 file path=opt/SUNWdtrt/tst/common/aggs/tst.negtruncquant.d mode=0444
324 file path=opt/SUNWdtrt/tst/common/aggs/tst.negtruncquant.d.out mode=0444
325 file path=opt/SUNWdtrt/tst/common/aggs/tst.normalize.d mode=0444
326 file path=opt/SUNWdtrt/tst/common/aggs/tst.normalize.d.out mode=0444
327 file path=opt/SUNWdtrt/tst/common/aggs/tst.order.d mode=0444
328 file path=opt/SUNWdtrt/tst/common/aggs/tst.order.d.out mode=0444
329 file path=opt/SUNWdtrt/tst/common/aggs/tst.quantize.d mode=0444
330 file path=opt/SUNWdtrt/tst/common/aggs/tst.quantize.d.out mode=0444
331 file path=opt/SUNWdtrt/tst/common/aggs/tst.quantmany.d mode=0444
332 file path=opt/SUNWdtrt/tst/common/aggs/tst.quantmany.d.out mode=0444
333 file path=opt/SUNWdtrt/tst/common/aggs/tst.quantround.d mode=0444
334 file path=opt/SUNWdtrt/tst/common/aggs/tst.quantround.d.out mode=0444
335 file path=opt/SUNWdtrt/tst/common/aggs/tst.quantzero.d mode=0444
336 file path=opt/SUNWdtrt/tst/common/aggs/tst.quantzero.d.out mode=0444
337 file path=opt/SUNWdtrt/tst/common/aggs/tst.signature.d mode=0444
338 file path=opt/SUNWdtrt/tst/common/aggs/tst.signedkeys.d mode=0444
339 file path=opt/SUNWdtrt/tst/common/aggs/tst.signedkeys.d.out mode=0444
340 file path=opt/SUNWdtrt/tst/common/aggs/tst.signedkeyspos.d mode=0444
341 file path=opt/SUNWdtrt/tst/common/aggs/tst.signedkeyspos.d.out mode=0444
342 file path=opt/SUNWdtrt/tst/common/aggs/tst.sizedkeys.d mode=0444
343 file path=opt/SUNWdtrt/tst/common/aggs/tst.sizedkeys.d.out mode=0444
344 file path=opt/SUNWdtrt/tst/common/aggs/tst.stddev.d mode=0444
345 file path=opt/SUNWdtrt/tst/common/aggs/tst.stddev.d.out mode=0444
346 file path=opt/SUNWdtrt/tst/common/aggs/tst.subr.d mode=0444
347 file path=opt/SUNWdtrt/tst/common/aggs/tst.sum.d mode=0444
348 file path=opt/SUNWdtrt/tst/common/aggs/tst.sum.d.out mode=0444
349 file path=opt/SUNWdtrt/tst/common/aggs/tst.trunc.d mode=0444
350 file path=opt/SUNWdtrt/tst/common/aggs/tst.trunc.d.out mode=0444
351 file path=opt/SUNWdtrt/tst/common/aggs/tst.trunc0.d mode=0444
352 file path=opt/SUNWdtrt/tst/common/aggs/tst.trunc0.d.out mode=0444
353 file path=opt/SUNWdtrt/tst/common/aggs/tst.truncquant.d mode=0444
354 file path=opt/SUNWdtrt/tst/common/aggs/tst.truncquant.d.out mode=0444
355 file path=opt/SUNWdtrt/tst/common/aggs/tst.valsortkeypos.d mode=0444
356 file path=opt/SUNWdtrt/tst/common/aggs/tst.valsortkeypos.d.out mode=0444
357 file path=opt/SUNWdtrt/tst/common/arithmetic/err.D_DIV_ZERO.divby0.d mode=0444
358 file path=opt/SUNWdtrt/tst/common/arithmetic/err.D_DIV_ZERO.divby0_1.d \
359     mode=0444
360 file path=opt/SUNWdtrt/tst/common/arithmetic/err.D_DIV_ZERO.divby0_2.d \
361     mode=0444
362 file path=opt/SUNWdtrt/tst/common/arithmetic/err.D_DIV_ZERO.modby0.d mode=0444
363 file path=opt/SUNWdtrt/tst/common/arithmetic/err.D_SYNTAX.addmin.d mode=0444
364 file path=opt/SUNWdtrt/tst/common/arithmetic/err.D_SYNTAX.divmin.d mode=0444
365 file path=opt/SUNWdtrt/tst/common/arithmetic/err.D_SYNTAX.muladd.d mode=0444
366 file path=opt/SUNWdtrt/tst/common/arithmetic/err.D_SYNTAX.muldiv.d mode=0444
367 file path=opt/SUNWdtrt/tst/common/arithmetic/tst.basics.d mode=0444
368 file path=opt/SUNWdtrt/tst/common/arithmetic/tst.basics.d.out mode=0444
369 file path=opt/SUNWdtrt/tst/common/arithmetic/tst.compcast.d mode=0444
370 file path=opt/SUNWdtrt/tst/common/arithmetic/tst.compcast.d.out mode=0444
371 file path=opt/SUNWdtrt/tst/common/arithmetic/tst.compnarrowassign.d mode=0444
372 file path=opt/SUNWdtrt/tst/common/arithmetic/tst.compnarrowassign.d.out \
373     mode=0444
374 file path=opt/SUNWdtrt/tst/common/arithmetic/tst.execcast.d mode=0444
375 file path=opt/SUNWdtrt/tst/common/arithmetic/tst.execcast.d.out mode=0444
376 file path=opt/SUNWdtrt/tst/common/arrays/err.D_ARR_BADREF.bad.d mode=0444
377 file path=opt/SUNWdtrt/tst/common/arrays/err.D_DECL_ARRBIG.toobig.d mode=0444
378 file path=opt/SUNWdtrt/tst/common/arrays/err.D_DECL_ARRNULL.bad.d mode=0444
379 file path=opt/SUNWdtrt/tst/common/arrays/err.D_DECL_ARRSUB.bad.d mode=0444
380 file path=opt/SUNWdtrt/tst/common/arrays/err.D_DECL_PROTO_TYPE.badtuple.d \
381     mode=0444
382 file path=opt/SUNWdtrt/tst/common/arrays/err.D_IDENT_UNDEF.badureg.d mode=0444
383 file path=opt/SUNWdtrt/tst/common/arrays/tst.basic1.d mode=0444
384 file path=opt/SUNWdtrt/tst/common/arrays/tst.basic2.d mode=0444
385 file path=opt/SUNWdtrt/tst/common/arrays/tst.basic3.d mode=0444
386 file path=opt/SUNWdtrt/tst/common/arrays/tst.basic4.d mode=0444
```

```
file path=opt/SUNWdtrt/tst/common/arrays/tst.basic5.d mode=0444
file path=opt/SUNWdtrt/tst/common/arrays/tst.basic6.d mode=0444
file path=opt/SUNWdtrt/tst/common/arrays/tst.uregsarray.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/err.D_OP_INCOMPAT.dupgtype.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/err.D_OP_INCOMPAT.dupttype.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/err.D_OP_INCOMPAT.this.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/err.D_PROTO_ARG.badsig.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/err.D_PROTO_LEN.toofew.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/err.D_PROTO_LEN.toomany.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/err.D_SYNTAX.errassign.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/err.tupoflow.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/tst.cpyarray.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/tst.diffprofile.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/tst.initialize.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/tst.invalidref.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/tst.misc.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/tst.orthogonality.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/tst.this.d mode=0444
file path=opt/SUNWdtrt/tst/common/assocs/tst.valassign.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/begin/err.D_PDESC_ZERO.begin.d mode=0444
file path=opt/SUNWdtrt/tst/common/begin/err.D_PDESC_ZERO.tick.d mode=0444
file path=opt/SUNWdtrt/tst/common/begin/tst.begin.d mode=0444
file path=opt/SUNWdtrt/tst/common/begin/tst.begin.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/begin/tst.multibegin.d mode=0444
file path=opt/SUNWdtrt/tst/common/begin/tst.multibegin.d.out mode=0444
file \
    path=opt/SUNWdtrt/tst/common/bitfields/err.D_ADDROF_BITFIELD.BitfieldAddress
    mode=0444
file path=opt/SUNWdtrt/tst/common/bitfields/err.D_DECL_BFCONST.NegBitField.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/bitfields/err.D_DECL_BFCONST.ZeroBitField.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/bitfields/err.D_DECL_BFSIZE.ExceedBaseType.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/bitfields/err.D_DECL_BFSIZE.GreaterThan64.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/bitfields/err.D_DECL_BFTYPE.badtype.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/bitfields/err.D_OFFSETOF_BITFIELD.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/bitfields/err.D_SIZEOF_BITFIELD.SizeofBitfield.
    mode=0444
file path=opt/SUNWdtrt/tst/common/bitfields/tst.BitFieldPromotion.d mode=0444
file path=opt/SUNWdtrt/tst/common/bitfields/tst.SizeofBitField.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/err.end.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/err.resize1.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/err.resize2.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/err.resize3.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/err.zerobuf.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.alignring.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.cputime.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.dynvarsize.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.fill1.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.fill1.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.resize1.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.resize2.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.resize3.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.ring1.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.ring2.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.ring2.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.ring3.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.ring3.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.smallring.d mode=0444
```



```
file path=opt/SUNWdtrt/tst/common/buffering/tst.switch1.d mode=0444
file path=opt/SUNWdtrt/tst/common/buffering/tst.switch1.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/err.D_XLATE_NOCONV.cpuusage.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/err.D_XLATE_NOCONV.nice.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/err.D_XLATE_NOCONV.priority.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/err.D_XLATE_NOCONV.prsize.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/err.D_XLATE_NOCONV.rssize.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.arg0.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.arg0clause.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.arg1.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.arg1to8.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.arg1to8clause.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.caller.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.caller1.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.epid.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.epid1.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.errno.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.errno1.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.execname.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.hpriority.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.id.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.id1.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.ipl.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.ipl1.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.lwpsinfo.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.lwpsinfo1.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.pid.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.pid1.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.psinfo.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.psinfo1.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.tid.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.tid1.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.timestamp.d mode=0444
file path=opt/SUNWdtrt/tst/common/builtinvar/tst.vtimestamp.d mode=0444
file path=opt/SUNWdtrt/tst/common/cg/err.D_NOREG.noreg.d mode=0444
file path=opt/SUNWdtrt/tst/common/cg/err.baddif.d mode=0444
file path=opt/SUNWdtrt/tst/common/clauses/err.D_IDENT_UNDEF.aggfun.d mode=0444
file path=opt/SUNWdtrt/tst/common/clauses/err.D_IDENT_UNDEF.aggtup.d mode=0444
file path=opt/SUNWdtrt/tst/common/clauses/err.D_IDENT_UNDEF.arrtup.d mode=0444
file path=opt/SUNWdtrt/tst/common/clauses/err.D_IDENT_UNDEF.body.d mode=0444
file path=opt/SUNWdtrt/tst/common/clauses/err.D_IDENT_UNDEF.both.d mode=0444
file path=opt/SUNWdtrt/tst/common/clauses/err.D_IDENT_UNDEF.pred.d mode=0444
file path=opt/SUNWdtrt/tst/common/clauses/tst.nopred.d mode=0444
file path=opt/SUNWdtrt/tst/common/clauses/tst.pred.d mode=0444
file path=opt/SUNWdtrt/tst/common/clauses/tst.predfirst.d mode=0444
file path=opt/SUNWdtrt/tst/common/clauses/tst.predlast.d mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/err.D_PDESC_ZERO.lowfrequency.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/err.D_PDESC_ZERO.malformedoverflow.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/err.D_PDESC_ZERO.nonexistentevent.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/err.cpcvscpustatpart1.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/err.cpcvscpustatpart2.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/err.cputrackfailtostart.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/err.cputrackterminates.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/err.toomanyenablings.d mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/tst.allcpus.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/tst.genericevent.d mode=0444
file path=opt/SUNWdtrt/tst/common/cpc/tst.platformevent.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/decls/err.D_DECL_LOCASSC.NonLocalAssoc.d \
```

```
519     mode=0444
520 file path=opt/SUNWdtrt/tst/common/decls/err.D_DECL_LONGINT.LongStruct.d \
521     mode=0444
522 file path=opt/SUNWdtrt/tst/common/decls/err.D_DECL_PARMCLASS.BadStorageClass.d \
523     mode=0444
524 file path=opt/SUNWdtrt/tst/common/decls/err.D_DECL_PROTO_NAME.VoidName.d \
525     mode=0444
526 file path=opt/SUNWdtrt/tst/common/decls/err.D_DECL_PROTO_TYPE.Dyn.d mode=0444
527 file path=opt/SUNWdtrt/tst/common/decls/err.D_DECL_PROTO_VARARGS.VarLenArgs.d \
528     mode=0444
529 file path=opt/SUNWdtrt/tst/common/decls/err.D_DECL_PROTO_VOID.NonSoleVoid.d \
530     mode=0444
531 file path=opt/SUNWdtrt/tst/common/decls/err.D_DECL_SIGNINT.UnsignedStruct.d \
532     mode=0444
533 file path=opt/SUNWdtrt/tst/common/decls/err.D_DECL_VOIDATTR.ShortVoidDecl.d \
534     mode=0444
535 file path=opt/SUNWdtrt/tst/common/decls/tst.arrays.d mode=0444
536 file path=opt/SUNWdtrt/tst/common/decls/tst.basics.d mode=0444
537 file path=opt/SUNWdtrt/tst/common/decls/tst.funcs.d mode=0444
538 file path=opt/SUNWdtrt/tst/common/decls/tst.pointers.d mode=0444
539 file path=opt/SUNWdtrt/tst/common/decls/tst.varargsfuncs.d mode=0444
540 file path=opt/SUNWdtrt/tst/common/drops/drp.DTRACEDROP_AGGREGATION.d mode=0444
541 file path=opt/SUNWdtrt/tst/common/drops/drp.DTRACEDROP_DBLERROR.d mode=0444
542 file path=opt/SUNWdtrt/tst/common/drops/drp.DTRACEDROP_DYNAMIC.d mode=0444
543 file path=opt/SUNWdtrt/tst/common/drops/drp.DTRACEDROP_PRINCIPAL.d mode=0444
544 file path=opt/SUNWdtrt/tst/common/drops/drp.DTRACEDROP_PRINCIPAL.end.d \
545     mode=0444
546 file path=opt/SUNWdtrt/tst/common/drops/drp.DTRACEDROP_SPEC.d mode=0444
547 file path=opt/SUNWdtrt/tst/common/drops/drp.DTRACEDROP_SPECUNAVAIL.d mode=0444
548 file path=opt/SUNWdtrt/tst/common/drops/drp.DTRACEDROP_STKSTROVERFLOW.d \
549     mode=0444
550 file \
551     path=opt/SUNWdtrt/tst/common/dtraceUtil/err.D_PDESC_ZERO.InvalidDescription1
552     mode=0444
553 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.AddSearchPath.d.ksh mode=0444
554 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.BufsizeGiga.d.ksh mode=0444
555 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.BufsizeKilo.d.ksh mode=0444
556 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.BufsizeMega.d.ksh mode=0444
557 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.BufsizeTera.d.ksh mode=0444
558 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DataModel32.d.ksh mode=0444
559 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DataModel64.d.ksh mode=0444
560 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DefineNameWithCPP.d.ksh \
561     mode=0444
562 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DefineNameWithCPP.d.ksh.out \
563     mode=0444
564 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithFunction.d.ksh \
565     mode=0444
566 file \
567     path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithFunction.d.ksh.out \
568     mode=0444
569 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithID.d.ksh \
570     mode=0444
571 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithID.d.ksh.out \
572     mode=0444
573 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithModule.d.ksh \
574     mode=0444
575 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithModule.d.ksh.out \
576     mode=0444
577 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithName.d.ksh \
578     mode=0444
579 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithName.d.ksh.out \
580     mode=0444
581 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithProvider.d.ksh \
582     mode=0444
583 file \
584     path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithProvider.d.ksh.out \
```



```
585     mode=0444
586 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.DestructWithoutW.d.ksh \
587     mode=0444
588 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ELFGenerationOut.d.ksh \
589     mode=0444
590 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ELFGenerationWithO.d.ksh \
591     mode=0444
592 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ExitStatus1.d.ksh mode=0444
593 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ExitStatus2.d.ksh mode=0444
594 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ExtraneousProbeIds.d.ksh \
595     mode=0444
596 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidFuncName1.d.ksh \
597     mode=0444
598 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidFuncName2.d.ksh \
599     mode=0444
600 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidId1.d.ksh mode=0444
601 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidId2.d.ksh mode=0444
602 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidId3.d.ksh mode=0444
603 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidModule1.d.ksh \
604     mode=0444
605 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidModule2.d.ksh \
606     mode=0444
607 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidModule3.d.ksh \
608     mode=0444
609 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidModule4.d.ksh \
610     mode=0444
611 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidProbeIdentifier.d.ksh \
612     mode=0444
613 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidProvider1.d.ksh \
614     mode=0444
615 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidProvider2.d.ksh \
616     mode=0444
617 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidProvider3.d.ksh \
618     mode=0444
619 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidProvider4.d.ksh \
620     mode=0444
621 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceFunc1.d.ksh \
622     mode=0444
623 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceFunc2.d.ksh \
624     mode=0444
625 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceFunc3.d.ksh \
626     mode=0444
627 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceFunc4.d.ksh \
628     mode=0444
629 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceFunc5.d.ksh \
630     mode=0444
631 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceFunc6.d.ksh \
632     mode=0444
633 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceFunc7.d.ksh \
634     mode=0444
635 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceFunc8.d.ksh \
636     mode=0444
637 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceFunc9.d.ksh \
638     mode=0444
639 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceID1.d.ksh \
640     mode=0444
641 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceID2.d.ksh \
642     mode=0444
643 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceID3.d.ksh \
644     mode=0444
645 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceID4.d.ksh \
646     mode=0444
647 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceID5.d.ksh \
648     mode=0444
649 file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceID6.d.ksh \
650     mode=0444
```

```
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceID7.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceModule1.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceModule2.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceModule3.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceModule4.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceModule5.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceModule6.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceModule7.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceModule8.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceName1.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceName2.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceName3.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceName4.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceName5.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceName6.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceName7.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceName8.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceName9.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceProvider1.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceProvider2.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceProvider3.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceProvider4.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.InvalidTraceProvider5.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.MultipleInvalidProbeId.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.PreprocessorStatement.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.QuietMode.d.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.QuietMode.d.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.TestCompile.d.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.TestCompile.d.ksh.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.UnDefineNameWithCPP.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ZeroFunctionProbes.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ZeroFunctionProbes.d.ksh.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ZeroModuleProbes.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ZeroModuleProbes.d.ksh.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ZeroNameProbes.d.ksh \
```



```
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ZeroNameProbes.d.ksh.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ZeroProbeIdentfier.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ZeroProbesWithoutZ.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ZeroProviderProbes.d.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/dtraceUtil/tst.ZeroProviderProbes.d.ksh.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/end/err.D_IDENT_UNDEF.timespent.d mode=0444
file path=opt/SUNWdtrt/tst/common/end/tst.end.d mode=0444
file path=opt/SUNWdtrt/tst/common/end/tst.endwithoutbegin.d mode=0444
file path=opt/SUNWdtrt/tst/common/end/tst.multibeginend.d mode=0444
file path=opt/SUNWdtrt/tst/common/end/tst.multiend.d mode=0444
file path=opt/SUNWdtrt/tst/common/enum/err.D_DECL_IDRED.EnumSameName.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/enum/err.D_UNKNOWN.RepeatIdentifiers.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/enum/tst.EnumEquality.d mode=0444
file path=opt/SUNWdtrt/tst/common/enum/tst.EnumSameValue.d mode=0444
file path=opt/SUNWdtrt/tst/common/enum/tst.EnumValAssign.d mode=0444
file path=opt/SUNWdtrt/tst/common/env/err.D_PRAGMA_OPTSET.setfromscript.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/env/err.D_PRAGMA_OPTSET.unsetfromscript.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/env/tst.ld_nolazyload.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/env/tst.ld_nolazyload.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/env/tst.setenv1.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/env/tst.setenv1.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/env/tst.setenv2.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/env/tst.setenv2.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/env/tst.unsetenv1.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/env/tst.unsetenv1.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/env/tst.unsetenv2.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/env/tst.unsetenv2.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/error/tst.DTRACEFLT_BADADDR.d mode=0444
file path=opt/SUNWdtrt/tst/common/error/tst.DTRACEFLT_DIVZERO.d mode=0444
file path=opt/SUNWdtrt/tst/common/error/tst.DTRACEFLT_UNKNOWN.d mode=0444
file path=opt/SUNWdtrt/tst/common/error/tst.error.d mode=0444
file path=opt/SUNWdtrt/tst/common/error/tst.errorend.d mode=0444
file path=opt/SUNWdtrt/tst/common/exit/err.D_PROTO_LEN.noarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/exit/err.exitarg1.d mode=0444
file path=opt/SUNWdtrt/tst/common/exit/tst.basic1.d mode=0444
file path=opt/SUNWdtrt/tst/common/fbtprovider/err.D_PDESC_ZERO.notreturn.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/fbtprovider/tst.basic.d mode=0444
file path=opt/SUNWdtrt/tst/common/fbtprovider/tst.functionentry.d mode=0444
file path=opt/SUNWdtrt/tst/common/fbtprovider/tst.functionreturnvalue.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/fbtprovider/tst.ioctlargs.d mode=0444
file path=opt/SUNWdtrt/tst/common/fbtprovider/tst.offset.d mode=0444
file path=opt/SUNWdtrt/tst/common/fbtprovider/tst.offsetzero.d mode=0444
file path=opt/SUNWdtrt/tst/common/fbtprovider/tst.return.d mode=0444
file path=opt/SUNWdtrt/tst/common/fbtprovider/tst.return0.d mode=0444
file path=opt/SUNWdtrt/tst/common/fbtprovider/tst.tailcall.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_FUNC_UNDEF.progenyofbad1.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_OP_VFPTR.badop.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_ARG.chillbadarg.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_ARG.copyoutbadarg.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_ARG.mobadarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_ARG.raisebadarg.d \
```

```
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_ARG.tolower.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_ARG.toupper.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.allocanoarg.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.badbreakpoint.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.chilltoofew.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.chilltoomany.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.copyoutstrbadarg.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.copyoutstrtoofew.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.copyouttoofew.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.copyouttoomany.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.motoofew.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.motoomany.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.mtabadarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.mtatoofew.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.mtatoomany.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.panicbadarg.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.progenyofbad2.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.stopbadarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.tolower.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.tolowertoomany.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.toupper.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_PROTO_LEN.touppertoomany.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_STRINGOF_TYPE.badstringof.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.D_VAR_UNDEF.badvar.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.badalloca.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.badalloca2.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.badbcopy.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.badbcopy1.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.badbcopy2.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.badbcopy3.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.badbcopy4.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.badbcopy5.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.badbcopy6.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.badchill.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.chillbadarg.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.copyout.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.copyoutbadaddr.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.copyoutstrbadaddr.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.inet_ntoa6badaddr.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.inet_ntoabadaddr.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.inet_ntopbadaddr.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/err.inet_ntopbadarg.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.badfreopen.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.basename.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.basename.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.bcopy.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.chill.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.cleanpath.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.cleanpath.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.copyin.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.copyinto.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.ddi_pathname.d mode=0444
```



```
file path=opt/SUNWdtrt/tst/common/funcs/tst.default.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.freopen.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.ftruncate.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.ftruncate.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.hton.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.index.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.index.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.inet_ntoa.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.inet_ntoa.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.inet_ntoa6.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.inet_ntoa6.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.inet_ntop.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.inet_ntop.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.lltostr.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.lltostr.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.lltostrbase.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.lltostrbase.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.mutex_owned.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.mutex_owner.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.mutex_type_adaptive.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.progenyof.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.rand.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.strchr.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.strchr.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.strjoin.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.strjoin.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.strstr.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.strstr.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.strtok.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.strtok.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.strtok_null.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.substr.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.substr.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.substrminate.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.substrminate.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.system.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.system.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.tolower.d mode=0444
file path=opt/SUNWdtrt/tst/common/funcs/tst.toupper.d mode=0444
file path=opt/SUNWdtrt/tst/common/grammar/err.D_ADDROF_LVAL.d mode=0444
file path=opt/SUNWdtrt/tst/common/grammar/err.D_EMPTY.empty.d mode=0444
file path=opt/SUNWdtrt/tst/common/grammar/tst.clauses.d mode=0444
file path=opt/SUNWdtrt/tst/common/grammar/tst.stmts.d mode=0444
file path=opt/SUNWdtrt/tst/common/include/tst.includefirst.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/inline/err.D_DECL_IDRED.redef1.d mode=0444
file path=opt/SUNWdtrt/tst/common/inline/err.D_DECL_IDRED.redef2.d mode=0444
file path=opt/SUNWdtrt/tst/common/inline/err.D_IDENT_UNDEF.recur.d mode=0444
file path=opt/SUNWdtrt/tst/common/inline/err.D_OP_INCOMPAT.baddef1.d mode=0444
file path=opt/SUNWdtrt/tst/common/inline/err.D_OP_INCOMPAT.baddef2.d mode=0444
file path=opt/SUNWdtrt/tst/common/inline/err.D_OP_INCOMPAT.badxlate.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/inline/tst.InlineDataAssign.d mode=0444
file path=opt/SUNWdtrt/tst/common/inline/tst.InlineExpression.d mode=0444
file path=opt/SUNWdtrt/tst/common/inline/tst.InlineKinds.d mode=0444
file path=opt/SUNWdtrt/tst/common/inline/tst.InlineKinds.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/inline/tst.InlineTypedef.d mode=0444
file path=opt/SUNWdtrt/tst/common/inline/tst.InlineWritableAssign.d mode=0444
file path=opt/SUNWdtrt/tst/common/io/tst.fds.d mode=0444
file path=opt/SUNWdtrt/tst/common/io/tst.fds.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/io/tst.fds.exe mode=0555
file path=opt/SUNWdtrt/tst/common/ip/get.ipv4remote.pl mode=0555
file path=opt/SUNWdtrt/tst/common/ip/get.ipv6remote.pl mode=0555
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4localicmp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4localicmp.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4localtcp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4localtcp.ksh.out mode=0444
```

```
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4localudp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4localudp.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4remoteicmp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4remoteicmp.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4remotetcp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4remotetcp.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4remoteudp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv4remoteudp.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv6localicmp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv6localicmp.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv6remoteicmp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.ipv6remoteicmp.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.localtcpstate.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.localtcpstate.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.remotetcpstate.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/ip/tst.remotetcpstate.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/test.jar
file path=opt/SUNWdtrt/tst/common/java_api/tst.Abort.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.Abort.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.Bean.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.Bean.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.Close.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.Close.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.Drop.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.Drop.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.Enable.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.Enable.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.FunctionLookup.exe mode=0555
file path=opt/SUNWdtrt/tst/common/java_api/tst.FunctionLookup.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.FunctionLookup.ksh.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.GetAggregate.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.MaxConsumers.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.MaxConsumers.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.MultiAggPrinta.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.MultiAggPrinta.ksh.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.ProbeData.exe mode=0555
file path=opt/SUNWdtrt/tst/common/java_api/tst.ProbeData.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.ProbeData.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.ProbeDescription.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.ProbeDescription.ksh.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.StateMachine.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.StateMachine.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.StopLock.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.StopLock.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.printa.d mode=0444
file path=opt/SUNWdtrt/tst/common/java_api/tst.printa.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/json/tst.general.d mode=0444
file path=opt/SUNWdtrt/tst/common/json/tst.general.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/json/tst.strsize.d mode=0444
file path=opt/SUNWdtrt/tst/common/json/tst.strsize.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/json/tst.usdt.d mode=0444
file path=opt/SUNWdtrt/tst/common/json/tst.usdt.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/json/tst.usdt.exe mode=0555
file path=opt/SUNWdtrt/tst/common/lexer/err.D_CHR_NL.char.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_CHR_NULL.char.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_INT_DIGIT.InvalidDigit.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_INT_OFLOW.BigInt.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_STR_NL.string.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_SYNTAX.brace1.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_SYNTAX.brace2.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_SYNTAX.brack1.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_SYNTAX.brack2.d mode=0444
```



```
file path=opt/SUNWdtrt/tst/common/lexer/err.D_SYNTAX.brack3.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_SYNTAX.paren1.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_SYNTAX.paren2.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/err.D_SYNTAX.paren3.d mode=0444
file path=opt/SUNWdtrt/tst/common/lexer/tst.D_MACRO_OFLOW.ParIntOvflow.d.ksh \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_FACTOREVEN.nodivide.d
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_FACTOREVEN.notfactor.d
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_FACTORMATCH.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_FACTORNSTEPS.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_FACTORSMALL.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_FACTORTYPE.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_FACTORVAL.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_HIGHMATCH.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_HIGHTYPE.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_HIGHVAL.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_LOWMATCH.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_LOWTYPE.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_LOWVAL.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_MAGRANGE.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_MAGTOOBIG.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_NSTEPMATCH.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_NSTEPTYPE.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/err.D_LLQUANT_NSTEPVAL.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.bases.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.bases.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.basic.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.basic.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.negorder.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.negorder.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.negvalue.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.negvalue.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.normal.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.normal.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.range.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.range.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.steps.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.steps.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.trunc.d mode=0444
file path=opt/SUNWdtrt/tst/common/llquantize/tst.trunc.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/mdb/tst.dtracedcmd.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/mib/tst.icmp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/mib/tst.tcp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/mib/tst.udp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/misc/err.D_PRAGMA_OPTSET.d mode=0444
file path=opt/SUNWdtrt/tst/common/misc/tst.badopt.d mode=0444
file path=opt/SUNWdtrt/tst/common/misc/tst.boolopt.d mode=0444
file path=opt/SUNWdtrt/tst/common/misc/tst.boolopt.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/misc/tst.dynopt.d mode=0444
```

```
1047 file path=opt/SUNWdtrt/tst/common/misc/tst.dynopt.d.out mode=0444
1048 file path=opt/SUNWdtrt/tst/common/misc/tst.enablerace.ksh mode=0444
1049 file path=opt/SUNWdtrt/tst/common/misc/tst.haslam.d mode=0444
1050 file path=opt/SUNWdtrt/tst/common/misc/tst.include.ksh mode=0444
1051 file path=opt/SUNWdtrt/tst/common/misc/tst.macroglob.ksh mode=0444
1052 file path=opt/SUNWdtrt/tst/common/misc/tst.macroglob.ksh.out mode=0444
1053 file path=opt/SUNWdtrt/tst/common/misc/tst.roch.d mode=0444
1054 file path=opt/SUNWdtrt/tst/common/misc/tst.schrock.ksh mode=0444
1055 file path=opt/SUNWdtrt/tst/common/multiaggs/err.D_PRINTA_AGGKEY.d mode=0444
1056 file path=opt/SUNWdtrt/tst/common/multiaggs/err.D_PRINTA_AGGPROTO.d mode=0444
1057 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.many.d mode=0444
1058 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.many.d.out mode=0444
1059 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.same.d mode=0444
1060 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.same.d.out mode=0444
1061 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.sort.d mode=0444
1062 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.sort.d.out mode=0444
1063 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.sortpos.d mode=0444
1064 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.sortpos.d.out mode=0444
1065 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.tuplecompat.d mode=0444
1066 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.tuplecompat.d.out mode=0444
1067 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.zero.d mode=0444
1068 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.zero.d.out mode=0444
1069 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.zero2.d mode=0444
1070 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.zero2.d.out mode=0444
1071 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.zero3.d mode=0444
1072 file path=opt/SUNWdtrt/tst/common/multiaggs/tst.zero3.d.out mode=0444
1073 file path=opt/SUNWdtrt/tst/common/nfs/tst.call.d mode=0444
1074 file path=opt/SUNWdtrt/tst/common/nfs/tst.call.exe mode=0555
1075 file path=opt/SUNWdtrt/tst/common/nfs/tst.call3.d mode=0444
1076 file path=opt/SUNWdtrt/tst/common/nfs/tst.call3.exe mode=0555
1077 file path=opt/SUNWdtrt/tst/common/offsetof/err.D_OFFSETOF_BITFIELD.bitfield.d \
1078     mode=0444
1079 file path=opt/SUNWdtrt/tst/common/offsetof/err.D_OFFSETOF_TYPE.badtype.d \
1080     mode=0444
1081 file path=opt/SUNWdtrt/tst/common/offsetof/err.D_OFFSETOF_TYPE.notsou.d \
1082     mode=0444
1083 file path=opt/SUNWdtrt/tst/common/offsetof/err.D_UNKNOWN.OffsetofNULL.d \
1084     mode=0444
1085 file path=opt/SUNWdtrt/tst/common/offsetof/err.D_UNKNOWN.badmemb.d mode=0444
1086 file path=opt/SUNWdtrt/tst/common/offsetof/tst.OffsetofAlias.d mode=0444
1087 file path=opt/SUNWdtrt/tst/common/offsetof/tst.OffsetofArith.d mode=0444
1088 file path=opt/SUNWdtrt/tst/common/offsetof/tst.OffsetofUnion.d mode=0444
1089 file path=opt/SUNWdtrt/tst/common/offsetof/tst.struct.d mode=0444
1090 file path=opt/SUNWdtrt/tst/common/offsetof/tst.struct.d.out mode=0444
1091 file path=opt/SUNWdtrt/tst/common/offsetof/tst.union.d mode=0444
1092 file path=opt/SUNWdtrt/tst/common/offsetof/tst.union.d.out mode=0444
1093 file path=opt/SUNWdtrt/tst/common/operators/tst.ternary.d mode=0444
1094 file path=opt/SUNWdtrt/tst/common/operators/tst.ternary.d.out mode=0444
1095 file path=opt/SUNWdtrt/tst/common/pid/err.D_PDESC_ZERO.badlib.d mode=0444
1096 file path=opt/SUNWdtrt/tst/common/pid/err.D_PDESC_ZERO.badlib.exe mode=0555
1097 file path=opt/SUNWdtrt/tst/common/pid/err.D_PDESC_ZERO.badproc1.d mode=0444
1098 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_BADPID.badproc2.d mode=0444
1099 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_CREATEFAIL.many.d mode=0444
1100 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_CREATEFAIL.many.exe mode=0555
1101 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_FUNC.badfunc.d mode=0444
1102 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_FUNC.badfunc.exe mode=0555
1103 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_LIB.libdash.d mode=0444
1104 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_LIB.libdash.exe mode=0555
1105 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_NAME.alldash.d mode=0444
1106 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_NAME.alldash.exe mode=0555
1107 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_NAME.badname.d mode=0444
1108 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_NAME.badname.exe mode=0555
1109 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_NAME.globdash.d mode=0444
1110 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_NAME.globdash.exe mode=0555
1111 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_OFF.toobig.d mode=0444
1112 file path=opt/SUNWdtrt/tst/common/pid/err.D_PROC_OFF.toobig.exe mode=0555
```



```
1113 file path=opt/SUNWdtrt/tst/common/pid/tst.addprobes.ksh mode=0444
1114 file path=opt/SUNWdtrt/tst/common/pid/tst.args1.d mode=0444
1115 file path=opt/SUNWdtrt/tst/common/pid/tst.args1.exe mode=0555
1116 file path=opt/SUNWdtrt/tst/common/pid/tst.coverage.d mode=0444
1117 file path=opt/SUNWdtrt/tst/common/pid/tst.coverage.exe mode=0555
1118 file path=opt/SUNWdtrt/tst/common/pid/tst.emptystack.d mode=0444
1119 file path=opt/SUNWdtrt/tst/common/pid/tst.emptystack.d.out mode=0444
1120 file path=opt/SUNWdtrt/tst/common/pid/tst.emptystack.exe mode=0555
1121 file path=opt/SUNWdtrt/tst/common/pid/tst.float.d mode=0444
1122 file path=opt/SUNWdtrt/tst/common/pid/tst.float.exe mode=0555
1123 file path=opt/SUNWdtrt/tst/common/pid/tst.fork.d mode=0444
1124 file path=opt/SUNWdtrt/tst/common/pid/tst.fork.exe mode=0555
1125 file path=opt/SUNWdtrt/tst/common/pid/tst.gcc.d mode=0444
1126 file path=opt/SUNWdtrt/tst/common/pid/tst.gcc.exe mode=0555
1127 file path=opt/SUNWdtrt/tst/common/pid/tst.killonerror.ksh mode=0444
1128 file path=opt/SUNWdtrt/tst/common/pid/tst.main.ksh mode=0444
1129 file path=opt/SUNWdtrt/tst/common/pid/tst.manypids.ksh mode=0444
1130 file path=opt/SUNWdtrt/tst/common/pid/tst.newprobes.ksh mode=0444
1131 file path=opt/SUNWdtrt/tst/common/pid/tst.newprobes.ksh.out mode=0444
1132 file path=opt/SUNWdtrt/tst/common/pid/tst.probemod.ksh mode=0444
1133 file path=opt/SUNWdtrt/tst/common/pid/tst.provregex1.ksh mode=0444
1134 file path=opt/SUNWdtrt/tst/common/pid/tst.provregex2.ksh mode=0444
1135 file path=opt/SUNWdtrt/tst/common/pid/tst.provregex2.ksh.out mode=0444
1136 file path=opt/SUNWdtrt/tst/common/pid/tst.provregex3.ksh mode=0444
1137 file path=opt/SUNWdtrt/tst/common/pid/tst.provregex3.ksh.out mode=0444
1138 file path=opt/SUNWdtrt/tst/common/pid/tst.provregex4.ksh mode=0444
1139 file path=opt/SUNWdtrt/tst/common/pid/tst.provregex4.ksh.out mode=0444
1140 file path=opt/SUNWdtrt/tst/common/pid/tst.ret1.d mode=0444
1141 file path=opt/SUNWdtrt/tst/common/pid/tst.ret1.exe mode=0555
1142 file path=opt/SUNWdtrt/tst/common/pid/tst.ret2.d mode=0444
1143 file path=opt/SUNWdtrt/tst/common/pid/tst.ret2.exe mode=0555
1144 file path=opt/SUNWdtrt/tst/common/pid/tst.utf8probefunc.ksh mode=0444
1145 file path=opt/SUNWdtrt/tst/common/pid/tst.utf8probefunc.ksh.out mode=0444
1146 file path=opt/SUNWdtrt/tst/common/pid/tst.utf8probemod.ksh mode=0444
1147 file path=opt/SUNWdtrt/tst/common/pid/tst.utf8probemod.ksh.out mode=0444
1148 file path=opt/SUNWdtrt/tst/common/pid/tst.vfork.d mode=0444
1149 file path=opt/SUNWdtrt/tst/common/pid/tst.vfork.exe mode=0555
1150 file path=opt/SUNWdtrt/tst/common/pid/tst.weak1.d mode=0444
1151 file path=opt/SUNWdtrt/tst/common/pid/tst.weak1.exe mode=0555
1152 file path=opt/SUNWdtrt/tst/common/pid/tst.weak2.d mode=0444
1153 file path=opt/SUNWdtrt/tst/common/pid/tst.weak2.exe mode=0555
1154 file path=opt/SUNWdtrt/tst/common/plockstat/tst.available.d mode=0444
1155 file path=opt/SUNWdtrt/tst/common/plockstat/tst.available.exe mode=0555
1156 file path=opt/SUNWdtrt/tst/common/plockstat/tst.libmap.d mode=0444
1157 file path=opt/SUNWdtrt/tst/common/plockstat/tst.libmap.exe mode=0555
1158 file path=opt/SUNWdtrt/tst/common/pointers/err.BadAlign.d mode=0444
1159 file path=opt/SUNWdtrt/tst/common/pointers/err.D_ADDROF_VAR.ArrayVar.d \
1160     mode=0444
1161 file path=opt/SUNWdtrt/tst/common/pointers/err.D_ADDROF_VAR.DynamicVar.d \
1162     mode=0444
1163 file path=opt/SUNWdtrt/tst/common/pointers/err.D_ADDROF_VAR.agg.d mode=0444
1164 file path=opt/SUNWdtrt/tst/common/pointers/err.D_DEREF_NONPTR.noptr.d \
1165     mode=0444
1166 file path=opt/SUNWdtrt/tst/common/pointers/err.D_DEREF_VOID.VoidPointerDeref.d \
1167     mode=0444
1168 file path=opt/SUNWdtrt/tst/common/pointers/err.D_OP_ARRFUN.ArrayAssignment.d \
1169     mode=0444
1170 file \
1171     path=opt/SUNWdtrt/tst/common/pointers/err.D_OP_INCOMPAT.VoidPointerArith.d \
1172     mode=0444
1173 file path=opt/SUNWdtrt/tst/common/pointers/err.D_OP_LVAL.AddressChange.d \
1174     mode=0444
1175 file path=opt/SUNWdtrt/tst/common/pointers/err.D_OP_PTR.NonPointerAccess.d \
1176     mode=0444
1177 file path=opt/SUNWdtrt/tst/common/pointers/err.D_OP_PTR.badpointer.d mode=0444
1178 file path=opt/SUNWdtrt/tst/common/pointers/err.D_OP_SOU.BadPointerAccess.d \
```

```
    mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/err.D_OP_SOU.badpointer.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/err.InvalidAddress1.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/err.InvalidAddress2.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/err.InvalidAddress3.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/err.InvalidAddress4.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/err.InvalidAddress5.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.ArrayPointer1.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.ArrayPointer2.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.ArrayPointer3.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.GlobalVar.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.IntegerArithmetic1.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.PointerArithmetic1.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.PointerArithmetic2.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.PointerArithmetic3.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.PointerAssignment.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.ValidPointer1.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.ValidPointer2.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.VoidCast.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.assigncast1.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.assigncast2.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.basic1.d mode=0444
file path=opt/SUNWdtrt/tst/common/pointers/tst.basic2.d mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/err.D_PRAGERR.d mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/err.D_PRAGMA_DEPEND.main.d mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/err.D_PRAGMA_INVAL.d mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/err.D_PRAGMA_MALFORM.d mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/err.D_PRAGMA_UNUSED.UnusedPragma.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/err.circlibdep.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/err.invalidlibdep.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/tst.libchain.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/tst.libdep.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/tst.libdepfullyconnected.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/tst.libdepsepdir.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/tst.temporal.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/tst.temporal2.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/pragma/tst.temporal3.d mode=0444
file path=opt/SUNWdtrt/tst/common/predicates/err.D_PRED_SCALAR.NonScalarPred.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/predicates/err.D_SYNTAX.invalid.d mode=0444
file path=opt/SUNWdtrt/tst/common/predicates/err.D_SYNTAX.operr.d mode=0444
file path=opt/SUNWdtrt/tst/common/predicates/tst.argsnotcached.d mode=0444
file path=opt/SUNWdtrt/tst/common/predicates/tst.basics.d mode=0444
file path=opt/SUNWdtrt/tst/common/predicates/tst.basics.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/predicates/tst.complex.d mode=0444
file path=opt/SUNWdtrt/tst/common/predicates/tst.complex.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/err.D_IDENT_UNDEF.afterprobe.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/err.D_PRAGCTL_INVAL.tabdefine.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/err.D_SYNTAX.withoutpound.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/err.defincomp.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/err.ifdefelsenotendif.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/err.ifdefincomp.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/err.ifdefnotendif.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/err.incompelse.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/err.mulelse.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.ifdef.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.ifdef.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.ifndef.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.ifndef.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.ifnotdef.d mode=0444
```



```
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.ifnotdef.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.logicaland.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.logicaland.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.logicalandor.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.logicalandor.d.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.logicalor.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.logicalor.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.muland.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.muland.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.mulor.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.mulor.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.precondi.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.precondi.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.predicatedeclare.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.preexp.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.preexp.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.preexpelse.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.preexpelse.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.preexpif.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.preexpif.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.preexpifelse.d mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.preexpifelse.d.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/preprocessor/tst.withinprobe.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/err.D_PRINT_AGG.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/err.D_PRINT_VOID.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/err.D_PROTO_LEN.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.array.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.array.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.bitfield.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.bitfield.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.dyn.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.enum.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.enum.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.primitive.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.primitive.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.struct.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.struct.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.xlate.d mode=0444
file path=opt/SUNWdtrt/tst/common/print/tst.xlate.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printa/err.D_PRINTA_AGGARG.badagg.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printa/err.D_PRINTA_AGGARG.badfmt.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printa/err.D_PRINTA_AGGARG.badval.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printa/err.D_PRINTA_PROTO.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/printa/err.D_PRINTF_ARG_TYPE.jstack.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printa/err.D_PRINTF_ARG_TYPE.stack.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printa/err.D_PRINTF_ARG_TYPE.ustack.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.basics.d mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.basics.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.def.d mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.def.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.dynwidth.d mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.dynwidth.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.fmt.d mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.fmt.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.largeusersym.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.many.d mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.manyval.d mode=0444
```

```
file path=opt/SUNWdtrt/tst/common/printa/tst.manyval.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.stack.d mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.tuple.d mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.tuple.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.walltimestamp.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/printa/tst.walltimestamp.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_AGG_CONV.aggfmt.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_ARG_EXTRA.toomany.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_ARG_EXTRA.widths.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_ARG_FMT.badfmt.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_ARG_PROTO.novalue.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_ARG_TYPE.aggarg.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_ARG_TYPE.recursive.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_DYN_PROTO.noprec.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_DYN_PROTO.nowidth.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_DYN_TYPE.badprec.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PRINTF_DYN_TYPE.badwidth.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_PROTO_LEN.toofew.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_SYNTAX.badconv1.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_SYNTAX.badconv2.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/err.D_SYNTAX.badconv3.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.basics.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.basics.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.flags.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.flags.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.hello.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.hello.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.ints.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.ints.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.precs.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.precs.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.print-f.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.print-f.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.printT.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.printT.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.printY.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.printY.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.printcont.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.printcont.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.printeE.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.printeE.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.printgG.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.printgG.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.rawfmt.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.rawfmt.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.signs.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.signs.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.str.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.str.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.sym.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.sym.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.uints.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.uints.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.widths.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.widths.d.out mode=0444
```



```
file path=opt/SUNWdtrt/tst/common/printf/tst.widths1.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.wp.d mode=0444
file path=opt/SUNWdtrt/tst/common/printf/tst.wp.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/privs/tst.fds.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/privs/tst.func_access.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/privs/tst.getf.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/privs/tst.noprivdrop.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/privs/tst.noprivrestrict.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/privs/tst.op_access.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/privs/tst.procpriv.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/privs/tst.providers.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/privs/tst.tick.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/privs/tst.unpriv_funcs.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/probes/err.D_PDESC_ZERO.probeqtn.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/err.D_PDESC_ZERO.probestar.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/probes/err.D_PDESC_ZERO.tickstar.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/err.D_SYNTAX.assign.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/err.D_SYNTAX.declare.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/err.D_SYNTAX.declarein.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/err.D_SYNTAX.lbraces.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/err.D_SYNTAX.probespec.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/err.D_SYNTAX.rbraces.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/err.D_SYNTAX.recdec.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/tst.basic1.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/tst.check.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/tst.declare.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/tst.declareafter.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/tst.emptyprobe.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/tst.pragma.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/tst.pragmaaftertab.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/tst.pragmainside.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/tst.pragmaoutside.d mode=0444
file path=opt/SUNWdtrt/tst/common/probes/tst.probestar.d mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.create.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.discard.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.exec.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.execfail.ENOENT.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.execfail.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.exitcore.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.exitexit.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.exitkilled.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.signal.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.sigwait.d mode=0444
file path=opt/SUNWdtrt/tst/common/proc/tst.sigwait.exe mode=0555
file path=opt/SUNWdtrt/tst/common/proc/tst.startexit.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/err.D_PDESC_ZERO.profile.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/err.D_PDESC_ZEROonens.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/err.D_PDESC_ZEROonensec.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/err.D_PDESC_ZEROoneus.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/err.D_PDESC_ZEROoneusec.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.argtest.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.argtest.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.basic.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.basic.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.func.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.mod.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilehz.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilehz.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilems.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilems.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilemsec.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilemsec.d.out mode=0444
```

```
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilenhz.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilenhz.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilens.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilens.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilensec.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilensec.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profiles.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profiles.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilesec.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profilesec.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profileus.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profileus.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profileusec.d mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.profileusec.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.sym.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.ufunc.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.ufuncsort.exe mode=0555
file path=opt/SUNWdtrt/tst/common/profile-n/tst.ufuncsort.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.ufuncsort.ksh.out mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.umod.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/profile-n/tst.usym.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/providers/err.D_PDESC_INVAL.wrongdec4.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/providers/err.D_PDESC_ZERO.nonprofile.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/providers/err.D_PDESC_ZERO.wrongdec1.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/providers/err.D_PDESC_ZERO.wrongdec2.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/providers/err.D_PDESC_ZERO.wrongdec3.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.basics.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.basics.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.beginexit.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.beginprof.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.beginprof.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.probattrs.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.probattrs.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.probefunc.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.probefunc.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.probemod.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.probemod.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.probename.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.probename.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.probprov.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.probprov.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.profend.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.profend.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.profexit.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.profexit.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.trace.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.trace.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.twoprof.d mode=0444
file path=opt/SUNWdtrt/tst/common/providers/tst.twoprof.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/raise/tst.raise1.d mode=0444
file path=opt/SUNWdtrt/tst/common/raise/tst.raise1.exe mode=0555
file path=opt/SUNWdtrt/tst/common/raise/tst.raise2.d mode=0444
file path=opt/SUNWdtrt/tst/common/raise/tst.raise2.exe mode=0555
file path=opt/SUNWdtrt/tst/common/raise/tst.raise3.d mode=0444
file path=opt/SUNWdtrt/tst/common/raise/tst.raise3.exe mode=0555
file path=opt/SUNWdtrt/tst/common/rates/tst.aggrate.d mode=0444
file path=opt/SUNWdtrt/tst/common/rates/tst.aggrate.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/rates/tst.statusrate.d mode=0444
file path=opt/SUNWdtrt/tst/common/rates/tst.switchrate.d mode=0444
file path=opt/SUNWdtrt/tst/common/rates/tst.switchrate.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.basename.d mode=0444
```



```
file path=opt/SUNWdtrt/tst/common/safety/tst.caller.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.cleanpath.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.copyin.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.copyin2.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.ddi_pathname.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.dirname.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.errno.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.execname.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.gid.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.hton.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.index.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.msgdsize.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.msgsize.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.null.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.pid.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.ppid.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.progenyof.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.random.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.rw.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.shortstr.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.stack.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.stackdepth.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.stddev.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.strchr.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.strjoin.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.strstr.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.strtok.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.substr.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.ucaller.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.uid.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.unalign.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.uregs.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.ustack.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.ustackdepth.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.vahole.d mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.violentdeath.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/safety/tst.zonename.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/err.D_ARR_LOCAL.thisarray.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/err.D_DECL_CLASS.selfthis.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/err.D_DECL_CLASS.thisself.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/err.D_DECL_IDRED.errval.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/err.D_OP_INCOMPAT.dec.err.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/err.D_OP_INCOMPAT.dupgtype.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/err.D_OP_INCOMPAT.dupltype.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/err.D_OP_INCOMPAT.dupttype.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/err.D_SYNTAX.declare.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/tst.basicvar.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/tst.basicvar.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/tst.localvar.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/tst.misc.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/tst.self.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/tst.selfarray.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/tst.selfarray2.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/tst.selfthis.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/tst.this.d mode=0444
file path=opt/SUNWdtrt/tst/common/scalars/tst.thisself.d mode=0444
file path=opt/SUNWdtrt/tst/common/sched/tst.enqueue.d mode=0444
file path=opt/SUNWdtrt/tst/common/sched/tst.oncpu.d mode=0444
file path=opt/SUNWdtrt/tst/common/sched/tst.stackdepth.d mode=0444
```

```
file path=opt/SUNWdtrt/tst/common/scripting/err.D_MACRO_UNDEF.invalidargs.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/err.D_OP_LVAL.rdonly.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/err.D_OP_WRITE.usepidmacro.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/err.D_SYNTAX.concat.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/err.D_SYNTAX.desc.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/err.D_SYNTAX.inval.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/err.D_SYNTAX.pid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.D_MACRO_UNUSED.overflow.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.arg0.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.arguments.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.assign.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.basic.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.egid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.egid.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.euid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.euid.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.gid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.gid.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.pgid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.pid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.ppid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.ppid.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.projid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.projid.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.quite.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.sid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.sid.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.stringmacro.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.taskid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.taskid.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.trace.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.uid.d mode=0444
file path=opt/SUNWdtrt/tst/common/scripting/tst.uid.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/sdt/tst.sdtargs.d mode=0444
file path=opt/SUNWdtrt/tst/common/sdt/tst.sdtargs.exe mode=0555
file path=opt/SUNWdtrt/tst/common/sizeof/err.D_IDENT_BADREF.SizeofAssoc.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/err.D_IDENT_UNDEF.UnknownSymbol.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/err.D_SIZEOF_TYPE.badstruct.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/err.D_SIZEOF_TYPE.d mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/err.D_SYNTAX.SizeofBadType.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/tst.SizeofArray.d mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/tst.SizeofDataTypes.d mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/tst.SizeofExpression.d mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/tst.SizeofNULL.d mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/tst.SizeofStrConst.d mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/tst.SizeofStrConst.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/tst.SizeofString1.d mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/tst.SizeofString1.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/tst.SizeofString2.d mode=0444
file path=opt/SUNWdtrt/tst/common/sizeof/tst.SizeofString2.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.BufSizeVariations1.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.BufSizeVariations2.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_ACT_SPEC.SpeculateWithBreakPo
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_ACT_SPEC.SpeculateWithChill.d
```



```
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_ACT_SPEC.SpeculateWithCopyOut
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_ACT_SPEC.SpeculateWithCopyOut
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_ACT_SPEC.SpeculateWithPanic.d
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_ACT_SPEC.SpeculateWithRaise.d
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_ACT_SPEC.SpeculateWithStop.d
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.D_AGG_COMM.AggAftCommit.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_AGG_SPEC.SpeculateWithAvg.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_AGG_SPEC.SpeculateWithCount.d
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_AGG_SPEC.SpeculateWithLquant.
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_AGG_SPEC.SpeculateWithMax.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_AGG_SPEC.SpeculateWithMin.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_AGG_SPEC.SpeculateWithQuant.d
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_AGG_SPEC.SpeculateWithStddev.
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_AGG_SPEC.SpeculateWithSum.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_COMM_COMM.CommitAftCommit.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.D_COMM_COMM.DisjointCommit.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_COMM_DREC.CommitAftDataRec.d
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_DREC_COMM.DataRecAftCommit.d
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_DREC_COMM.ExitAfterCommit.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.D_EXIT_SPEC.ExitAftSpec.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.D_PRAGMA_MALFORM.NspecExpr.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_PRAGMA_OPTSET.HugeNspecValue.
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_PRAGMA_OPTSET.InvalidSpecSize
    mode=0444
```

```
file \
    path=opt/SUNWdtrt/tst/common/speculation/err.D_PRAGMA_OPTSET.NegSpecSize.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.D_PROTO_LEN.SpecNoId.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.D_SPEC_COMM.SpecAftCommit.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.D_SPEC_DREC.SpecAftDataRec.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.D_SPEC_SPEC.SpecAftSpec.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.NegativeBufSize.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.NegativeNspec.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.NegativeSpecSize.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.SpecSizeVariations1.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/err.SpecSizeVariations2.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.CommitAfterDiscard.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.CommitWithZero.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.DataRecAftDiscard.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.DiscardAftCommit.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.DiscardAftDataRec.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.DiscardAftDiscard.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.DiscardWithZero.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.ExitAftDiscard.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.NoSpecBuffer.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.SpecSizeVariations1.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.SpecSizeVariations2.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.SpecSizeVariations3.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.SpeculateWithRandom.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.SpeculationCommit.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.SpeculationDiscard.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.SpeculationID.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.SpeculationWithZero.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.TwoSpecBuffers.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.negcommit.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.negspec.d mode=0444
file path=opt/SUNWdtrt/tst/common/speculation/tst.zerosize.d mode=0444
file path=opt/SUNWdtrt/tst/common/stability/err.D_ATTR_MIN.MinAttributes.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/stack/err.D_STACK_PROTO.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/stack/err.D_STACK_SIZE.d mode=0444
file path=opt/SUNWdtrt/tst/common/stack/err.D_USTACK_FRAMES.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/stack/err.D_USTACK_PROTO.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/stack/err.D_USTACK_STRSIZE.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/stack/tst.default.d mode=0444
file path=opt/SUNWdtrt/tst/common/stackdepth/tst.default.d mode=0444
file path=opt/SUNWdtrt/tst/common/stop/tst.stop1.d mode=0444
file path=opt/SUNWdtrt/tst/common/stop/tst.stop1.exe mode=0555
file path=opt/SUNWdtrt/tst/common/stop/tst.stop2.d mode=0444
file path=opt/SUNWdtrt/tst/common/stop/tst.stop2.exe mode=0555
file path=opt/SUNWdtrt/tst/common/strlen/tst.strlen1.d mode=0444
file path=opt/SUNWdtrt/tst/common/strtoll/err.BaseTooLarge.d mode=0444
file path=opt/SUNWdtrt/tst/common/strtoll/err.BaseTooSmall.d mode=0444
```



```
file path=opt/SUNWdtrt/tst/common/strtoll/tst.strtoll.d mode=0444
file path=opt/SUNWdtrt/tst/common/strtoll/tst.strtoll.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/struct/err.D_ADDROF_VAR.StructPointer.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/struct/err.D_DECL_COMBO.StructWithoutColon.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/struct/err.D_DECL_COMBO.StructWithoutColon1.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/struct/err.D_DECL_INCOMPLETE.circular.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/struct/err.D_DECL_INCOMPLETE.order.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/struct/err.D_DECL_INCOMPLETE.order2.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/struct/err.D_DECL_INCOMPLETE.recursive.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/struct/err.D_DECL_INCOMPLETE.simple.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/struct/err.D_DECL_VOIDOBJ.baddec.d mode=0444
file path=opt/SUNWdtrt/tst/common/struct/err.D_PROTO_ARG.DupStructAssoc.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/struct/tst.StructAssoc.d mode=0444
file path=opt/SUNWdtrt/tst/common/struct/tst.StructDataTypes.d mode=0444
file path=opt/SUNWdtrt/tst/common/struct/tst.StructInside.d mode=0444
file path=opt/SUNWdtrt/tst/common/struct/tst.clauselocal.d mode=0444
file path=opt/SUNWdtrt/tst/common/struct/tst.clauselocal.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/syscall/tst.args.d mode=0444
file path=opt/SUNWdtrt/tst/common/syscall/tst.args.exe mode=0555
file path=opt/SUNWdtrt/tst/common/syscall/tst.openret.ksh mode=0444
file path=opt/SUNWdtrt/tst/common/sysevent/tst.post.d mode=0444
file path=opt/SUNWdtrt/tst/common/sysevent/tst.post.exe mode=0555
file path=opt/SUNWdtrt/tst/common/sysevent/tst.post_chan.d mode=0444
file path=opt/SUNWdtrt/tst/common/sysevent/tst.post_chan.exe mode=0555
file path=opt/SUNWdtrt/tst/common/tick-n/err.D_PDESC_ZERO.tick.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/err.D_PDESC_ZEROonens.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/err.D_PDESC_ZEROonensec.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/err.D_PDESC_ZEROoneus.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/err.D_PDESC_ZEROoneusec.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickarg0.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickms.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickms.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickmsec.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickmsec.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickns.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickns.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.ticknsec.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.ticknsec.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.ticks.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.ticks.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.ticksec.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.ticksec.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickus.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickus.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickusec.d mode=0444
file path=opt/SUNWdtrt/tst/common/tick-n/tst.tickusec.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/trace/err.D_PROTO_LEN.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/trace/err.D_TRACE_AGG.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/trace/err.D_TRACE_VOID.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/trace/tst.dyn.d mode=0444
file path=opt/SUNWdtrt/tst/common/trace/tst.misc.d mode=0444
file path=opt/SUNWdtrt/tst/common/trace/tst.qstring.d mode=0444
file path=opt/SUNWdtrt/tst/common/trace/tst.qstring.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/trace/tst.string.d mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/err.D_PROTO_ARG.badsize.d mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/err.D_PROTO_LEN.toofew.d mode=0444
```

```
file path=opt/SUNWdtrt/tst/common/tracemem/err.D_TRACEMEM_ADDR.badaddr.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/err.D_TRACEMEM_ARGS.d mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/err.D_TRACEMEM_DYNSIZE.d mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/err.D_TRACEMEM_SIZE.negsize.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/err.D_TRACEMEM_SIZE.zerosize.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/tst.dynsize.d mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/tst.dynsize.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/tst.rootvp.d mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/tst.smallsize.d mode=0444
file path=opt/SUNWdtrt/tst/common/tracemem/tst.smallsize.d.out mode=0444
file \
    path=opt/SUNWdtrt/tst/common/translators/err.D_DECL_TYPERED.BadTransDecl.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/translators/err.D_OP_INCOMPLETE.NonExistentInpu
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/err.D_SYNTAX.BadTransDecl1.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/err.D_SYNTAX.BadTransDecl3.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/err.D_SYNTAX.BadTransDecl4.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/translators/err.D_TYPE_MEMBER.NonExistentInput2
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/translators/err.D_XLATE_INCOMPAT.BadInputType1.
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/translators/err.D_XLATE_MEMB.NonExistentOutput2
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/err.D_XLATE_NONE.BadTransDecl6.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/translators/err.D_XLATE_REDECL.RepeatTransDecl.
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/err.D_XLATE_SOU.BadTransDecl8.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/err.D_XLATE_SOU.BadTransInt.d \
    mode=0444
file \
    path=opt/SUNWdtrt/tst/common/translators/err.D_XLATE_SOU.NonExistentOutput1.
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.CircularTransDecl.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.EmptyTransDecl.d mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.ForwardTag.d mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.InputAliasTrans.d mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.InputIntTrans.d mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.OutputAliasTrans.d mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.PartialDereferencing.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.PartialOutputTransDefn.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.ProcModelTrans.d mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.RepeatDeclaration.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.SimultaneousTranslators.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.StructureAssignment.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.TestTransStability1.ksh \
    mode=0444
```



```
file path=opt/SUNWdtrt/tst/common/translators/tst.TestTransStability1.ksh.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.TestTransStability2.ksh \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.TestTransStability2.ksh.out \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.TransNonPointer.d mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.TransOutputPointer.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.TransPointer.d mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.TranslateSelf.d mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.UnionInputTrans.d mode=0444
file path=opt/SUNWdtrt/tst/common/translators/tst.UnionOutputTrans.d mode=0444
file path=opt/SUNWdtrt/tst/common/typedef/err.D_DECL_IDRED.DupTypeDef.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/typedef/err.D_SYNTAX.BadExistingTypedef.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/typedef/err.D_SYNTAX.TypedefInClause.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/typedef/tst.ChainTypedef.d mode=0444
file path=opt/SUNWdtrt/tst/common/typedef/tst.TypedefDataAssign.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_CAST_INVAL.badcast.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_CG_DYN.ResultDynType.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_CHR_OFLOW.charconst.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_DECL_BADCLASS.bad.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_DECL_CHARATTR.badtype3.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_DECL_COMBO.badtype4.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_DECL_COMBO.badtype5.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_DECL_ENCONST.badeval.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_DECL_ENOFLOW.enoflow.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_DECL_ENOFLOW.enuflow.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_DECL_SCOPE.scopeop.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_DECL_USELESS.baddec.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_OP_ACT.badcond.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_OP_ARITH.badoperand.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_OP_INCOMPAT.badassign.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_OP_INT.badbitop.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_OP_INT.badshift.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_OP_SCALAR.badcond.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_OP_SCALAR.badincop.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_OP_SCALAR.badlogop.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_PROTO_LEN.badcond1.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_SYNTAX.badenum.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_SYNTAX.badid.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_SYNTAX.badstruct.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_UNKNOWN.badtype1.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_UNKNOWN.badtype2.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_UNKNOWN.dupenum.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_UNKNOWN.dupstruct.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/err.D_XLATE_REDECL.ResultDynType.d \
    mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.assignops.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.badshiftops.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.basics.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.basics.d.out mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.bitops.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.charconstants.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.complex.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.condexpr.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.const.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.constants.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.conv.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.enum.d mode=0444
file path=opt/SUNWdtrt/tst/common/types/tst.intincop.d mode=0444
```

```
1971 file path=opt/SUNWdtrt/tst/common/types/tst.intops.d mode=0444
1972 file path=opt/SUNWdtrt/tst/common/types/tst.inttypes.d mode=0444
1973 file path=opt/SUNWdtrt/tst/common/types/tst.ptrincop.d mode=0444
1974 file path=opt/SUNWdtrt/tst/common/types/tst.ptrops.d mode=0444
1975 file path=opt/SUNWdtrt/tst/common/types/tst.relenum.d mode=0444
1976 file path=opt/SUNWdtrt/tst/common/types/tst.relstring.d mode=0444
1977 file path=opt/SUNWdtrt/tst/common/types/tst.shiftops.d mode=0444
1978 file path=opt/SUNWdtrt/tst/common/types/tst.stringconstants.d mode=0444
1979 file path=opt/SUNWdtrt/tst/common/types/tst.struct.d mode=0444
1980 file path=opt/SUNWdtrt/tst/common/types/tst.typedef.d mode=0444
1981 file path=opt/SUNWdtrt/tst/common/types/tst.unaryop.d mode=0444
1982 file path=opt/SUNWdtrt/tst/common/uctf/err.invalidpid.d mode=0444
1983 file path=opt/SUNWdtrt/tst/common/uctf/err.invalidpid2.d mode=0444
1984 file path=opt/SUNWdtrt/tst/common/uctf/err.invalidpid3.d mode=0444
1985 file path=opt/SUNWdtrt/tst/common/uctf/err.invalidtype.ksh mode=0444
1986 file path=opt/SUNWdtrt/tst/common/uctf/err.invalidtype2.ksh mode=0444
1987 file path=opt/SUNWdtrt/tst/common/uctf/err.user64mode.ksh mode=0444
1988 file path=opt/SUNWdtrt/tst/common/uctf/tst.aouttype.exe mode=0555
1989 file path=opt/SUNWdtrt/tst/common/uctf/tst.aouttype.ksh mode=0444
1990 file path=opt/SUNWdtrt/tst/common/uctf/tst.chasestrings.exe mode=0555
1991 file path=opt/SUNWdtrt/tst/common/uctf/tst.chasestrings.ksh mode=0444
1992 file path=opt/SUNWdtrt/tst/common/uctf/tst.chasestrings.ksh.out mode=0444
1993 file path=opt/SUNWdtrt/tst/common/uctf/tst.libtype.exe mode=0555
1994 file path=opt/SUNWdtrt/tst/common/uctf/tst.libtype.ksh mode=0444
1995 file path=opt/SUNWdtrt/tst/common/uctf/tst.linkmap.ksh mode=0444
1996 file path=opt/SUNWdtrt/tst/common/uctf/tst.pidprint.ksh mode=0444
1997 file path=opt/SUNWdtrt/tst/common/uctf/tst.pidprinttarg.ksh mode=0444
1998 file path=opt/SUNWdtrt/tst/common/uctf/tst.printtype.exe mode=0555
1999 file path=opt/SUNWdtrt/tst/common/uctf/tst.printtype.ksh mode=0444
2000 file path=opt/SUNWdtrt/tst/common/uctf/tst.printtype.ksh.out mode=0444
2001 file path=opt/SUNWdtrt/tst/common/uctf/tst.printtypetarg.ksh mode=0444
2002 file path=opt/SUNWdtrt/tst/common/uctf/tst.userlandkey.ksh mode=0444
2003 file path=opt/SUNWdtrt/tst/common/uctf/tst.userlandkey.ksh.out mode=0444
2004 file path=opt/SUNWdtrt/tst/common/uctf/tst.userstrings.ksh mode=0444
2005 file path=opt/SUNWdtrt/tst/common/uctf/tst.userstrings.ksh.out mode=0444
2006 #endif /* ! codereview */
2007 file path=opt/SUNWdtrt/tst/common/union/err.D_ADDROF_VAR.UnionPointer.d \
2008     mode=0444
2009 file path=opt/SUNWdtrt/tst/common/union/err.D_DECL_COMBO.UnionWithoutColon.d \
2010     mode=0444
2011 file path=opt/SUNWdtrt/tst/common/union/err.D_DECL_COMBO.UnionWithoutColon1.d \
2012     mode=0444
2013 file path=opt/SUNWdtrt/tst/common/union/err.D_DECL_INCOMPLETE.circular.d \
2014     mode=0444
2015 file path=opt/SUNWdtrt/tst/common/union/err.D_DECL_INCOMPLETE.order.d \
2016     mode=0444
2017 file path=opt/SUNWdtrt/tst/common/union/err.D_DECL_INCOMPLETE.recursive.d \
2018     mode=0444
2019 file path=opt/SUNWdtrt/tst/common/union/err.D_DECL_INCOMPLETE.simple.d \
2020     mode=0444
2021 file path=opt/SUNWdtrt/tst/common/union/err.D_PROTO_ARG.DupUnionAssoc.d \
2022     mode=0444
2023 file path=opt/SUNWdtrt/tst/common/union/tst.UnionAssoc.d mode=0444
2024 file path=opt/SUNWdtrt/tst/common/union/tst.UnionDataTypes.d mode=0444
2025 file path=opt/SUNWdtrt/tst/common/union/tst.UnionInside.d mode=0444
2026 file path=opt/SUNWdtrt/tst/common/usdt/tst.andpid.ksh mode=0444
2027 file path=opt/SUNWdtrt/tst/common/usdt/tst.argmap.d mode=0444
2028 file path=opt/SUNWdtrt/tst/common/usdt/tst.argmap.exe mode=0555
2029 file path=opt/SUNWdtrt/tst/common/usdt/tst.args.d mode=0444
2030 file path=opt/SUNWdtrt/tst/common/usdt/tst.args.exe mode=0555
2031 file path=opt/SUNWdtrt/tst/common/usdt/tst.badguess.ksh mode=0444
2032 file path=opt/SUNWdtrt/tst/common/usdt/tst.corruptenv.ksh mode=0444
2033 file path=opt/SUNWdtrt/tst/common/usdt/tst.dlclose1.ksh mode=0444
2034 file path=opt/SUNWdtrt/tst/common/usdt/tst.dlclose1.ksh.out mode=0444
2035 file path=opt/SUNWdtrt/tst/common/usdt/tst.dlclose2.ksh mode=0444
2036 file path=opt/SUNWdtrt/tst/common/usdt/tst.dlclose2.ksh.out mode=0444
```



```
2037 file path=opt/SUNWdtrt/tst/common/usdt/tst.dlclose3.ksh mode=0444
2038 file path=opt/SUNWdtrt/tst/common/usdt/tst.eliminate.ksh mode=0444
2039 file path=opt/SUNWdtrt/tst/common/usdt/tst.enabled.ksh mode=0444
2040 file path=opt/SUNWdtrt/tst/common/usdt/tst.enabled.ksh.out mode=0444
2041 file path=opt/SUNWdtrt/tst/common/usdt/tst.enabled2.ksh mode=0444
2042 file path=opt/SUNWdtrt/tst/common/usdt/tst.enabled2.ksh.out mode=0444
2043 file path=opt/SUNWdtrt/tst/common/usdt/tst.entryreturn.ksh mode=0444
2044 file path=opt/SUNWdtrt/tst/common/usdt/tst.entryreturn.ksh.out mode=0444
2045 file path=opt/SUNWdtrt/tst/common/usdt/tst.fork.ksh mode=0444
2046 file path=opt/SUNWdtrt/tst/common/usdt/tst.fork.ksh.out mode=0444
2047 file path=opt/SUNWdtrt/tst/common/usdt/tst.forker.exe mode=0555
2048 file path=opt/SUNWdtrt/tst/common/usdt/tst.forker.ksh mode=0444
2049 file path=opt/SUNWdtrt/tst/common/usdt/tst.guess32.ksh mode=0444
2050 file path=opt/SUNWdtrt/tst/common/usdt/tst.guess64.ksh mode=0444
2051 file path=opt/SUNWdtrt/tst/common/usdt/tst.header.ksh mode=0444
2052 file path=opt/SUNWdtrt/tst/common/usdt/tst.include.ksh mode=0444
2053 file path=opt/SUNWdtrt/tst/common/usdt/tst.lazyprobe.exe mode=0555
2054 file path=opt/SUNWdtrt/tst/common/usdt/tst.lazyprobe1.ksh mode=0444
2055 file path=opt/SUNWdtrt/tst/common/usdt/tst.lazyprobe2.ksh mode=0444
2056 file path=opt/SUNWdtrt/tst/common/usdt/tst.linkpriv.ksh mode=0444
2057 file path=opt/SUNWdtrt/tst/common/usdt/tst.linkunpriv.ksh mode=0444
2058 file path=opt/SUNWdtrt/tst/common/usdt/tst.multiple.ksh mode=0444
2059 file path=opt/SUNWdtrt/tst/common/usdt/tst.multiple.ksh.out mode=0444
2060 file path=opt/SUNWdtrt/tst/common/usdt/tst.multiprov.ksh mode=0444
2061 file path=opt/SUNWdtrt/tst/common/usdt/tst.multiprov.ksh.out mode=0444
2062 file path=opt/SUNWdtrt/tst/common/usdt/tst.nodtrace.ksh mode=0444
2063 file path=opt/SUNWdtrt/tst/common/usdt/tst.noprobes.ksh mode=0444
2064 file path=opt/SUNWdtrt/tst/common/usdt/tst.noreap.ksh mode=0444
2065 file path=opt/SUNWdtrt/tst/common/usdt/tst.noreapring.ksh mode=0444
2066 file path=opt/SUNWdtrt/tst/common/usdt/tst.onlyenabled.ksh mode=0444
2067 file path=opt/SUNWdtrt/tst/common/usdt/tst.reap.ksh mode=0444
2068 file path=opt/SUNWdtrt/tst/common/usdt/tst.reeval.ksh mode=0444
2069 file path=opt/SUNWdtrt/tst/common/usdt/tst.static.ksh mode=0444
2070 file path=opt/SUNWdtrt/tst/common/usdt/tst.static.ksh.out mode=0444
2071 file path=opt/SUNWdtrt/tst/common/usdt/tst.static2.ksh mode=0444
2072 file path=opt/SUNWdtrt/tst/common/usdt/tst.static2.ksh.out mode=0444
2073 file path=opt/SUNWdtrt/tst/common/usdt/tst.user.ksh mode=0444
2074 file path=opt/SUNWdtrt/tst/common/usdt/tst.user.ksh.out mode=0444
2075 file path=opt/SUNWdtrt/tst/common/ustack/tst.bigstack.d mode=0444
2076 file path=opt/SUNWdtrt/tst/common/ustack/tst.bigstack.exe mode=0555
2077 file path=opt/SUNWdtrt/tst/common/ustack/tst.depth.ksh mode=0444
2078 file path=opt/SUNWdtrt/tst/common/ustack/tst.spin.exe mode=0555
2079 file path=opt/SUNWdtrt/tst/common/ustack/tst.spin.ksh mode=0444
2080 file path=opt/SUNWdtrt/tst/common/vars/tst.gid.d mode=0444
2081 file path=opt/SUNWdtrt/tst/common/vars/tst.nullassign.d mode=0444
2082 file path=opt/SUNWdtrt/tst/common/vars/tst.ppid.d mode=0444
2083 file path=opt/SUNWdtrt/tst/common/vars/tst.ucaller.ksh mode=0444
2084 file path=opt/SUNWdtrt/tst/common/vars/tst.ucaller.ksh.out mode=0444
2085 file path=opt/SUNWdtrt/tst/common/vars/tst.uid.d mode=0444
2086 file path=opt/SUNWdtrt/tst/common/vars/tst.walltimestamp.d mode=0444
2087 file path=opt/SUNWdtrt/tst/common/version/tst.1.0.d mode=0444
2088 $(i386_ONLY)file path=opt/SUNWdtrt/tst/i86xpv/xdt/tst.basic.ksh mode=0444
2089 $(i386_ONLY)file path=opt/SUNWdtrt/tst/i86xpv/xdt/tst.hvmenable.ksh mode=0444
2090 $(i386_ONLY)file path=opt/SUNWdtrt/tst/i86xpv/xdt/tst.memenable.ksh mode=0444
2091 $(i386_ONLY)file path=opt/SUNWdtrt/tst/i86xpv/xdt/tst.schedargs.ksh mode=0444
2092 $(i386_ONLY)file path=opt/SUNWdtrt/tst/i86xpv/xdt/tst.schedenable.ksh \
2093     mode=0444
2094 legacy pkg=SUNWdtrt category=internal \
2095     desc="DTrace Test Suite Internal Distribution" \
2096     hotline="Contact the DTrace discussion forum" name="DTrace Test Suite"
2097 license cr_Sun license=cr_Sun
2098 license lic_CDDL license=lic_CDDL
2099 depend fmri=runtime/java type=require
2100 depend fmri=runtime/java/runtime64 type=require
```

```
**********************************************************
  437125 Tue Jan 14 16:49:02 2014
new/usr/src/uts/common/dtrace/dtrace.c
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
______unchanged_portion_omitted_

5501 /*
5502  * Emulate the execution of DTrace IR instructions specified by the given
5503  * DIF object.  This function is deliberately void of assertions as all of
5504  * the necessary checks are handled by a call to dtrace_difo_validate().
5505  */
5506 static uint64_t
5507 dtrace_dif_emulate(dtrace_difo_t *difo, dtrace_mstate_t *mstate,
5508     dtrace_vstate_t *vstate, dtrace_state_t *state)
5509 {
5510         const dif_instr_t *text = difo->dtdo_buf;
5511         const uint_t textlen = difo->dtdo_len;
5512         const char *strtab = difo->dtdo_strtab;
5513         const uint64_t *inttab = difo->dtdo_inttab;

5515         uint64_t rval = 0;
5516         dtrace_statvar_t *svar;
5517         dtrace_dstate_t *dstate = &vstate->dtvs_dynvars;
5518         dtrace_difv_t *v;
5519         volatile uint16_t *flags = &cpu_core[CPU->cpu_id].cpuc_dtrace_flags;
5520         volatile uintptr_t *illval = &cpu_core[CPU->cpu_id].cpuc_dtrace_illval;

5522         dtrace_key_t tupregs[DIF_DTR_NREGS + 2]; /* +2 for thread and id */
5523         uint64_t regs[DIF_DIR_NREGS];
5524         uint64_t *tmp;

5526         uint8_t cc_n = 0, cc_z = 0, cc_v = 0, cc_c = 0;
5527         int64_t cc_r;
5528         uint_t pc = 0, id, opc;
5529         uint8_t ttop = 0;
5530         dif_instr_t instr;
5531         uint_t r1, r2, rd;

5533         /*
5534          * We stash the current DIF object into the machine state: we need it
5535          * for subsequent access checking.
5536          */
5537         mstate->dtms_difo = difo;

5539         regs[DIF_REG_R0] = 0;           /* %r0 is fixed at zero */

5541         while (pc < textlen && !(*flags & CPU_DTRACE_FAULT)) {
5542                 opc = pc;

5544                 instr = text[pc++];
5545                 r1 = DIF_INSTR_R1(instr);
5546                 r2 = DIF_INSTR_R2(instr);
5547                 rd = DIF_INSTR_RD(instr);

5549                 switch (DIF_INSTR_OP(instr)) {
5550                 case DIF_OP_OR:
5551                         regs[rd] = regs[r1] | regs[r2];
5552                         break;
5553                 case DIF_OP_XOR:
5554                         regs[rd] = regs[r1] ^ regs[r2];
```



```
5555                         break;
5556                 case DIF_OP_AND:
5557                         regs[rd] = regs[r1] & regs[r2];
5558                         break;
5559                 case DIF_OP_SLL:
5560                         regs[rd] = regs[r1] << regs[r2];
5561                         break;
5562                 case DIF_OP_SRL:
5563                         regs[rd] = regs[r1] >> regs[r2];
5564                         break;
5565                 case DIF_OP_SUB:
5566                         regs[rd] = regs[r1] - regs[r2];
5567                         break;
5568                 case DIF_OP_ADD:
5569                         regs[rd] = regs[r1] + regs[r2];
5570                         break;
5571                 case DIF_OP_MUL:
5572                         regs[rd] = regs[r1] * regs[r2];
5573                         break;
5574                 case DIF_OP_SDIV:
5575                         if (regs[r2] == 0) {
5576                                 regs[rd] = 0;
5577                                 *flags |= CPU_DTRACE_DIVZERO;
5578                         } else {
5579                                 regs[rd] = (int64_t)regs[r1] /
5580                                     (int64_t)regs[r2];
5581                         }
5582                         break;

5584                 case DIF_OP_UDIV:
5585                         if (regs[r2] == 0) {
5586                                 regs[rd] = 0;
5587                                 *flags |= CPU_DTRACE_DIVZERO;
5588                         } else {
5589                                 regs[rd] = regs[r1] / regs[r2];
5590                         }
5591                         break;

5593                 case DIF_OP_SREM:
5594                         if (regs[r2] == 0) {
5595                                 regs[rd] = 0;
5596                                 *flags |= CPU_DTRACE_DIVZERO;
5597                         } else {
5598                                 regs[rd] = (int64_t)regs[r1] %
5599                                     (int64_t)regs[r2];
5600                         }
5601                         break;

5603                 case DIF_OP_UREM:
5604                         if (regs[r2] == 0) {
5605                                 regs[rd] = 0;
5606                                 *flags |= CPU_DTRACE_DIVZERO;
5607                         } else {
5608                                 regs[rd] = regs[r1] % regs[r2];
5609                         }
5610                         break;

5612                 case DIF_OP_NOT:
5613                         regs[rd] = ~regs[r1];
5614                         break;
5615                 case DIF_OP_MOV:
5616                         regs[rd] = regs[r1];
5617                         break;
5618                 case DIF_OP_CMP:
5619                         cc_r = regs[r1] - regs[r2];
5620                         cc_n = cc_r < 0;
```

```
5621                        cc_z = cc_r == 0;
5622                        cc_v = 0;
5623                        cc_c = regs[r1] < regs[r2];
5624                        break;
5625                case DIF_OP_TST:
5626                        cc_n = cc_v = cc_c = 0;
5627                        cc_z = regs[r1] == 0;
5628                        break;
5629                case DIF_OP_BA:
5630                        pc = DIF_INSTR_LABEL(instr);
5631                        break;
5632                case DIF_OP_BE:
5633                        if (cc_z)
5634                                pc = DIF_INSTR_LABEL(instr);
5635                        break;
5636                case DIF_OP_BNE:
5637                        if (cc_z == 0)
5638                                pc = DIF_INSTR_LABEL(instr);
5639                        break;
5640                case DIF_OP_BG:
5641                        if ((cc_z | (cc_n ^ cc_v)) == 0)
5642                                pc = DIF_INSTR_LABEL(instr);
5643                        break;
5644                case DIF_OP_BGU:
5645                        if ((cc_c | cc_z) == 0)
5646                                pc = DIF_INSTR_LABEL(instr);
5647                        break;
5648                case DIF_OP_BGE:
5649                        if ((cc_n ^ cc_v) == 0)
5650                                pc = DIF_INSTR_LABEL(instr);
5651                        break;
5652                case DIF_OP_BGEU:
5653                        if (cc_c == 0)
5654                                pc = DIF_INSTR_LABEL(instr);
5655                        break;
5656                case DIF_OP_BL:
5657                        if (cc_n ^ cc_v)
5658                                pc = DIF_INSTR_LABEL(instr);
5659                        break;
5660                case DIF_OP_BLU:
5661                        if (cc_c)
5662                                pc = DIF_INSTR_LABEL(instr);
5663                        break;
5664                case DIF_OP_BLE:
5665                        if (cc_z | (cc_n ^ cc_v))
5666                                pc = DIF_INSTR_LABEL(instr);
5667                        break;
5668                case DIF_OP_BLEU:
5669                        if (cc_c | cc_z)
5670                                pc = DIF_INSTR_LABEL(instr);
5671                        break;
5672                case DIF_OP_RLDSB:
5673                        if (!dtrace_canload(regs[r1], 1, mstate, vstate))
5674                                break;
5675                        /*FALLTHROUGH*/
5676                case DIF_OP_LDSB:
5677                        regs[rd] = (int8_t)dtrace_load8(regs[r1]);
5678                        break;
5679                case DIF_OP_RLDSH:
5680                        if (!dtrace_canload(regs[r1], 2, mstate, vstate))
5681                                break;
5682                        /*FALLTHROUGH*/
5683                case DIF_OP_LDSH:
5684                        regs[rd] = (int16_t)dtrace_load16(regs[r1]);
5685                        break;
5686                case DIF_OP_RLDSW:
```



```
5687                        if (!dtrace_canload(regs[r1], 4, mstate, vstate))
5688                                break;
5689                        /*FALLTHROUGH*/
5690                case DIF_OP_LDSW:
5691                        regs[rd] = (int32_t)dtrace_load32(regs[r1]);
5692                        break;
5693                case DIF_OP_RLDUB:
5694                        if (!dtrace_canload(regs[r1], 1, mstate, vstate))
5695                                break;
5696                        /*FALLTHROUGH*/
5697                case DIF_OP_LDUB:
5698                        regs[rd] = dtrace_load8(regs[r1]);
5699                        break;
5700                case DIF_OP_RLDUH:
5701                        if (!dtrace_canload(regs[r1], 2, mstate, vstate))
5702                                break;
5703                        /*FALLTHROUGH*/
5704                case DIF_OP_LDUH:
5705                        regs[rd] = dtrace_load16(regs[r1]);
5706                        break;
5707                case DIF_OP_RLDUW:
5708                        if (!dtrace_canload(regs[r1], 4, mstate, vstate))
5709                                break;
5710                        /*FALLTHROUGH*/
5711                case DIF_OP_LDUW:
5712                        regs[rd] = dtrace_load32(regs[r1]);
5713                        break;
5714                case DIF_OP_RLDX:
5715                        if (!dtrace_canload(regs[r1], 8, mstate, vstate))
5716                                break;
5717                        /*FALLTHROUGH*/
5718                case DIF_OP_LDX:
5719                        regs[rd] = dtrace_load64(regs[r1]);
5720                        break;
5721                case DIF_OP_ULDSB:
5722                        DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
5723 #endif /* ! codereview */
5724                        regs[rd] = (int8_t)
5725                            dtrace_fuword8((void *)(uintptr_t)regs[r1]);
5726                        DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
5727 #endif /* ! codereview */
5728                        break;
5729                case DIF_OP_ULDSH:
5730                        DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
5731 #endif /* ! codereview */
5732                        regs[rd] = (int16_t)
5733                            dtrace_fuword16((void *)(uintptr_t)regs[r1]);
5734                        DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
5735 #endif /* ! codereview */
5736                        break;
5737                case DIF_OP_ULDSW:
5738                        DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
5739 #endif /* ! codereview */
5740                        regs[rd] = (int32_t)
5741                            dtrace_fuword32((void *)(uintptr_t)regs[r1]);
5742                        DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
5743 #endif /* ! codereview */
5744                        break;
5745                case DIF_OP_ULDUB:
5746                        DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
5747 #endif /* ! codereview */
5748                        regs[rd] =
5749                            dtrace_fuword8((void *)(uintptr_t)regs[r1]);
5750                        DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
5751 #endif /* ! codereview */
5752                        break;
```

```
5753                 case DIF_OP_ULDUH:
5754                         DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
5755 #endif /* ! codereview */
5756                         regs[rd] =
5757                             dtrace_fuword16((void *)(uintptr_t)regs[r1]);
5758                         DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
5759 #endif /* ! codereview */
5760                         break;
5761                 case DIF_OP_ULDUW:
5762                         DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
5763 #endif /* ! codereview */
5764                         regs[rd] =
5765                             dtrace_fuword32((void *)(uintptr_t)regs[r1]);
5766                         DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
5767 #endif /* ! codereview */
5768                         break;
5769                 case DIF_OP_ULDX:
5770                         DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
5771 #endif /* ! codereview */
5772                         regs[rd] =
5773                             dtrace_fuword64((void *)(uintptr_t)regs[r1]);
5774                         DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
5775 #endif /* ! codereview */
5776                         break;
5777                 case DIF_OP_RET:
5778                         rval = regs[rd];
5779                         pc = textlen;
5780                         break;
5781                 case DIF_OP_NOP:
5782                         break;
5783                 case DIF_OP_SETX:
5784                         regs[rd] = inttab[DIF_INSTR_INTEGER(instr)];
5785                         break;
5786                 case DIF_OP_SETS:
5787                         regs[rd] = (uint64_t)(uintptr_t)
5788                             (strtab + DIF_INSTR_STRING(instr));
5789                         break;
5790                 case DIF_OP_SCMP: {
5791                         size_t sz = state->dts_options[DTRACEOPT_STRSIZE];
5792                         uintptr_t s1 = regs[r1];
5793                         uintptr_t s2 = regs[r2];
5795                         if (s1 != NULL &&
5796                             !dtrace_strcanload(s1, sz, mstate, vstate))
5797                                 break;
5798                         if (s2 != NULL &&
5799                             !dtrace_strcanload(s2, sz, mstate, vstate))
5800                                 break;
5802                         cc_r = dtrace_strncmp((char *)s1, (char *)s2, sz);
5804                         cc_n = cc_r < 0;
5805                         cc_z = cc_r == 0;
5806                         cc_v = cc_c = 0;
5807                         break;
5808                 }
5809                 case DIF_OP_LDGA:
5810                         regs[rd] = dtrace_dif_variable(mstate, state,
5811                             r1, regs[r2]);
5812                         break;
5813                 case DIF_OP_LDGS:
5814                         id = DIF_INSTR_VAR(instr);
5816                         if (id >= DIF_VAR_OTHER_UBASE) {
5817                                 uintptr_t a;
```



```
5819                                 id -= DIF_VAR_OTHER_UBASE;
5820                                 svar = vstate->dtvs_globals[id];
5821                                 ASSERT(svar != NULL);
5822                                 v = &svar->dtsv_var;
5824                                 if (!(v->dtdv_type.dtdt_flags & DIF_TF_BYREF)) {
5825                                         regs[rd] = svar->dtsv_data;
5826                                         break;
5827                                 }
5829                                 a = (uintptr_t)svar->dtsv_data;
5831                                 if (*(uint8_t *)a == UINT8_MAX) {
5832                                         /*
5833                                          * If the 0th byte is set to UINT8_MAX
5834                                          * then this is to be treated as a
5835                                          * reference to a NULL variable.
5836                                          */
5837                                         regs[rd] = NULL;
5838                                 } else {
5839                                         regs[rd] = a + sizeof (uint64_t);
5840                                 }
5842                                 break;
5843                         }
5845                         regs[rd] = dtrace_dif_variable(mstate, state, id, 0);
5846                         break;
5848                 case DIF_OP_STGS:
5849                         id = DIF_INSTR_VAR(instr);
5851                         ASSERT(id >= DIF_VAR_OTHER_UBASE);
5852                         id -= DIF_VAR_OTHER_UBASE;
5854                         svar = vstate->dtvs_globals[id];
5855                         ASSERT(svar != NULL);
5856                         v = &svar->dtsv_var;
5858                         if (v->dtdv_type.dtdt_flags & DIF_TF_BYREF) {
5859                                 uintptr_t a = (uintptr_t)svar->dtsv_data;
5861                                 ASSERT(a != NULL);
5862                                 ASSERT(svar->dtsv_size != 0);
5864                                 if (regs[rd] == NULL) {
5865                                         *(uint8_t *)a = UINT8_MAX;
5866                                         break;
5867                                 } else {
5868                                         *(uint8_t *)a = 0;
5869                                         a += sizeof (uint64_t);
5870                                 }
5871                                 if (!dtrace_vcanload(
5872                                     (void *)(uintptr_t)regs[rd], &v->dtdv_type,
5873                                     mstate, vstate))
5874                                         break;
5876                                 dtrace_vcopy((void *)(uintptr_t)regs[rd],
5877                                     (void *)a, &v->dtdv_type);
5878                                 break;
5879                         }
5881                         svar->dtsv_data = regs[rd];
5882                         break;
5884                 case DIF_OP_LDTA:
```

```
5885                        /*
5886                         * There are no DTrace built-in thread-local arrays at
5887                         * present.  This opcode is saved for future work.
5888                         */
5889                        *flags |= CPU_DTRACE_ILLOP;
5890                        regs[rd] = 0;
5891                        break;

5893                case DIF_OP_LDLS:
5894                        id = DIF_INSTR_VAR(instr);

5896                        if (id < DIF_VAR_OTHER_UBASE) {
5897                                /*
5898                                 * For now, this has no meaning.
5899                                 */
5900                                regs[rd] = 0;
5901                                break;
5902                        }

5904                        id -= DIF_VAR_OTHER_UBASE;

5906                        ASSERT(id < vstate->dtvs_nlocals);
5907                        ASSERT(vstate->dtvs_locals != NULL);

5909                        svar = vstate->dtvs_locals[id];
5910                        ASSERT(svar != NULL);
5911                        v = &svar->dtsv_var;

5913                        if (v->dtdv_type.dtdt_flags & DIF_TF_BYREF) {
5914                                uintptr_t a = (uintptr_t)svar->dtsv_data;
5915                                size_t sz = v->dtdv_type.dtdt_size;

5917                                sz += sizeof (uint64_t);
5918                                ASSERT(svar->dtsv_size == NCPU * sz);
5919                                a += CPU->cpu_id * sz;

5921                                if (*(uint8_t *)a == UINT8_MAX) {
5922                                        /*
5923                                         * If the 0th byte is set to UINT8_MAX
5924                                         * then this is to be treated as a
5925                                         * reference to a NULL variable.
5926                                         */
5927                                        regs[rd] = NULL;
5928                                } else {
5929                                        regs[rd] = a + sizeof (uint64_t);
5930                                }

5932                                break;
5933                        }

5935                        ASSERT(svar->dtsv_size == NCPU * sizeof (uint64_t));
5936                        tmp = (uint64_t *)(uintptr_t)svar->dtsv_data;
5937                        regs[rd] = tmp[CPU->cpu_id];
5938                        break;

5940                case DIF_OP_STLS:
5941                        id = DIF_INSTR_VAR(instr);

5943                        ASSERT(id >= DIF_VAR_OTHER_UBASE);
5944                        id -= DIF_VAR_OTHER_UBASE;
5945                        ASSERT(id < vstate->dtvs_nlocals);

5947                        ASSERT(vstate->dtvs_locals != NULL);
5948                        svar = vstate->dtvs_locals[id];
5949                        ASSERT(svar != NULL);
5950                        v = &svar->dtsv_var;
```



```
5952                        if (v->dtdv_type.dtdt_flags & DIF_TF_BYREF) {
5953                                uintptr_t a = (uintptr_t)svar->dtsv_data;
5954                                size_t sz = v->dtdv_type.dtdt_size;

5956                                sz += sizeof (uint64_t);
5957                                ASSERT(svar->dtsv_size == NCPU * sz);
5958                                a += CPU->cpu_id * sz;

5960                                if (regs[rd] == NULL) {
5961                                        *(uint8_t *)a = UINT8_MAX;
5962                                        break;
5963                                } else {
5964                                        *(uint8_t *)a = 0;
5965                                        a += sizeof (uint64_t);
5966                                }

5968                                if (!dtrace_vcanload(
5969                                    (void *)(uintptr_t)regs[rd], &v->dtdv_type,
5970                                    mstate, vstate))
5971                                        break;

5973                                dtrace_vcopy((void *)(uintptr_t)regs[rd],
5974                                    (void *)a, &v->dtdv_type);
5975                                break;
5976                        }

5978                        ASSERT(svar->dtsv_size == NCPU * sizeof (uint64_t));
5979                        tmp = (uint64_t *)(uintptr_t)svar->dtsv_data;
5980                        tmp[CPU->cpu_id] = regs[rd];
5981                        break;

5983                case DIF_OP_LDTS: {
5984                        dtrace_dynvar_t *dvar;
5985                        dtrace_key_t *key;

5987                        id = DIF_INSTR_VAR(instr);
5988                        ASSERT(id >= DIF_VAR_OTHER_UBASE);
5989                        id -= DIF_VAR_OTHER_UBASE;
5990                        v = &vstate->dtvs_tlocals[id];

5992                        key = &tupregs[DIF_DTR_NREGS];
5993                        key[0].dttk_value = (uint64_t)id;
5994                        key[0].dttk_size = 0;
5995                        DTRACE_TLS_THRKEY(key[1].dttk_value);
5996                        key[1].dttk_size = 0;

5998                        dvar = dtrace_dynvar(dstate, 2, key,
5999                            sizeof (uint64_t), DTRACE_DYNVAR_NOALLOC,
6000                            mstate, vstate);

6002                        if (dvar == NULL) {
6003                                regs[rd] = 0;
6004                                break;
6005                        }

6007                        if (v->dtdv_type.dtdt_flags & DIF_TF_BYREF) {
6008                                regs[rd] = (uint64_t)(uintptr_t)dvar->dtdv_data;
6009                        } else {
6010                                regs[rd] = *((uint64_t *)dvar->dtdv_data);
6011                        }

6013                        break;
6014                }

6016                case DIF_OP_STTS: {
```

```
6017                        dtrace_dynvar_t *dvar;
6018                        dtrace_key_t *key;

6020                        id = DIF_INSTR_VAR(instr);
6021                        ASSERT(id >= DIF_VAR_OTHER_UBASE);
6022                        id -= DIF_VAR_OTHER_UBASE;

6024                        key = &tupregs[DIF_DTR_NREGS];
6025                        key[0].dttk_value = (uint64_t)id;
6026                        key[0].dttk_size = 0;
6027                        DTRACE_TLS_THRKEY(key[1].dttk_value);
6028                        key[1].dttk_size = 0;
6029                        v = &vstate->dtvs_tlocals[id];

6031                        dvar = dtrace_dynvar(dstate, 2, key,
6032                            v->dtdv_type.dtdt_size > sizeof (uint64_t) ?
6033                            v->dtdv_type.dtdt_size : sizeof (uint64_t),
6034                            regs[rd] ? DTRACE_DYNVAR_ALLOC :
6035                            DTRACE_DYNVAR_DEALLOC, mstate, vstate);

6037                        /*
6038                         * Given that we're storing to thread-local data,
6039                         * we need to flush our predicate cache.
6040                         */
6041                        curthread->t_predcache = NULL;

6043                        if (dvar == NULL)
6044                                break;

6046                        if (v->dtdv_type.dtdt_flags & DIF_TF_BYREF) {
6047                                if (!dtrace_vcanload(
6048                                    (void *)(uintptr_t)regs[rd],
6049                                    &v->dtdv_type, mstate, vstate))
6050                                        break;

6052                                dtrace_vcopy((void *)(uintptr_t)regs[rd],
6053                                    dvar->dtdv_data, &v->dtdv_type);
6054                        } else {
6055                                *((uint64_t *)dvar->dtdv_data) = regs[rd];
6056                        }

6058                        break;
6059                }

6061                case DIF_OP_SRA:
6062                        regs[rd] = (int64_t)regs[r1] >> regs[r2];
6063                        break;

6065                case DIF_OP_CALL:
6066                        dtrace_dif_subr(DIF_INSTR_SUBR(instr), rd,
6067                            regs, tupregs, ttop, mstate, state);
6068                        break;

6070                case DIF_OP_PUSHTR:
6071                        if (ttop == DIF_DTR_NREGS) {
6072                                *flags |= CPU_DTRACE_TUPOFLOW;
6073                                break;
6074                        }

6076                        if (r1 == DIF_TYPE_STRING) {
6077                                /*
6078                                 * If this is a string type and the size is 0,
6079                                 * we'll use the system-wide default string
6080                                 * size.  Note that we are _not_ looking at
6081                                 * the value of the DTRACEOPT_STRSIZE option;
6082                                 * had this been set, we would expect to have
```



```
6083                                 * a non-zero size value in the "pushtr".
6084                                 */
6085                                tupregs[ttop].dttk_size =
6086                                    dtrace_strlen((char *)(uintptr_t)regs[rd],
6087                                    regs[r2] ? regs[r2] :
6088                                    dtrace_strsize_default) + 1;
6089                        } else {
6090                                tupregs[ttop].dttk_size = regs[r2];
6091                        }

6093                        tupregs[ttop++].dttk_value = regs[rd];
6094                        break;

6096                case DIF_OP_PUSHTV:
6097                        if (ttop == DIF_DTR_NREGS) {
6098                                *flags |= CPU_DTRACE_TUPOFLOW;
6099                                break;
6100                        }

6102                        tupregs[ttop].dttk_value = regs[rd];
6103                        tupregs[ttop++].dttk_size = 0;
6104                        break;

6106                case DIF_OP_POPTS:
6107                        if (ttop != 0)
6108                                ttop--;
6109                        break;

6111                case DIF_OP_FLUSHTS:
6112                        ttop = 0;
6113                        break;

6115                case DIF_OP_LDGAA:
6116                case DIF_OP_LDTAA: {
6117                        dtrace_dynvar_t *dvar;
6118                        dtrace_key_t *key = tupregs;
6119                        uint_t nkeys = ttop;

6121                        id = DIF_INSTR_VAR(instr);
6122                        ASSERT(id >= DIF_VAR_OTHER_UBASE);
6123                        id -= DIF_VAR_OTHER_UBASE;

6125                        key[nkeys].dttk_value = (uint64_t)id;
6126                        key[nkeys++].dttk_size = 0;

6128                        if (DIF_INSTR_OP(instr) == DIF_OP_LDTAA) {
6129                                DTRACE_TLS_THRKEY(key[nkeys].dttk_value);
6130                                key[nkeys++].dttk_size = 0;
6131                                v = &vstate->dtvs_tlocals[id];
6132                        } else {
6133                                v = &vstate->dtvs_globals[id]->dtsv_var;
6134                        }

6136                        dvar = dtrace_dynvar(dstate, nkeys, key,
6137                            v->dtdv_type.dtdt_size > sizeof (uint64_t) ?
6138                            v->dtdv_type.dtdt_size : sizeof (uint64_t),
6139                            DTRACE_DYNVAR_NOALLOC, mstate, vstate);

6141                        if (dvar == NULL) {
6142                                regs[rd] = 0;
6143                                break;
6144                        }

6146                        if (v->dtdv_type.dtdt_flags & DIF_TF_BYREF) {
6147                                regs[rd] = (uint64_t)(uintptr_t)dvar->dtdv_data;
6148                        } else {
```

```
6149                            regs[rd] = *((uint64_t *)dvar->dtdv_data);
6150                    }
6151
6152                    break;
6153            }
6154
6155            case DIF_OP_STGAA:
6156            case DIF_OP_STTAA: {
6157                    dtrace_dynvar_t *dvar;
6158                    dtrace_key_t *key = tupregs;
6159                    uint_t nkeys = ttop;
6160
6161                    id = DIF_INSTR_VAR(instr);
6162                    ASSERT(id >= DIF_VAR_OTHER_UBASE);
6163                    id -= DIF_VAR_OTHER_UBASE;
6164
6165                    key[nkeys].dttk_value = (uint64_t)id;
6166                    key[nkeys++].dttk_size = 0;
6167
6168                    if (DIF_INSTR_OP(instr) == DIF_OP_STTAA) {
6169                            DTRACE_TLS_THRKEY(key[nkeys].dttk_value);
6170                            key[nkeys++].dttk_size = 0;
6171                            v = &vstate->dtvs_tlocals[id];
6172                    } else {
6173                            v = &vstate->dtvs_globals[id]->dtsv_var;
6174                    }
6175
6176                    dvar = dtrace_dynvar(dstate, nkeys, key,
6177                        v->dtdv_type.dtdt_size > sizeof (uint64_t) ?
6178                        v->dtdv_type.dtdt_size : sizeof (uint64_t),
6179                        regs[rd] ? DTRACE_DYNVAR_ALLOC :
6180                        DTRACE_DYNVAR_DEALLOC, mstate, vstate);
6181
6182                    if (dvar == NULL)
6183                            break;
6184
6185                    if (v->dtdv_type.dtdt_flags & DIF_TF_BYREF) {
6186                            if (!dtrace_vcanload(
6187                                (void *)(uintptr_t)regs[rd], &v->dtdv_type,
6188                                mstate, vstate))
6189                                    break;
6190
6191                            dtrace_vcopy((void *)(uintptr_t)regs[rd],
6192                                dvar->dtdv_data, &v->dtdv_type);
6193                    } else {
6194                            *((uint64_t *)dvar->dtdv_data) = regs[rd];
6195                    }
6196
6197                    break;
6198            }
6199
6200            case DIF_OP_ALLOCS: {
6201                    uintptr_t ptr = P2ROUNDUP(mstate->dtms_scratch_ptr, 8);
6202                    size_t size = ptr - mstate->dtms_scratch_ptr + regs[r1];
6203
6204                    /*
6205                     * Rounding up the user allocation size could have
6206                     * overflowed large, bogus allocations (like -1ULL) to
6207                     * 0.
6208                     */
6209                    if (size < regs[r1] ||
6210                        !DTRACE_INSCRATCH(mstate, size)) {
6211                            DTRACE_CPUFLAG_SET(CPU_DTRACE_NOSCRATCH);
6212                            regs[rd] = NULL;
6213                            break;
6214                    }
```



```
6216                            dtrace_bzero((void *) mstate->dtms_scratch_ptr, size);
6217                            mstate->dtms_scratch_ptr += size;
6218                            regs[rd] = ptr;
6219                            break;
6220                    }
6221
6222                    case DIF_OP_COPYS:
6223                            if (!dtrace_canstore(regs[rd], regs[r2],
6224                                mstate, vstate)) {
6225                                    *flags |= CPU_DTRACE_BADADDR;
6226                                    *illval = regs[rd];
6227                                    break;
6228                            }
6229
6230                            if (!dtrace_canload(regs[r1], regs[r2], mstate, vstate))
6231                                    break;
6232
6233                            dtrace_bcopy((void *)(uintptr_t)regs[r1],
6234                                (void *)(uintptr_t)regs[rd], (size_t)regs[r2]);
6235                            break;
6236
6237                    case DIF_OP_STB:
6238                            if (!dtrace_canstore(regs[rd], 1, mstate, vstate)) {
6239                                    *flags |= CPU_DTRACE_BADADDR;
6240                                    *illval = regs[rd];
6241                                    break;
6242                            }
6243                            *((uint8_t *)(uintptr_t)regs[rd]) = (uint8_t)regs[r1];
6244                            break;
6245
6246                    case DIF_OP_STH:
6247                            if (!dtrace_canstore(regs[rd], 2, mstate, vstate)) {
6248                                    *flags |= CPU_DTRACE_BADADDR;
6249                                    *illval = regs[rd];
6250                                    break;
6251                            }
6252                            if (regs[rd] & 1) {
6253                                    *flags |= CPU_DTRACE_BADALIGN;
6254                                    *illval = regs[rd];
6255                                    break;
6256                            }
6257                            *((uint16_t *)(uintptr_t)regs[rd]) = (uint16_t)regs[r1];
6258                            break;
6259
6260                    case DIF_OP_STW:
6261                            if (!dtrace_canstore(regs[rd], 4, mstate, vstate)) {
6262                                    *flags |= CPU_DTRACE_BADADDR;
6263                                    *illval = regs[rd];
6264                                    break;
6265                            }
6266                            if (regs[rd] & 3) {
6267                                    *flags |= CPU_DTRACE_BADALIGN;
6268                                    *illval = regs[rd];
6269                                    break;
6270                            }
6271                            *((uint32_t *)(uintptr_t)regs[rd]) = (uint32_t)regs[r1];
6272                            break;
6273
6274                    case DIF_OP_STX:
6275                            if (!dtrace_canstore(regs[rd], 8, mstate, vstate)) {
6276                                    *flags |= CPU_DTRACE_BADADDR;
6277                                    *illval = regs[rd];
6278                                    break;
6279                            }
6280                            if (regs[rd] & 7) {
```

```
6281                                *flags |= CPU_DTRACE_BADALIGN;
6282                                *illval = regs[rd];
6283                                break;
6284                        }
6285                        *((uint64_t *)(uintptr_t)regs[rd]) = regs[r1];
6286                        break;
6287                }
6288        }

6290        if (!(*flags & CPU_DTRACE_FAULT))
6291                return (rval);

6293        mstate->dtms_fltoffs = opc * sizeof (dif_instr_t);
6294        mstate->dtms_present |= DTRACE_MSTATE_FLTOFFS;

6296        return (0);
6297 }

6299 static void
6300 dtrace_action_breakpoint(dtrace_ecb_t *ecb)
6301 {
6302        dtrace_probe_t *probe = ecb->dte_probe;
6303        dtrace_provider_t *prov = probe->dtpr_provider;
6304        char c[DTRACE_FULLNAMELEN + 80], *str;
6305        char *msg = "dtrace: breakpoint action at probe ";
6306        char *ecbmsg = " (ecb ";
6307        uintptr_t mask = (0xf << (sizeof (uintptr_t) * NBBY / 4));
6308        uintptr_t val = (uintptr_t)ecb;
6309        int shift = (sizeof (uintptr_t) * NBBY) - 4, i = 0;

6311        if (dtrace_destructive_disallow)
6312                return;

6314        /*
6315         * It's impossible to be taking action on the NULL probe.
6316         */
6317        ASSERT(probe != NULL);

6319        /*
6320         * This is a poor man's (destitute man's?) sprintf():  we want to
6321         * print the provider name, module name, function name and name of
6322         * the probe, along with the hex address of the ECB with the breakpoint
6323         * action -- all of which we must place in the character buffer by
6324         * hand.
6325         */
6326        while (*msg != '\0')
6327                c[i++] = *msg++;

6329        for (str = prov->dtpv_name; *str != '\0'; str++)
6330                c[i++] = *str;
6331        c[i++] = ':';

6333        for (str = probe->dtpr_mod; *str != '\0'; str++)
6334                c[i++] = *str;
6335        c[i++] = ':';

6337        for (str = probe->dtpr_func; *str != '\0'; str++)
6338                c[i++] = *str;
6339        c[i++] = ':';

6341        for (str = probe->dtpr_name; *str != '\0'; str++)
6342                c[i++] = *str;

6344        while (*ecbmsg != '\0')
6345                c[i++] = *ecbmsg++;
```



```
6347        while (shift >= 0) {
6348                mask = (uintptr_t)0xf << shift;

6350                if (val >= ((uintptr_t)1 << shift))
6351                        c[i++] = "0123456789abcdef"[(val & mask) >> shift];
6352                shift -= 4;
6353        }

6355        c[i++] = ')';
6356        c[i] = '\0';

6358        debug_enter(c);
6359 }

6361 static void
6362 dtrace_action_panic(dtrace_ecb_t *ecb)
6363 {
6364        dtrace_probe_t *probe = ecb->dte_probe;

6366        /*
6367         * It's impossible to be taking action on the NULL probe.
6368         */
6369        ASSERT(probe != NULL);

6371        if (dtrace_destructive_disallow)
6372                return;

6374        if (dtrace_panicked != NULL)
6375                return;

6377        if (dtrace_casptr(&dtrace_panicked, NULL, curthread) != NULL)
6378                return;

6380        /*
6381         * We won the right to panic.  (We want to be sure that only one
6382         * thread calls panic() from dtrace_probe(), and that panic() is
6383         * called exactly once.)
6384         */
6385        dtrace_panic("dtrace: panic action at probe %s:%s:%s:%s (ecb %p)",
6386            probe->dtpr_provider->dtpv_name, probe->dtpr_mod,
6387            probe->dtpr_func, probe->dtpr_name, (void *)ecb);
6388 }

6390 static void
6391 dtrace_action_raise(uint64_t sig)
6392 {
6393        if (dtrace_destructive_disallow)
6394                return;

6396        if (sig >= NSIG) {
6397                DTRACE_CPUFLAG_SET(CPU_DTRACE_ILLOP);
6398                return;
6399        }

6401        /*
6402         * raise() has a queue depth of 1 -- we ignore all subsequent
6403         * invocations of the raise() action.
6404         */
6405        if (curthread->t_dtrace_sig == 0)
6406                curthread->t_dtrace_sig = (uint8_t)sig;

6408        curthread->t_sig_check = 1;
6409        aston(curthread);
6410 }

6412 static void
```

```
6413 dtrace_action_stop(void)
6414 {
6415         if (dtrace_destructive_disallow)
6416                 return;
6417
6418         if (!curthread->t_dtrace_stop) {
6419                 curthread->t_dtrace_stop = 1;
6420                 curthread->t_sig_check = 1;
6421                 aston(curthread);
6422         }
6423 }
6424
6425 static void
6426 dtrace_action_chill(dtrace_mstate_t *mstate, hrtime_t val)
6427 {
6428         hrtime_t now;
6429         volatile uint16_t *flags;
6430         cpu_t *cpu = CPU;
6431
6432         if (dtrace_destructive_disallow)
6433                 return;
6434
6435         flags = (volatile uint16_t *)&cpu_core[cpu->cpu_id].cpuc_dtrace_flags;
6436
6437         now = dtrace_gethrtime();
6438
6439         if (now - cpu->cpu_dtrace_chillmark > dtrace_chill_interval) {
6440                 /*
6441                  * We need to advance the mark to the current time.
6442                  */
6443                 cpu->cpu_dtrace_chillmark = now;
6444                 cpu->cpu_dtrace_chilled = 0;
6445         }
6446
6447         /*
6448          * Now check to see if the requested chill time would take us over
6449          * the maximum amount of time allowed in the chill interval.  (Or
6450          * worse, if the calculation itself induces overflow.)
6451          */
6452         if (cpu->cpu_dtrace_chilled + val > dtrace_chill_max ||
6453             cpu->cpu_dtrace_chilled + val < cpu->cpu_dtrace_chilled) {
6454                 *flags |= CPU_DTRACE_ILLOP;
6455                 return;
6456         }
6457
6458         while (dtrace_gethrtime() - now < val)
6459                 continue;
6460
6461         /*
6462          * Normally, we assure that the value of the variable "timestamp" does
6463          * not change within an ECB.  The presence of chill() represents an
6464          * exception to this rule, however.
6465          */
6466         mstate->dtms_present &= ~DTRACE_MSTATE_TIMESTAMP;
6467         cpu->cpu_dtrace_chilled += val;
6468 }
6469
6470 static void
6471 dtrace_action_ustack(dtrace_mstate_t *mstate, dtrace_state_t *state,
6472     uint64_t *buf, uint64_t arg)
6473 {
6474         int nframes = DTRACE_USTACK_NFRAMES(arg);
6475         int strsize = DTRACE_USTACK_STRSIZE(arg);
6476         uint64_t *pcs = &buf[1], *fps;
6477         char *str = (char *)&pcs[nframes];
6478         int size, offs = 0, i, j;
```



```
6479         uintptr_t old = mstate->dtms_scratch_ptr, saved;
6480         uint16_t *flags = &cpu_core[CPU->cpu_id].cpuc_dtrace_flags;
6481         char *sym;
6482
6483         /*
6484          * Should be taking a faster path if string space has not been
6485          * allocated.
6486          */
6487         ASSERT(strsize != 0);
6488
6489         /*
6490          * We will first allocate some temporary space for the frame pointers.
6491          */
6492         fps = (uint64_t *)P2ROUNDUP(mstate->dtms_scratch_ptr, 8);
6493         size = (uintptr_t)fps - mstate->dtms_scratch_ptr +
6494             (nframes * sizeof (uint64_t));
6495
6496         if (!DTRACE_INSCRATCH(mstate, size)) {
6497                 /*
6498                  * Not enough room for our frame pointers -- need to indicate
6499                  * that we ran out of scratch space.
6500                  */
6501                 DTRACE_CPUFLAG_SET(CPU_DTRACE_NOSCRATCH);
6502                 return;
6503         }
6504
6505         mstate->dtms_scratch_ptr += size;
6506         saved = mstate->dtms_scratch_ptr;
6507
6508         /*
6509          * Now get a stack with both program counters and frame pointers.
6510          */
6511         DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
6512         dtrace_getufpstack(buf, fps, nframes + 1);
6513         DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
6514
6515         /*
6516          * If that faulted, we're cooked.
6517          */
6518         if (*flags & CPU_DTRACE_FAULT)
6519                 goto out;
6520
6521         /*
6522          * Now we want to walk up the stack, calling the USTACK helper.  For
6523          * each iteration, we restore the scratch pointer.
6524          */
6525         for (i = 0; i < nframes; i++) {
6526                 mstate->dtms_scratch_ptr = saved;
6527
6528                 if (offs >= strsize)
6529                         break;
6530
6531                 sym = (char *)(uintptr_t)dtrace_helper(
6532                     DTRACE_HELPER_ACTION_USTACK,
6533                     mstate, state, pcs[i], fps[i]);
6534
6535                 /*
6536                  * If we faulted while running the helper, we're going to
6537                  * clear the fault and null out the corresponding string.
6538                  */
6539                 if (*flags & CPU_DTRACE_FAULT) {
6540                         *flags &= ~CPU_DTRACE_FAULT;
6541                         str[offs++] = '\0';
6542                         continue;
6543                 }
```

```
6545                 if (sym == NULL) {
6546                         str[offs++] = '\0';
6547                         continue;
6548                 }

6550                 DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);

6552                 /*
6553                  * Now copy in the string that the helper returned to us.
6554                  */
6555                 for (j = 0; offs + j < strsize; j++) {
6556                         if ((str[offs + j] = sym[j]) == '\0')
6557                                 break;
6558                 }

6560                 DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);

6562                 offs += j + 1;
6563         }

6565         if (offs >= strsize) {
6566                 /*
6567                  * If we didn't have room for all of the strings, we don't
6568                  * abort processing -- this needn't be a fatal error -- but we
6569                  * still want to increment a counter (dts_stkstroverflows) to
6570                  * allow this condition to be warned about.  (If this is from
6571                  * a jstack() action, it is easily tuned via jstackstrsize.)
6572                  */
6573                 dtrace_error(&state->dts_stkstroverflows);
6574         }

6576         while (offs < strsize)
6577                 str[offs++] = '\0';

6579 out:
6580         mstate->dtms_scratch_ptr = old;
6581 }

6583 static void
6584 dtrace_store_by_ref(dtrace_difo_t *dp, caddr_t tomax, size_t size,
6585     size_t *valoffsp, uint64_t *valp, uint64_t end, int intuple, int dtkind)
6586 {
6587         volatile uint16_t *flags;
6588         uint64_t val = *valp;
6589         size_t valoffs = *valoffsp;

6591         flags = (volatile uint16_t *)&cpu_core[CPU->cpu_id].cpuc_dtrace_flags;
6592         ASSERT(dtkind == DIF_TF_BYREF || dtkind == DIF_TF_BYUREF);

6594         /*
6595          * If this is a string, we're going to only load until we find the zero
6596          * byte -- after which we'll store zero bytes.
6597          */
6598         if (dp->dtdo_rtype.dtdt_kind == DIF_TYPE_STRING) {
6599                 char c = '\0' + 1;
6600                 size_t s;

6602                 for (s = 0; s < size; s++) {
6603                         if (c != '\0' && dtkind == DIF_TF_BYREF) {
6604                                 c = dtrace_load8(val++);
6605                         } else if (c != '\0' && dtkind == DIF_TF_BYUREF) {
6606                                 DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
6607                                 c = dtrace_fuword8((void *)(uintptr_t)val++);
6608                                 DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
6609                                 if (*flags & CPU_DTRACE_FAULT)
6610                                         break;
```



```
6611                         }

6613                         DTRACE_STORE(uint8_t, tomax, valoffs++, c);

6615                         if (c == '\0' && intuple)
6616                                 break;
6617                 }
6618         } else {
6619                 uint8_t c;
6620                 while (valoffs < end) {
6621                         if (dtkind == DIF_TF_BYREF) {
6622                                 c = dtrace_load8(val++);
6623                         } else if (dtkind == DIF_TF_BYUREF) {
6624                                 DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
6625                                 c = dtrace_fuword8((void *)(uintptr_t)val++);
6626                                 DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
6627                                 if (*flags & CPU_DTRACE_FAULT)
6628                                         break;
6629                         }

6631                         DTRACE_STORE(uint8_t, tomax,
6632                             valoffs++, c);
6633                 }
6634         }

6636         *valp = val;
6637         *valoffsp = valoffs;
6638 }

6640 #endif /* ! codereview */
6641 /*
6642  * If you're looking for the epicenter of DTrace, you just found it.  This
6643  * is the function called by the provider to fire a probe -- from which all
6644  * subsequent probe-context DTrace activity emanates.
6645  */
6646 void
6647 dtrace_probe(dtrace_id_t id, uintptr_t arg0, uintptr_t arg1,
6648     uintptr_t arg2, uintptr_t arg3, uintptr_t arg4)
6649 {
6650         processorid_t cpuid;
6651         dtrace_icookie_t cookie;
6652         dtrace_probe_t *probe;
6653         dtrace_mstate_t mstate;
6654         dtrace_ecb_t *ecb;
6655         dtrace_action_t *act;
6656         intptr_t offs;
6657         size_t size;
6658         int vtime, onintr;
6659         volatile uint16_t *flags;
6660         hrtime_t now, end;

6662         /*
6663          * Kick out immediately if this CPU is still being born (in which case
6664          * curthread will be set to -1) or the current thread can't allow
6665          * probes in its current context.
6666          */
6667         if (((uintptr_t)curthread & 1) || (curthread->t_flag & T_DONTDTRACE))
6668                 return;

6670         cookie = dtrace_interrupt_disable();
6671         probe = dtrace_probes[id - 1];
6672         cpuid = CPU->cpu_id;
6673         onintr = CPU_ON_INTR(CPU);

6675         CPU->cpu_dtrace_probes++;
```

```
6677         if (!onintr && probe->dtpr_predcache != DTRACE_CACHEIDNONE &&
6678             probe->dtpr_predcache == curthread->t_predcache) {
6679                 /*
6680                  * We have hit in the predicate cache; we know that
6681                  * this predicate would evaluate to be false.
6682                  */
6683                 dtrace_interrupt_enable(cookie);
6684                 return;
6685         }
6687         if (panic_quiesce) {
6688                 /*
6689                  * We don't trace anything if we're panicking.
6690                  */
6691                 dtrace_interrupt_enable(cookie);
6692                 return;
6693         }
6695         now = dtrace_gethrtime();
6696         vtime = dtrace_vtime_references != 0;
6698         if (vtime && curthread->t_dtrace_start)
6699                 curthread->t_dtrace_vtime += now - curthread->t_dtrace_start;
6701         mstate.dtms_difo = NULL;
6702         mstate.dtms_probe = probe;
6703         mstate.dtms_strtok = NULL;
6704         mstate.dtms_arg[0] = arg0;
6705         mstate.dtms_arg[1] = arg1;
6706         mstate.dtms_arg[2] = arg2;
6707         mstate.dtms_arg[3] = arg3;
6708         mstate.dtms_arg[4] = arg4;
6710         flags = (volatile uint16_t *)&cpu_core[cpuid].cpuc_dtrace_flags;
6712         for (ecb = probe->dtpr_ecb; ecb != NULL; ecb = ecb->dte_next) {
6713                 dtrace_predicate_t *pred = ecb->dte_predicate;
6714                 dtrace_state_t *state = ecb->dte_state;
6715                 dtrace_buffer_t *buf = &state->dts_buffer[cpuid];
6716                 dtrace_buffer_t *aggbuf = &state->dts_aggbuffer[cpuid];
6717                 dtrace_vstate_t *vstate = &state->dts_vstate;
6718                 dtrace_provider_t *prov = probe->dtpr_provider;
6719                 uint64_t tracememsize = 0;
6720                 int committed = 0;
6721                 caddr_t tomax;
6723                 /*
6724                  * A little subtlety with the following (seemingly innocuous)
6725                  * declaration of the automatic 'val':  by looking at the
6726                  * code, you might think that it could be declared in the
6727                  * action processing loop, below.  (That is, it's only used in
6728                  * the action processing loop.)  However, it must be declared
6729                  * out of that scope because in the case of DIF expression
6730                  * arguments to aggregating actions, one iteration of the
6731                  * action loop will use the last iteration's value.
6732                  */
6733 #ifdef lint
6734                 uint64_t val = 0;
6735 #else
6736                 uint64_t val;
6737 #endif
6739                 mstate.dtms_present = DTRACE_MSTATE_ARGS | DTRACE_MSTATE_PROBE;
6740                 mstate.dtms_access = DTRACE_ACCESS_ARGS | DTRACE_ACCESS_PROC;
6741                 mstate.dtms_getf = NULL;
```



```
6743                 *flags &= ~CPU_DTRACE_ERROR;
6745                 if (prov == dtrace_provider) {
6746                         /*
6747                          * If dtrace itself is the provider of this probe,
6748                          * we're only going to continue processing the ECB if
6749                          * arg0 (the dtrace_state_t) is equal to the ECB's
6750                          * creating state.  (This prevents disjoint consumers
6751                          * from seeing one another's metaprobes.)
6752                          */
6753                         if (arg0 != (uint64_t)(uintptr_t)state)
6754                                 continue;
6755                 }
6757                 if (state->dts_activity != DTRACE_ACTIVITY_ACTIVE) {
6758                         /*
6759                          * We're not currently active.  If our provider isn't
6760                          * the dtrace pseudo provider, we're not interested.
6761                          */
6762                         if (prov != dtrace_provider)
6763                                 continue;
6765                         /*
6766                          * Now we must further check if we are in the BEGIN
6767                          * probe.  If we are, we will only continue processing
6768                          * if we're still in WARMUP -- if one BEGIN enabling
6769                          * has invoked the exit() action, we don't want to
6770                          * evaluate subsequent BEGIN enablings.
6771                          */
6772                         if (probe->dtpr_id == dtrace_probeid_begin &&
6773                             state->dts_activity != DTRACE_ACTIVITY_WARMUP) {
6774                                 ASSERT(state->dts_activity ==
6775                                     DTRACE_ACTIVITY_DRAINING);
6776                                 continue;
6777                         }
6778                 }
6780                 if (ecb->dte_cond && !dtrace_priv_probe(state, &mstate, ecb))
6781                         continue;
6783                 if (now - state->dts_alive > dtrace_deadman_timeout) {
6784                         /*
6785                          * We seem to be dead.  Unless we (a) have kernel
6786                          * destructive permissions (b) have explicitly enabled
6787                          * destructive actions and (c) destructive actions have
6788                          * not been disabled, we're going to transition into
6789                          * the KILLED state, from which no further processing
6790                          * on this state will be performed.
6791                          */
6792                         if (!dtrace_priv_kernel_destructive(state) ||
6793                             !state->dts_cred.dcr_destructive ||
6794                             dtrace_destructive_disallow) {
6795                                 void *activity = &state->dts_activity;
6796                                 dtrace_activity_t current;
6798                                 do {
6799                                         current = state->dts_activity;
6800                                 } while (dtrace_cas32(activity, current,
6801                                     DTRACE_ACTIVITY_KILLED) != current);
6803                                 continue;
6804                         }
6805                 }
6807                 if ((offs = dtrace_buffer_reserve(buf, ecb->dte_needed,
6808                     ecb->dte_alignment, state, &mstate)) < 0)
```

```
6809                            continue;
6811                    tomax = buf->dtb_tomax;
6812                    ASSERT(tomax != NULL);
6814                    if (ecb->dte_size != 0) {
6815                            dtrace_rechdr_t dtrh;
6816                            if (!(mstate.dtms_present & DTRACE_MSTATE_TIMESTAMP)) {
6817                                    mstate.dtms_timestamp = dtrace_gethrtime();
6818                                    mstate.dtms_present |= DTRACE_MSTATE_TIMESTAMP;
6819                            }
6820                            ASSERT3U(ecb->dte_size, >=, sizeof (dtrace_rechdr_t));
6821                            dtrh.dtrh_epid = ecb->dte_epid;
6822                            DTRACE_RECORD_STORE_TIMESTAMP(&dtrh,
6823                                mstate.dtms_timestamp);
6824                            *((dtrace_rechdr_t *)(tomax + offs)) = dtrh;
6825                    }
6827                    mstate.dtms_epid = ecb->dte_epid;
6828                    mstate.dtms_present |= DTRACE_MSTATE_EPID;
6830                    if (state->dts_cred.dcr_visible & DTRACE_CRV_KERNEL)
6831                            mstate.dtms_access |= DTRACE_ACCESS_KERNEL;
6833                    if (pred != NULL) {
6834                            dtrace_difo_t *dp = pred->dtp_difo;
6835                            int rval;
6837                            rval = dtrace_dif_emulate(dp, &mstate, vstate, state);
6839                            if (!(*flags & CPU_DTRACE_ERROR) && !rval) {
6840                                    dtrace_cacheid_t cid = probe->dtpr_predcache;
6842                                    if (cid != DTRACE_CACHEIDNONE && !onintr) {
6843                                            /*
6844                                             * Update the predicate cache...
6845                                             */
6846                                            ASSERT(cid == pred->dtp_cacheid);
6847                                            curthread->t_predcache = cid;
6848                                    }
6850                                    continue;
6851                            }
6852                    }
6854                    for (act = ecb->dte_action; !(*flags & CPU_DTRACE_ERROR) &&
6855                        act != NULL; act = act->dta_next) {
6856                            size_t valoffs;
6857                            dtrace_difo_t *dp;
6858                            dtrace_recdesc_t *rec = &act->dta_rec;
6860                            size = rec->dtrd_size;
6861                            valoffs = offs + rec->dtrd_offset;
6863                            if (DTRACEACT_ISAGG(act->dta_kind)) {
6864                                    uint64_t v = 0xbad;
6865                                    dtrace_aggregation_t *agg;
6867                                    agg = (dtrace_aggregation_t *)act;
6869                                    if ((dp = act->dta_difo) != NULL)
6870                                            v = dtrace_dif_emulate(dp,
6871                                                &mstate, vstate, state);
6873                                    if (*flags & CPU_DTRACE_ERROR)
6874                                            continue;
```



```
6876                                    /*
6877                                     * Note that we always pass the expression
6878                                     * value from the previous iteration of the
6879                                     * action loop.  This value will only be used
6880                                     * if there is an expression argument to the
6881                                     * aggregating action, denoted by the
6882                                     * dtag_hasarg field.
6883                                     */
6884                                    dtrace_aggregate(agg, buf,
6885                                        offs, aggbuf, v, val);
6886                                    continue;
6887                            }
6889                            switch (act->dta_kind) {
6890                            case DTRACEACT_STOP:
6891                                    if (dtrace_priv_proc_destructive(state,
6892                                        &mstate))
6893                                            dtrace_action_stop();
6894                                    continue;
6896                            case DTRACEACT_BREAKPOINT:
6897                                    if (dtrace_priv_kernel_destructive(state))
6898                                            dtrace_action_breakpoint(ecb);
6899                                    continue;
6901                            case DTRACEACT_PANIC:
6902                                    if (dtrace_priv_kernel_destructive(state))
6903                                            dtrace_action_panic(ecb);
6904                                    continue;
6906                            case DTRACEACT_STACK:
6907                                    if (!dtrace_priv_kernel(state))
6908                                            continue;
6910                                    dtrace_getpcstack((pc_t *)(tomax + valoffs),
6911                                        size / sizeof (pc_t), probe->dtpr_aframes,
6912                                        DTRACE_ANCHORED(probe) ? NULL :
6913                                        (uint32_t *)arg0);
6915                                    continue;
6917                            case DTRACEACT_JSTACK:
6918                            case DTRACEACT_USTACK:
6919                                    if (!dtrace_priv_proc(state, &mstate))
6920                                            continue;
6922                                    /*
6923                                     * See comment in DIF_VAR_PID.
6924                                     */
6925                                    if (DTRACE_ANCHORED(mstate.dtms_probe) &&
6926                                        CPU_ON_INTR(CPU)) {
6927                                            int depth = DTRACE_USTACK_NFRAMES(
6928                                                rec->dtrd_arg) + 1;
6930                                            dtrace_bzero((void *)(tomax + valoffs),
6931                                                DTRACE_USTACK_STRSIZE(rec->dtrd_arg)
6932                                                + depth * sizeof (uint64_t));
6934                                            continue;
6935                                    }
6937                                    if (DTRACE_USTACK_STRSIZE(rec->dtrd_arg) != 0 &&
6938                                        curproc->p_dtrace_helpers != NULL) {
6939                                            /*
6940                                             * This is the slow path -- we have
```

```
6941                                     * allocated string space, and we're
6942                                     * getting the stack of a process that
6943                                     * has helpers.  Call into a separate
6944                                     * routine to perform this processing.
6945                                     */
6946                                    dtrace_action_ustack(&mstate, state,
6947                                        (uint64_t *)(tomax + valoffs),
6948                                        rec->dtrd_arg);
6949                                    continue;
6950                            }

6952                            /*
6953                             * Clear the string space, since there's no
6954                             * helper to do it for us.
6955                             */
6956                            if (DTRACE_USTACK_STRSIZE(rec->dtrd_arg) != 0) {
6957                                    int depth = DTRACE_USTACK_NFRAMES(
6958                                        rec->dtrd_arg);
6959                                    size_t strsize = DTRACE_USTACK_STRSIZE(
6960                                        rec->dtrd_arg);
6961                                    uint64_t *buf = (uint64_t *)(tomax +
6962                                        valoffs);
6963                                    void *strspace = &buf[depth + 1];

6965                                    dtrace_bzero(strspace,
6966                                        MIN(depth, strsize));
6967                            }

6969                            DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
6970                            dtrace_getupcstack((uint64_t *)
6971                                (tomax + valoffs),
6972                                DTRACE_USTACK_NFRAMES(rec->dtrd_arg) + 1);
6973                            DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
6974                            continue;

6976                    default:
6977                            break;
6978                    }

6980                    dp = act->dta_difo;
6981                    ASSERT(dp != NULL);

6983                    val = dtrace_dif_emulate(dp, &mstate, vstate, state);

6985                    if (*flags & CPU_DTRACE_ERROR)
6986                            continue;

6988                    switch (act->dta_kind) {
6989                    case DTRACEACT_SPECULATE: {
6990                            dtrace_rechdr_t *dtrh;

6992                            ASSERT(buf == &state->dts_buffer[cpuid]);
6993                            buf = dtrace_speculation_buffer(state,
6994                                cpuid, val);

6996                            if (buf == NULL) {
6997                                    *flags |= CPU_DTRACE_DROP;
6998                                    continue;
6999                            }

7001                            offs = dtrace_buffer_reserve(buf,
7002                                ecb->dte_needed, ecb->dte_alignment,
7003                                state, NULL);

7005                            if (offs < 0) {
7006                                    *flags |= CPU_DTRACE_DROP;
```



```
7007                                            continue;
7008                                    }

7010                                    tomax = buf->dtb_tomax;
7011                                    ASSERT(tomax != NULL);

7013                                    if (ecb->dte_size == 0)
7014                                            continue;

7016                                    ASSERT3U(ecb->dte_size, >=,
7017                                        sizeof (dtrace_rechdr_t));
7018                                    dtrh = ((void *)(tomax + offs));
7019                                    dtrh->dtrh_epid = ecb->dte_epid;
7020                                    /*
7021                                     * When the speculation is committed, all of
7022                                     * the records in the speculative buffer will
7023                                     * have their timestamps set to the commit
7024                                     * time.  Until then, it is set to a sentinel
7025                                     * value, for debugability.
7026                                     */
7027                                    DTRACE_RECORD_STORE_TIMESTAMP(dtrh, UINT64_MAX);
7028                                    continue;
7029                            }

7031                    case DTRACEACT_CHILL:
7032                            if (dtrace_priv_kernel_destructive(state))
7033                                    dtrace_action_chill(&mstate, val);
7034                            continue;

7036                    case DTRACEACT_RAISE:
7037                            if (dtrace_priv_proc_destructive(state,
7038                                &mstate))
7039                                    dtrace_action_raise(val);
7040                            continue;

7042                    case DTRACEACT_COMMIT:
7043                            ASSERT(!committed);

7045                            /*
7046                             * We need to commit our buffer state.
7047                             */
7048                            if (ecb->dte_size)
7049                                    buf->dtb_offset = offs + ecb->dte_size;
7050                            buf = &state->dts_buffer[cpuid];
7051                            dtrace_speculation_commit(state, cpuid, val);
7052                            committed = 1;
7053                            continue;

7055                    case DTRACEACT_DISCARD:
7056                            dtrace_speculation_discard(state, cpuid, val);
7057                            continue;

7059                    case DTRACEACT_DIFEXPR:
7060                    case DTRACEACT_LIBACT:
7061                    case DTRACEACT_PRINTF:
7062                    case DTRACEACT_PRINTA:
7063                    case DTRACEACT_SYSTEM:
7064                    case DTRACEACT_FREOPEN:
7065                    case DTRACEACT_TRACEMEM:
7066                            break;

7068                    case DTRACEACT_TRACEMEM_DYNSIZE:
7069                            tracememsize = val;
7070                            break;

7072                    case DTRACEACT_SYM:
```

```
7073                        case DTRACEACT_MOD:
7074                                if (!dtrace_priv_kernel(state))
7075                                        continue;
7076                                break;
7077
7078                        case DTRACEACT_USYM:
7079                        case DTRACEACT_UMOD:
7080                        case DTRACEACT_UADDR: {
7081                                struct pid *pid = curthread->t_procp->p_pidp;
7082
7083                                if (!dtrace_priv_proc(state, &mstate))
7084                                        continue;
7085
7086                                DTRACE_STORE(uint64_t, tomax,
7087                                    valoffs, (uint64_t)pid->pid_id);
7088                                DTRACE_STORE(uint64_t, tomax,
7089                                    valoffs + sizeof (uint64_t), val);
7090
7091                                continue;
7092                        }
7093
7094                        case DTRACEACT_EXIT: {
7095                                /*
7096                                 * For the exit action, we are going to attempt
7097                                 * to atomically set our activity to be
7098                                 * draining.  If this fails (either because
7099                                 * another CPU has beat us to the exit action,
7100                                 * or because our current activity is something
7101                                 * other than ACTIVE or WARMUP), we will
7102                                 * continue.  This assures that the exit action
7103                                 * can be successfully recorded at most once
7104                                 * when we're in the ACTIVE state.  If we're
7105                                 * encountering the exit() action while in
7106                                 * COOLDOWN, however, we want to honor the new
7107                                 * status code.  (We know that we're the only
7108                                 * thread in COOLDOWN, so there is no race.)
7109                                 */
7110                                void *activity = &state->dts_activity;
7111                                dtrace_activity_t current = state->dts_activity;
7112
7113                                if (current == DTRACE_ACTIVITY_COOLDOWN)
7114                                        break;
7115
7116                                if (current != DTRACE_ACTIVITY_WARMUP)
7117                                        current = DTRACE_ACTIVITY_ACTIVE;
7118
7119                                if (dtrace_cas32(activity, current,
7120                                    DTRACE_ACTIVITY_DRAINING) != current) {
7121                                        *flags |= CPU_DTRACE_DROP;
7122                                        continue;
7123                                }
7124
7125                                break;
7126                        }
7127
7128                        default:
7129                                ASSERT(0);
7130                        }
7131
7132                        if (dp->dtdo_rtype.dtdt_flags & DIF_TF_BYREF ||
7133                            dp->dtdo_rtype.dtdt_flags & DIF_TF_BYUREF) {
5722                        if (dp->dtdo_rtype.dtdt_flags & DIF_TF_BYREF) {
7134                                uintptr_t end = valoffs + size;
7135
7136                                if (tracememsize != 0 &&
7137                                    valoffs + tracememsize < end) {
```



```
7138                                        end = valoffs + tracememsize;
7139                                        tracememsize = 0;
7140                                }
7141
7142                                if (dp->dtdo_rtype.dtdt_flags & DIF_TF_BYREF &&
7143                                    !dtrace_vcanload((void *)(uintptr_t)val,
5731                                if (!dtrace_vcanload((void *)(uintptr_t)val,
7144                                    &dp->dtdo_rtype, &mstate, vstate))
7145                                        continue;
7146
7147                                dtrace_store_by_ref(dp, tomax, size, &valoffs,
7148                                    &val, end, act->dta_intuple,
7149                                    dp->dtdo_rtype.dtdt_flags & DIF_TF_BYREF ?
7150                                    DIF_TF_BYREF: DIF_TF_BYUREF);
5735                                /*
5736                                 * If this is a string, we're going to only
5737                                 * load until we find the zero byte -- after
5738                                 * which we'll store zero bytes.
5739                                 */
5740                                if (dp->dtdo_rtype.dtdt_kind ==
5741                                    DIF_TYPE_STRING) {
5742                                        char c = '\0' + 1;
5743                                        int intuple = act->dta_intuple;
5744                                        size_t s;
5745
5746                                        for (s = 0; s < size; s++) {
5747                                                if (c != '\0')
5748                                                        c = dtrace_load8(val++);
5749
5750                                                DTRACE_STORE(uint8_t, tomax,
5751                                                    valoffs++, c);
5752
5753                                                if (c == '\0' && intuple)
5754                                                        break;
5755                                        }
5756
5757                                        continue;
5758                                }
5759
5760                                while (valoffs < end) {
5761                                        DTRACE_STORE(uint8_t, tomax, valoffs++,
5762                                            dtrace_load8(val++));
5763                                }
5764
7151                                continue;
7152                        }
7153
7154                        switch (size) {
7155                        case 0:
7156                                break;
7157
7158                        case sizeof (uint8_t):
7159                                DTRACE_STORE(uint8_t, tomax, valoffs, val);
7160                                break;
7161                        case sizeof (uint16_t):
7162                                DTRACE_STORE(uint16_t, tomax, valoffs, val);
7163                                break;
7164                        case sizeof (uint32_t):
7165                                DTRACE_STORE(uint32_t, tomax, valoffs, val);
7166                                break;
7167                        case sizeof (uint64_t):
7168                                DTRACE_STORE(uint64_t, tomax, valoffs, val);
7169                                break;
7170                        default:
7171                                /*
7172                                 * Any other size should have been returned by
```

```
7173                                     * reference, not by value.
7174                                     */
7175                                    ASSERT(0);
7176                                    break;
7177                            }
7178                    }

7180                    if (*flags & CPU_DTRACE_DROP)
7181                            continue;

7183                    if (*flags & CPU_DTRACE_FAULT) {
7184                            int ndx;
7185                            dtrace_action_t *err;

7187                            buf->dtb_errors++;

7189                            if (probe->dtpr_id == dtrace_probeid_error) {
7190                                    /*
7191                                     * There's nothing we can do -- we had an
7192                                     * error on the error probe.  We bump an
7193                                     * error counter to at least indicate that
7194                                     * this condition happened.
7195                                     */
7196                                    dtrace_error(&state->dts_dblerrors);
7197                                    continue;
7198                            }

7200                            if (vtime) {
7201                                    /*
7202                                     * Before recursing on dtrace_probe(), we
7203                                     * need to explicitly clear out our start
7204                                     * time to prevent it from being accumulated
7205                                     * into t_dtrace_vtime.
7206                                     */
7207                                    curthread->t_dtrace_start = 0;
7208                            }

7210                            /*
7211                             * Iterate over the actions to figure out which action
7212                             * we were processing when we experienced the error.
7213                             * Note that act points _past_ the faulting action; if
7214                             * act is ecb->dte_action, the fault was in the
7215                             * predicate, if it's ecb->dte_action->dta_next it's
7216                             * in action #1, and so on.
7217                             */
7218                            for (err = ecb->dte_action, ndx = 0;
7219                                err != act; err = err->dta_next, ndx++)
7220                                    continue;

7222                            dtrace_probe_error(state, ecb->dte_epid, ndx,
7223                                (mstate.dtms_present & DTRACE_MSTATE_FLTOFFS) ?
7224                                mstate.dtms_fltoffs : -1, DTRACE_FLAGS2FLT(*flags),
7225                                cpu_core[cpuid].cpuc_dtrace_illval);

7227                            continue;
7228                    }

7230                    if (!committed)
7231                            buf->dtb_offset = offs + ecb->dte_size;
7232            }

7234            end = dtrace_gethrtime();
7235            if (vtime)
7236                    curthread->t_dtrace_start = end;

7238            CPU->cpu_dtrace_nsec += end - now;
```



```
7240            dtrace_interrupt_enable(cookie);
7241    }
______unchanged_portion_omitted_

8975    /*
8976     * Validate a DTrace DIF object by checking the IR instructions.  The following
8977     * rules are currently enforced by dtrace_difo_validate():
8978     *
8979     * 1. Each instruction must have a valid opcode
8980     * 2. Each register, string, variable, or subroutine reference must be valid
8981     * 3. No instruction can modify register %r0 (must be zero)
8982     * 4. All instruction reserved bits must be set to zero
8983     * 5. The last instruction must be a "ret" instruction
8984     * 6. All branch targets must reference a valid instruction _after_ the branch
8985     */
8986    static int
8987    dtrace_difo_validate(dtrace_difo_t *dp, dtrace_vstate_t *vstate, uint_t nregs,
8988        cred_t *cr)
8989    {
8990            int err = 0, i;
8991            int (*efunc)(uint_t pc, const char *, ...) = dtrace_difo_err;
8992            int kcheckload;
8993            uint_t pc;

8995            kcheckload = cr == NULL ||
8996                (vstate->dtvs_state->dts_cred.dcr_visible & DTRACE_CRV_KERNEL) == 0;

8998            dp->dtdo_destructive = 0;

9000            for (pc = 0; pc < dp->dtdo_len && err == 0; pc++) {
9001                    dif_instr_t instr = dp->dtdo_buf[pc];

9003                    uint_t r1 = DIF_INSTR_R1(instr);
9004                    uint_t r2 = DIF_INSTR_R2(instr);
9005                    uint_t rd = DIF_INSTR_RD(instr);
9006                    uint_t rs = DIF_INSTR_RS(instr);
9007                    uint_t label = DIF_INSTR_LABEL(instr);
9008                    uint_t v = DIF_INSTR_VAR(instr);
9009                    uint_t subr = DIF_INSTR_SUBR(instr);
9010                    uint_t type = DIF_INSTR_TYPE(instr);
9011                    uint_t op = DIF_INSTR_OP(instr);

9013                    switch (op) {
9014                    case DIF_OP_OR:
9015                    case DIF_OP_XOR:
9016                    case DIF_OP_AND:
9017                    case DIF_OP_SLL:
9018                    case DIF_OP_SRL:
9019                    case DIF_OP_SRA:
9020                    case DIF_OP_SUB:
9021                    case DIF_OP_ADD:
9022                    case DIF_OP_MUL:
9023                    case DIF_OP_SDIV:
9024                    case DIF_OP_UDIV:
9025                    case DIF_OP_SREM:
9026                    case DIF_OP_UREM:
9027                    case DIF_OP_COPYS:
9028                            if (r1 >= nregs)
9029                                    err += efunc(pc, "invalid register %u\n", r1);
9030                            if (r2 >= nregs)
9031                                    err += efunc(pc, "invalid register %u\n", r2);
9032                            if (rd >= nregs)
9033                                    err += efunc(pc, "invalid register %u\n", rd);
9034                            if (rd == 0)
9035                                    err += efunc(pc, "cannot write to %r0\n");
```

```
9036                    break;
9037            case DIF_OP_NOT:
9038            case DIF_OP_MOV:
9039            case DIF_OP_ALLOCS:
9040                    if (r1 >= nregs)
9041                            err += efunc(pc, "invalid register %u\n", r1);
9042                    if (r2 != 0)
9043                            err += efunc(pc, "non-zero reserved bits\n");
9044                    if (rd >= nregs)
9045                            err += efunc(pc, "invalid register %u\n", rd);
9046                    if (rd == 0)
9047                            err += efunc(pc, "cannot write to %r0\n");
9048                    break;
9049            case DIF_OP_LDSB:
9050            case DIF_OP_LDSH:
9051            case DIF_OP_LDSW:
9052            case DIF_OP_LDUB:
9053            case DIF_OP_LDUH:
9054            case DIF_OP_LDUW:
9055            case DIF_OP_LDX:
9056                    if (r1 >= nregs)
9057                            err += efunc(pc, "invalid register %u\n", r1);
9058                    if (r2 != 0)
9059                            err += efunc(pc, "non-zero reserved bits\n");
9060                    if (rd >= nregs)
9061                            err += efunc(pc, "invalid register %u\n", rd);
9062                    if (rd == 0)
9063                            err += efunc(pc, "cannot write to %r0\n");
9064                    if (kcheckload)
9065                            dp->dtdo_buf[pc] = DIF_INSTR_LOAD(op +
9066                                DIF_OP_RLDSB - DIF_OP_LDSB, r1, rd);
9067                    break;
9068            case DIF_OP_RLDSB:
9069            case DIF_OP_RLDSH:
9070            case DIF_OP_RLDSW:
9071            case DIF_OP_RLDUB:
9072            case DIF_OP_RLDUH:
9073            case DIF_OP_RLDUW:
9074            case DIF_OP_RLDX:
9075                    if (r1 >= nregs)
9076                            err += efunc(pc, "invalid register %u\n", r1);
9077                    if (r2 != 0)
9078                            err += efunc(pc, "non-zero reserved bits\n");
9079                    if (rd >= nregs)
9080                            err += efunc(pc, "invalid register %u\n", rd);
9081                    if (rd == 0)
9082                            err += efunc(pc, "cannot write to %r0\n");
9083                    break;
9084            case DIF_OP_ULDSB:
9085            case DIF_OP_ULDSH:
9086            case DIF_OP_ULDSW:
9087            case DIF_OP_ULDUB:
9088            case DIF_OP_ULDUH:
9089            case DIF_OP_ULDUW:
9090            case DIF_OP_ULDX:
9091                    if (r1 >= nregs)
9092                            err += efunc(pc, "invalid register %u\n", r1);
9093                    if (r2 != 0)
9094                            err += efunc(pc, "non-zero reserved bits\n");
9095                    if (rd >= nregs)
9096                            err += efunc(pc, "invalid register %u\n", rd);
9097                    if (rd == 0)
9098                            err += efunc(pc, "cannot write to %r0\n");
9099                    break;
9100            case DIF_OP_STB:
9101            case DIF_OP_STH:
```



```
9102            case DIF_OP_STW:
9103            case DIF_OP_STX:
9104                    if (r1 >= nregs)
9105                            err += efunc(pc, "invalid register %u\n", r1);
9106                    if (r2 != 0)
9107                            err += efunc(pc, "non-zero reserved bits\n");
9108                    if (rd >= nregs)
9109                            err += efunc(pc, "invalid register %u\n", rd);
9110                    if (rd == 0)
9111                            err += efunc(pc, "cannot write to 0 address\n");
9112                    break;
9113            case DIF_OP_CMP:
9114            case DIF_OP_SCMP:
9115                    if (r1 >= nregs)
9116                            err += efunc(pc, "invalid register %u\n", r1);
9117                    if (r2 >= nregs)
9118                            err += efunc(pc, "invalid register %u\n", r2);
9119                    if (rd != 0)
9120                            err += efunc(pc, "non-zero reserved bits\n");
9121                    break;
9122            case DIF_OP_TST:
9123                    if (r1 >= nregs)
9124                            err += efunc(pc, "invalid register %u\n", r1);
9125                    if (r2 != 0 || rd != 0)
9126                            err += efunc(pc, "non-zero reserved bits\n");
9127                    break;
9128            case DIF_OP_BA:
9129            case DIF_OP_BE:
9130            case DIF_OP_BNE:
9131            case DIF_OP_BG:
9132            case DIF_OP_BGU:
9133            case DIF_OP_BGE:
9134            case DIF_OP_BGEU:
9135            case DIF_OP_BL:
9136            case DIF_OP_BLU:
9137            case DIF_OP_BLE:
9138            case DIF_OP_BLEU:
9139                    if (label >= dp->dtdo_len) {
9140                            err += efunc(pc, "invalid branch target %u\n",
9141                                label);
9142                    }
9143                    if (label <= pc) {
9144                            err += efunc(pc, "backward branch to %u\n",
9145                                label);
9146                    }
9147                    break;
9148            case DIF_OP_RET:
9149                    if (r1 != 0 || r2 != 0)
9150                            err += efunc(pc, "non-zero reserved bits\n");
9151                    if (rd >= nregs)
9152                            err += efunc(pc, "invalid register %u\n", rd);
9153                    break;
9154            case DIF_OP_NOP:
9155            case DIF_OP_POPTS:
9156            case DIF_OP_FLUSHTS:
9157                    if (r1 != 0 || r2 != 0 || rd != 0)
9158                            err += efunc(pc, "non-zero reserved bits\n");
9159                    break;
9160            case DIF_OP_SETX:
9161                    if (DIF_INSTR_INTEGER(instr) >= dp->dtdo_intlen) {
9162                            err += efunc(pc, "invalid integer ref %u\n",
9163                                DIF_INSTR_INTEGER(instr));
9164                    }
9165                    if (rd >= nregs)
9166                            err += efunc(pc, "invalid register %u\n", rd);
9167                    if (rd == 0)
```

```
9168                                    err += efunc(pc, "cannot write to %r0\n");
9169                            break;
9170                    case DIF_OP_SETS:
9171                            if (DIF_INSTR_STRING(instr) >= dp->dtdo_strlen) {
9172                                    err += efunc(pc, "invalid string ref %u\n",
9173                                        DIF_INSTR_STRING(instr));
9174                            }
9175                            if (rd >= nregs)
9176                                    err += efunc(pc, "invalid register %u\n", rd);
9177                            if (rd == 0)
9178                                    err += efunc(pc, "cannot write to %r0\n");
9179                            break;
9180                    case DIF_OP_LDGA:
9181                    case DIF_OP_LDTA:
9182                            if (r1 > DIF_VAR_ARRAY_MAX)
9183                                    err += efunc(pc, "invalid array %u\n", r1);
9184                            if (r2 >= nregs)
9185                                    err += efunc(pc, "invalid register %u\n", r2);
9186                            if (rd >= nregs)
9187                                    err += efunc(pc, "invalid register %u\n", rd);
9188                            if (rd == 0)
9189                                    err += efunc(pc, "cannot write to %r0\n");
9190                            break;
9191                    case DIF_OP_LDGS:
9192                    case DIF_OP_LDTS:
9193                    case DIF_OP_LDLS:
9194                    case DIF_OP_LDGAA:
9195                    case DIF_OP_LDTAA:
9196                            if (v < DIF_VAR_OTHER_MIN || v > DIF_VAR_OTHER_MAX)
9197                                    err += efunc(pc, "invalid variable %u\n", v);
9198                            if (rd >= nregs)
9199                                    err += efunc(pc, "invalid register %u\n", rd);
9200                            if (rd == 0)
9201                                    err += efunc(pc, "cannot write to %r0\n");
9202                            break;
9203                    case DIF_OP_STGS:
9204                    case DIF_OP_STTS:
9205                    case DIF_OP_STLS:
9206                    case DIF_OP_STGAA:
9207                    case DIF_OP_STTAA:
9208                            if (v < DIF_VAR_OTHER_UBASE || v > DIF_VAR_OTHER_MAX)
9209                                    err += efunc(pc, "invalid variable %u\n", v);
9210                            if (rs >= nregs)
9211                                    err += efunc(pc, "invalid register %u\n", rd);
9212                            break;
9213                    case DIF_OP_CALL:
9214                            if (subr > DIF_SUBR_MAX)
9215                                    err += efunc(pc, "invalid subr %u\n", subr);
9216                            if (rd >= nregs)
9217                                    err += efunc(pc, "invalid register %u\n", rd);
9218                            if (rd == 0)
9219                                    err += efunc(pc, "cannot write to %r0\n");

9221                            if (subr == DIF_SUBR_COPYOUT ||
9222                                subr == DIF_SUBR_COPYOUTSTR) {
9223                                    dp->dtdo_destructive = 1;
9224                            }

9226                            if (subr == DIF_SUBR_GETF) {
9227                                    /*
9228                                     * If we have a getf() we need to record that
9229                                     * in our state.  Note that our state can be
9230                                     * NULL if this is a helper -- but in that
9231                                     * case, the call to getf() is itself illegal,
9232                                     * and will be caught (slightly later) when
9233                                     * the helper is validated.
```



```
9234                                     */
9235                                    if (vstate->dtvs_state != NULL)
9236                                            vstate->dtvs_state->dts_getf++;
9237                            }

9239                            break;
9240                    case DIF_OP_PUSHTR:
9241                            if (type != DIF_TYPE_STRING && type != DIF_TYPE_CTF)
9242                                    err += efunc(pc, "invalid ref type %u\n", type);
9243                            if (r2 >= nregs)
9244                                    err += efunc(pc, "invalid register %u\n", r2);
9245                            if (rs >= nregs)
9246                                    err += efunc(pc, "invalid register %u\n", rs);
9247                            break;
9248                    case DIF_OP_PUSHTV:
9249                            if (type != DIF_TYPE_CTF)
9250                                    err += efunc(pc, "invalid val type %u\n", type);
9251                            if (r2 >= nregs)
9252                                    err += efunc(pc, "invalid register %u\n", r2);
9253                            if (rs >= nregs)
9254                                    err += efunc(pc, "invalid register %u\n", rs);
9255                            break;
9256                    default:
9257                            err += efunc(pc, "invalid opcode %u\n",
9258                                DIF_INSTR_OP(instr));
9259                    }
9260            }

9262            if (dp->dtdo_len != 0 &&
9263                DIF_INSTR_OP(dp->dtdo_buf[dp->dtdo_len - 1]) != DIF_OP_RET) {
9264                    err += efunc(dp->dtdo_len - 1,
9265                        "expected 'ret' as last DIF instruction\n");
9266            }

9268            if (!(dp->dtdo_rtype.dtdt_flags & (DIF_TF_BYREF | DIF_TF_BYUREF))) {
7882            if (!(dp->dtdo_rtype.dtdt_flags & DIF_TF_BYREF)) {
9269                    /*
9270                     * If we're not returning by reference, the size must be either
9271                     * 0 or the size of one of the base types.
9272                     */
9273                    switch (dp->dtdo_rtype.dtdt_size) {
9274                    case 0:
9275                    case sizeof (uint8_t):
9276                    case sizeof (uint16_t):
9277                    case sizeof (uint32_t):
9278                    case sizeof (uint64_t):
9279                            break;

9281                    default:
9282                            err += efunc(dp->dtdo_len - 1, "bad return size\n");
9283                    }
9284            }

9286            for (i = 0; i < dp->dtdo_varlen && err == 0; i++) {
9287                    dtrace_difv_t *v = &dp->dtdo_vartab[i], *existing = NULL;
9288                    dtrace_diftype_t *vt, *et;
9289                    uint_t id, ndx;

9291                    if (v->dtdv_scope != DIFV_SCOPE_GLOBAL &&
9292                        v->dtdv_scope != DIFV_SCOPE_THREAD &&
9293                        v->dtdv_scope != DIFV_SCOPE_LOCAL) {
9294                            err += efunc(i, "unrecognized variable scope %d\n",
9295                                v->dtdv_scope);
9296                            break;
9297                    }
```

```
9299                if (v->dtdv_kind != DIFV_KIND_ARRAY &&
9300                    v->dtdv_kind != DIFV_KIND_SCALAR) {
9301                        err += efunc(i, "unrecognized variable type %d\n",
9302                            v->dtdv_kind);
9303                        break;
9304                }

9306                if ((id = v->dtdv_id) > DIF_VARIABLE_MAX) {
9307                        err += efunc(i, "%d exceeds variable id limit\n", id);
9308                        break;
9309                }

9311                if (id < DIF_VAR_OTHER_UBASE)
9312                        continue;

9314                /*
9315                 * For user-defined variables, we need to check that this
9316                 * definition is identical to any previous definition that we
9317                 * encountered.
9318                 */
9319                ndx = id - DIF_VAR_OTHER_UBASE;

9321                switch (v->dtdv_scope) {
9322                case DIFV_SCOPE_GLOBAL:
9323                        if (ndx < vstate->dtvs_nglobals) {
9324                                dtrace_statvar_t *svar;

9326                                if ((svar = vstate->dtvs_globals[ndx]) != NULL)
9327                                        existing = &svar->dtsv_var;
9328                        }

9330                        break;

9332                case DIFV_SCOPE_THREAD:
9333                        if (ndx < vstate->dtvs_ntlocals)
9334                                existing = &vstate->dtvs_tlocals[ndx];
9335                        break;

9337                case DIFV_SCOPE_LOCAL:
9338                        if (ndx < vstate->dtvs_nlocals) {
9339                                dtrace_statvar_t *svar;

9341                                if ((svar = vstate->dtvs_locals[ndx]) != NULL)
9342                                        existing = &svar->dtsv_var;
9343                        }

9345                        break;
9346                }

9348                vt = &v->dtdv_type;

9350                if (vt->dtdt_flags & DIF_TF_BYREF) {
9351                        if (vt->dtdt_size == 0) {
9352                                err += efunc(i, "zero-sized variable\n");
9353                                break;
9354                        }

9356                        if (v->dtdv_scope == DIFV_SCOPE_GLOBAL &&
9357                            vt->dtdt_size > dtrace_global_maxsize) {
9358                                err += efunc(i, "oversized by-ref global\n");
9359                                break;
9360                        }
9361                }

9363                if (existing == NULL || existing->dtdv_id == 0)
9364                        continue;
```



```
9366                ASSERT(existing->dtdv_id == v->dtdv_id);
9367                ASSERT(existing->dtdv_scope == v->dtdv_scope);

9369                if (existing->dtdv_kind != v->dtdv_kind)
9370                        err += efunc(i, "%d changed variable kind\n", id);

9372                et = &existing->dtdv_type;

9374                if (vt->dtdt_flags != et->dtdt_flags) {
9375                        err += efunc(i, "%d changed variable type flags\n", id);
9376                        break;
9377                }

9379                if (vt->dtdt_size != 0 && vt->dtdt_size != et->dtdt_size) {
9380                        err += efunc(i, "%d changed variable type size\n", id);
9381                        break;
9382                }
9383        }

9385        return (err);
9386 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    9576 Tue Jan 14 16:49:04 2014
new/usr/src/uts/common/sys/ctf_api.h
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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, Version 1.0 only
   6  * (the "License").  You may not use this file except in compliance
   7  * with the License.
   8  *
   9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10  * or http://www.opensolaris.org/os/licensing.
  11  * See the License for the specific language governing permissions
  12  * and limitations under the License.
  13  *
  14  * When distributing Covered Code, include this CDDL HEADER in each
  15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16  * If applicable, add the following below this CDDL HEADER, with the
  17  * fields enclosed by brackets "[]" replaced with your own identifying
  18  * information: Portions Copyright [yyyy] [name of copyright owner]
  19  *
  20  * CDDL HEADER END
  21  */
  22 /*
  23  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */
  26 /*
  27  * Copyright (c) 2013, Joyent, Inc.  All rights reserved.
  27  * Copyright (c) 2012, Joyent, Inc.  All rights reserved.
  28  */

  30 /*
  31  * This header file defines the interfaces available from the CTF debugger
  32  * library, libctf, and an equivalent kernel module.  This API can be used by
  33  * a debugger to operate on data in the Compact ANSI-C Type Format (CTF).
  34  * This is NOT a public interface, although it may eventually become one in
  35  * the fullness of time after we gain more experience with the interfaces.
  36  *
  37  * In the meantime, be aware that any program linked with this API in this
  38  * release of Solaris is almost guaranteed to break in the next release.
  39  *
  40  * In short, do not user this header file or the CTF routines for any purpose.
  41  */

  43 #ifndef _CTF_API_H
  44 #define _CTF_API_H

  46 #include <sys/types.h>
  47 #include <sys/param.h>
  48 #include <sys/elf.h>
  49 #include <sys/ctf.h>

  51 #ifdef  __cplusplus
  52 extern "C" {
  53 #endif

  55 /*
```



```
  56  * Clients can open one or more CTF containers and obtain a pointer to an
  57  * opaque ctf_file_t.  Types are identified by an opaque ctf_id_t token.
  58  * These opaque definitions allow libctf to evolve without breaking clients.
  59  */
  60 typedef struct ctf_file ctf_file_t;
  61 typedef long ctf_id_t;

  63 /*
  64  * If the debugger needs to provide the CTF library with a set of raw buffers
  65  * for use as the CTF data, symbol table, and string table, it can do so by
  66  * filling in ctf_sect_t structures and passing them to ctf_bufopen():
  67  */
  68 typedef struct ctf_sect {
  69         const char *cts_name;   /* section name (if any) */
  70         ulong_t cts_type;       /* section type (ELF SHT_... value) */
  71         ulong_t cts_flags;      /* section flags (ELF SHF_... value) */
  72         const void *cts_data;   /* pointer to section data */
  73         size_t cts_size;        /* size of data in bytes */
  74         size_t cts_entsize;     /* size of each section entry (symtab only) */
  75         off64_t cts_offset;     /* file offset of this section (if any) */
  76 } ctf_sect_t;
______unchanged_portion_omitted_

 110 #define CTF_FUNC_VARARG 0x1     /* function arguments end with varargs */

 112 /*
 113  * Functions that return integer status or a ctf_id_t use the following value
 114  * to indicate failure.  ctf_errno() can be used to obtain an error code.
 115  */
 116 #define CTF_ERR (-1L)

 118 /*
 119  * The CTF data model is inferred to be the caller's data model or the data
 120  * model of the given object, unless ctf_setmodel() is explicitly called.
 121  */
 122 #define CTF_MODEL_ILP32 1       /* object data model is ILP32 */
 123 #define CTF_MODEL_LP64  2       /* object data model is LP64 */
 124 #ifdef _LP64
 125 #define CTF_MODEL_NATIVE        CTF_MODEL_LP64
 126 #else
 127 #define CTF_MODEL_NATIVE        CTF_MODEL_ILP32
 128 #endif

 130 /*
 131  * Dynamic CTF containers can be created using ctf_create().  The ctf_add_*
 132  * routines can be used to add new definitions to the dynamic container.
 133  * New types are labeled as root or non-root to determine whether they are
 134  * visible at the top-level program scope when subsequently doing a lookup.
 135  */
 136 #define CTF_ADD_NONROOT 0       /* type only visible in nested scope */
 137 #define CTF_ADD_ROOT    1       /* type visible at top-level scope */

 139 /*
 140  * These typedefs are used to define the signature for callback functions
 141  * that can be used with the iteration and visit functions below:
 142  */
 143 typedef int ctf_visit_f(const char *, ctf_id_t, ulong_t, int, void *);
 144 typedef int ctf_member_f(const char *, ctf_id_t, ulong_t, void *);
 145 typedef int ctf_enum_f(const char *, int, void *);
 146 typedef int ctf_type_f(ctf_id_t, void *);
 147 typedef int ctf_label_f(const char *, const ctf_lblinfo_t *, void *);

 149 extern ctf_file_t *ctf_bufopen(const ctf_sect_t *, const ctf_sect_t *,
 150     const ctf_sect_t *, int *);
 151 extern ctf_file_t *ctf_fdopen(int, int *);
 152 extern ctf_file_t *ctf_open(const char *, int *);
```

```
153 extern ctf_file_t *ctf_create(int *);
154 extern ctf_file_t *ctf_dup(ctf_file_t *);
155 #endif /* ! codereview */
156 extern void ctf_close(ctf_file_t *);

158 extern ctf_file_t *ctf_parent_file(ctf_file_t *);
159 extern const char *ctf_parent_name(ctf_file_t *);

161 extern int ctf_import(ctf_file_t *, ctf_file_t *);
162 extern int ctf_setmodel(ctf_file_t *, int);
163 extern int ctf_getmodel(ctf_file_t *);

165 extern void ctf_setspecific(ctf_file_t *, void *);
166 extern void *ctf_getspecific(ctf_file_t *);

168 extern int ctf_errno(ctf_file_t *);
169 extern const char *ctf_errmsg(int);
170 extern int ctf_version(int);

172 extern int ctf_func_info(ctf_file_t *, ulong_t, ctf_funcinfo_t *);
173 extern int ctf_func_args(ctf_file_t *, ulong_t, uint_t, ctf_id_t *);

175 extern ctf_id_t ctf_lookup_by_name(ctf_file_t *, const char *);
176 extern ctf_id_t ctf_lookup_by_symbol(ctf_file_t *, ulong_t);

178 extern ctf_id_t ctf_type_resolve(ctf_file_t *, ctf_id_t);
179 extern ssize_t ctf_type_lname(ctf_file_t *, ctf_id_t, char *, size_t);
180 extern char *ctf_type_name(ctf_file_t *, ctf_id_t, char *, size_t);
181 extern char *ctf_type_qname(ctf_file_t *, ctf_id_t, char *, size_t,
182     const char *);
183 #endif /* ! codereview */
184 extern ssize_t ctf_type_size(ctf_file_t *, ctf_id_t);
185 extern ssize_t ctf_type_align(ctf_file_t *, ctf_id_t);
186 extern int ctf_type_kind(ctf_file_t *, ctf_id_t);
187 extern ctf_id_t ctf_type_reference(ctf_file_t *, ctf_id_t);
188 extern ctf_id_t ctf_type_pointer(ctf_file_t *, ctf_id_t);
189 extern int ctf_type_encoding(ctf_file_t *, ctf_id_t, ctf_encoding_t *);
190 extern int ctf_type_visit(ctf_file_t *, ctf_id_t, ctf_visit_f *, void *);
191 extern int ctf_type_cmp(ctf_file_t *, ctf_id_t, ctf_file_t *, ctf_id_t);
192 extern int ctf_type_compat(ctf_file_t *, ctf_id_t, ctf_file_t *, ctf_id_t);

194 extern int ctf_member_info(ctf_file_t *, ctf_id_t, const char *,
195     ctf_membinfo_t *);
196 extern int ctf_array_info(ctf_file_t *, ctf_id_t, ctf_arinfo_t *);

198 extern const char *ctf_enum_name(ctf_file_t *, ctf_id_t, int);
199 extern int ctf_enum_value(ctf_file_t *, ctf_id_t, const char *, int *);

201 extern const char *ctf_label_topmost(ctf_file_t *);
202 extern int ctf_label_info(ctf_file_t *, const char *, ctf_lblinfo_t *);

204 extern int ctf_member_iter(ctf_file_t *, ctf_id_t, ctf_member_f *, void *);
205 extern int ctf_enum_iter(ctf_file_t *, ctf_id_t, ctf_enum_f *, void *);
206 extern int ctf_type_iter(ctf_file_t *, ctf_type_f *, void *);
207 extern int ctf_label_iter(ctf_file_t *, ctf_label_f *, void *);

209 extern ctf_id_t ctf_add_array(ctf_file_t *, uint_t, const ctf_arinfo_t *);
210 extern ctf_id_t ctf_add_const(ctf_file_t *, uint_t, ctf_id_t);
211 extern ctf_id_t ctf_add_enum(ctf_file_t *, uint_t, const char *);
212 extern ctf_id_t ctf_add_float(ctf_file_t *, uint_t,
213     const char *, const ctf_encoding_t *);
214 extern ctf_id_t ctf_add_forward(ctf_file_t *, uint_t, const char *, uint_t);
215 extern ctf_id_t ctf_add_function(ctf_file_t *, uint_t,
216     const ctf_funcinfo_t *, const ctf_id_t *);
217 extern ctf_id_t ctf_add_integer(ctf_file_t *, uint_t,
218     const char *, const ctf_encoding_t *);
```



```
219 extern ctf_id_t ctf_add_pointer(ctf_file_t *, uint_t, ctf_id_t);
220 extern ctf_id_t ctf_add_type(ctf_file_t *, ctf_file_t *, ctf_id_t);
221 extern ctf_id_t ctf_add_typedef(ctf_file_t *, uint_t, const char *, ctf_id_t);
222 extern ctf_id_t ctf_add_restrict(ctf_file_t *, uint_t, ctf_id_t);
223 extern ctf_id_t ctf_add_struct(ctf_file_t *, uint_t, const char *);
224 extern ctf_id_t ctf_add_union(ctf_file_t *, uint_t, const char *);
225 extern ctf_id_t ctf_add_volatile(ctf_file_t *, uint_t, ctf_id_t);

227 extern int ctf_add_enumerator(ctf_file_t *, ctf_id_t, const char *, int);
228 extern int ctf_add_member(ctf_file_t *, ctf_id_t, const char *, ctf_id_t);

230 extern int ctf_set_array(ctf_file_t *, ctf_id_t, const ctf_arinfo_t *);

232 extern int ctf_delete_type(ctf_file_t *, ctf_id_t);

234 extern int ctf_update(ctf_file_t *);
235 extern int ctf_discard(ctf_file_t *);
236 extern int ctf_write(ctf_file_t *, int);

238 #ifdef _KERNEL

240 struct module;
241 extern ctf_file_t *ctf_modopen(struct module *, int *);

243 #endif

245 #ifdef  __cplusplus
246 }
247 #endif

249 #endif  /* _CTF_API_H */
```

```
**********************************************************
  102050 Tue Jan 14 16:49:04 2014
new/usr/src/uts/common/sys/dtrace.h
4474 DTrace Userland CTF Support
4475 DTrace userland Keyword
4476 DTrace tests should be better citizens
4479 pid provider types
4480 dof emulation missing checks
Reviewed by: Bryan Cantrill <bryan@joyent.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, Joyent, Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

#ifndef _SYS_DTRACE_H
#define _SYS_DTRACE_H

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * DTrace Dynamic Tracing Software: Kernel Interfaces
 *
 * Note: The contents of this file are private to the implementation of the
 * Solaris system and DTrace subsystem and are subject to change at any time
 * without notice.  Applications and drivers using these interfaces will fail
 * to run on future releases.  These interfaces should not be used for any
 * purpose except those expressly outlined in dtrace(7D) and libdtrace(3LIB).
 * Please refer to the "Solaris Dynamic Tracing Guide" for more information.
 */

#ifndef _ASM

#include <sys/types.h>
#include <sys/modctl.h>
#include <sys/processor.h>
```



```
#include <sys/systm.h>
#include <sys/ctf_api.h>
#include <sys/cyclic.h>
#include <sys/int_limits.h>

/*
 * DTrace Universal Constants and Typedefs
 */
#define DTRACE_CPUALL           -1      /* all CPUs */
#define DTRACE_IDNONE           0       /* invalid probe identifier */
#define DTRACE_EPIDNONE         0       /* invalid enabled probe identifier */
#define DTRACE_AGGIDNONE        0       /* invalid aggregation identifier */
#define DTRACE_AGGVARIDNONE     0       /* invalid aggregation variable ID */
#define DTRACE_CACHEIDNONE      0       /* invalid predicate cache */
#define DTRACE_PROVNONE         0       /* invalid provider identifier */
#define DTRACE_METAPROVNONE     0       /* invalid meta-provider identifier */
#define DTRACE_ARGNONE          -1      /* invalid argument index */

#define DTRACE_PROVNAMELEN      64
#define DTRACE_MODNAMELEN       64
#define DTRACE_FUNCNAMELEN      128
#define DTRACE_NAMELEN          64
#define DTRACE_FULLNAMELEN      (DTRACE_PROVNAMELEN + DTRACE_MODNAMELEN + \
                                DTRACE_FUNCNAMELEN + DTRACE_NAMELEN + 4)
#define DTRACE_ARGTYPELEN       128

typedef uint32_t dtrace_id_t;           /* probe identifier */
typedef uint32_t dtrace_epid_t;         /* enabled probe identifier */
typedef uint32_t dtrace_aggid_t;        /* aggregation identifier */
typedef int64_t dtrace_aggvarid_t;      /* aggregation variable identifier */
typedef uint16_t dtrace_actkind_t;      /* action kind */
typedef int64_t dtrace_optval_t;        /* option value */
typedef uint32_t dtrace_cacheid_t;      /* predicate cache identifier */

typedef enum dtrace_probespec {
        DTRACE_PROBESPEC_NONE = -1,
        DTRACE_PROBESPEC_PROVIDER = 0,
        DTRACE_PROBESPEC_MOD,
        DTRACE_PROBESPEC_FUNC,
        DTRACE_PROBESPEC_NAME
} dtrace_probespec_t;
```

```
______unchanged_portion_omitted_
```

```
#define DIF_TYPE_CTF            0       /* type is a CTF type */
#define DIF_TYPE_STRING         1       /* type is a D string */

#define DIF_TF_BYREF            0x1     /* type is passed by reference */
#define DIF_TF_BYUREF           0x2     /* user type is passed by reference */
#endif /* ! codereview */

/*
 * A DTrace Intermediate Format variable record is used to describe each of the
 * variables referenced by a given DIF object.  It contains an integer variable
 * identifier along with variable scope and properties, as shown below.  The
 * size of this structure must be sizeof (int) aligned.
 */
typedef struct dtrace_difv {
        uint32_t dtdv_name;             /* variable name index in dtdo_strtab */
        uint32_t dtdv_id;               /* variable reference identifier */
        uint8_t dtdv_kind;              /* variable kind (see below) */
        uint8_t dtdv_scope;             /* variable scope (see below) */
        uint16_t dtdv_flags;            /* variable flags (see below) */
        dtrace_diftype_t dtdv_type;     /* variable type (see above) */
} dtrace_difv_t;

#define DIFV_KIND_ARRAY         0       /* variable is an array of quantities */
```

```
376 #define DIFV_KIND_SCALAR        1       /* variable is a scalar quantity */

378 #define DIFV_SCOPE_GLOBAL       0       /* variable has global scope */
379 #define DIFV_SCOPE_THREAD       1       /* variable has thread scope */
380 #define DIFV_SCOPE_LOCAL        2       /* variable has local scope */

382 #define DIFV_F_REF              0x1     /* variable is referenced by DIFO */
383 #define DIFV_F_MOD              0x2     /* variable is written by DIFO */

385 /*
386  * DTrace Actions
387  *
388  * The upper byte determines the class of the action; the low bytes determines
389  * the specific action within that class.  The classes of actions are as
390  * follows:
391  *
392  *   [ no class ]                 <= May record process- or kernel-related data
393  *   DTRACEACT_PROC               <= Only records process-related data
394  *   DTRACEACT_PROC_DESTRUCTIVE   <= Potentially destructive to processes
395  *   DTRACEACT_KERNEL             <= Only records kernel-related data
396  *   DTRACEACT_KERNEL_DESTRUCTIVE <= Potentially destructive to the kernel
397  *   DTRACEACT_SPECULATIVE        <= Speculation-related action
398  *   DTRACEACT_AGGREGATION        <= Aggregating action
399  */
400 #define DTRACEACT_NONE                  0       /* no action */
401 #define DTRACEACT_DIFEXPR               1       /* action is DIF expression */
402 #define DTRACEACT_EXIT                  2       /* exit() action */
403 #define DTRACEACT_PRINTF                3       /* printf() action */
404 #define DTRACEACT_PRINTA                4       /* printa() action */
405 #define DTRACEACT_LIBACT                5       /* library-controlled action */
406 #define DTRACEACT_TRACEMEM              6       /* tracemem() action */
407 #define DTRACEACT_TRACEMEM_DYNSIZE      7       /* dynamic tracemem() size */

409 #define DTRACEACT_PROC                  0x0100
410 #define DTRACEACT_USTACK                (DTRACEACT_PROC + 1)
411 #define DTRACEACT_JSTACK                (DTRACEACT_PROC + 2)
412 #define DTRACEACT_USYM                  (DTRACEACT_PROC + 3)
413 #define DTRACEACT_UMOD                  (DTRACEACT_PROC + 4)
414 #define DTRACEACT_UADDR                 (DTRACEACT_PROC + 5)

416 #define DTRACEACT_PROC_DESTRUCTIVE      0x0200
417 #define DTRACEACT_STOP                  (DTRACEACT_PROC_DESTRUCTIVE + 1)
418 #define DTRACEACT_RAISE                 (DTRACEACT_PROC_DESTRUCTIVE + 2)
419 #define DTRACEACT_SYSTEM                (DTRACEACT_PROC_DESTRUCTIVE + 3)
420 #define DTRACEACT_FREOPEN               (DTRACEACT_PROC_DESTRUCTIVE + 4)

422 #define DTRACEACT_PROC_CONTROL          0x0300

424 #define DTRACEACT_KERNEL                0x0400
425 #define DTRACEACT_STACK                 (DTRACEACT_KERNEL + 1)
426 #define DTRACEACT_SYM                   (DTRACEACT_KERNEL + 2)
427 #define DTRACEACT_MOD                   (DTRACEACT_KERNEL + 3)

429 #define DTRACEACT_KERNEL_DESTRUCTIVE    0x0500
430 #define DTRACEACT_BREAKPOINT            (DTRACEACT_KERNEL_DESTRUCTIVE + 1)
431 #define DTRACEACT_PANIC                 (DTRACEACT_KERNEL_DESTRUCTIVE + 2)
432 #define DTRACEACT_CHILL                 (DTRACEACT_KERNEL_DESTRUCTIVE + 3)

434 #define DTRACEACT_SPECULATIVE           0x0600
435 #define DTRACEACT_SPECULATE             (DTRACEACT_SPECULATIVE + 1)
436 #define DTRACEACT_COMMIT                (DTRACEACT_SPECULATIVE + 2)
437 #define DTRACEACT_DISCARD               (DTRACEACT_SPECULATIVE + 3)

439 #define DTRACEACT_CLASS(x)              ((x) & 0xff00)

441 #define DTRACEACT_ISDESTRUCTIVE(x)      \
```



```
442         (DTRACEACT_CLASS(x) == DTRACEACT_PROC_DESTRUCTIVE || \
443         DTRACEACT_CLASS(x) == DTRACEACT_KERNEL_DESTRUCTIVE)

445 #define DTRACEACT_ISSPECULATIVE(x)      \
446         (DTRACEACT_CLASS(x) == DTRACEACT_SPECULATIVE)

448 #define DTRACEACT_ISPRINTFLIKE(x)       \
449         ((x) == DTRACEACT_PRINTF || (x) == DTRACEACT_PRINTA || \
450         (x) == DTRACEACT_SYSTEM || (x) == DTRACEACT_FREOPEN)

452 /*
453  * DTrace Aggregating Actions
454  *
455  * These are functions f(x) for which the following is true:
456  *
457  *    f(f(x_0) U f(x_1) U ... U f(x_n)) = f(x_0 U x_1 U ... U x_n)
458  *
459  * where x_n is a set of arbitrary data.  Aggregating actions are in their own
460  * DTrace action class, DTTRACEACT_AGGREGATION.  The macros provided here allow
461  * for easier processing of the aggregation argument and data payload for a few
462  * aggregating actions (notably:  quantize(), lquantize(), and ustack()).
463  */
464 #define DTRACEACT_AGGREGATION           0x0700
465 #define DTRACEAGG_COUNT                 (DTRACEACT_AGGREGATION + 1)
466 #define DTRACEAGG_MIN                   (DTRACEACT_AGGREGATION + 2)
467 #define DTRACEAGG_MAX                   (DTRACEACT_AGGREGATION + 3)
468 #define DTRACEAGG_AVG                   (DTRACEACT_AGGREGATION + 4)
469 #define DTRACEAGG_SUM                   (DTRACEACT_AGGREGATION + 5)
470 #define DTRACEAGG_STDDEV                (DTRACEACT_AGGREGATION + 6)
471 #define DTRACEAGG_QUANTIZE              (DTRACEACT_AGGREGATION + 7)
472 #define DTRACEAGG_LQUANTIZE             (DTRACEACT_AGGREGATION + 8)
473 #define DTRACEAGG_LLQUANTIZE            (DTRACEACT_AGGREGATION + 9)

475 #define DTRACEACT_ISAGG(x)              \
476         (DTRACEACT_CLASS(x) == DTRACEACT_AGGREGATION)

478 #define DTRACE_QUANTIZE_NBUCKETS        \
479         (((sizeof (uint64_t) * NBBY) - 1) * 2 + 1)

481 #define DTRACE_QUANTIZE_ZEROBUCKET      ((sizeof (uint64_t) * NBBY) - 1)

483 #define DTRACE_QUANTIZE_BUCKETVAL(buck)                                 \
484         (int64_t)((buck) < DTRACE_QUANTIZE_ZEROBUCKET ?                 \
485         -(1LL << (DTRACE_QUANTIZE_ZEROBUCKET - 1 - (buck))) :           \
486         (buck) == DTRACE_QUANTIZE_ZEROBUCKET ? 0 :                      \
487         1LL << ((buck) - DTRACE_QUANTIZE_ZEROBUCKET - 1))

489 #define DTRACE_LQUANTIZE_STEPSHIFT              48
490 #define DTRACE_LQUANTIZE_STEPMASK               ((uint64_t)UINT16_MAX << 48)
491 #define DTRACE_LQUANTIZE_LEVELSHIFT             32
492 #define DTRACE_LQUANTIZE_LEVELMASK              ((uint64_t)UINT16_MAX << 32)
493 #define DTRACE_LQUANTIZE_BASESHIFT              0
494 #define DTRACE_LQUANTIZE_BASEMASK               UINT32_MAX

496 #define DTRACE_LQUANTIZE_STEP(x)                \
497         (uint16_t)(((x) & DTRACE_LQUANTIZE_STEPMASK) >> \
498         DTRACE_LQUANTIZE_STEPSHIFT)

500 #define DTRACE_LQUANTIZE_LEVELS(x)              \
501         (uint16_t)(((x) & DTRACE_LQUANTIZE_LEVELMASK) >> \
502         DTRACE_LQUANTIZE_LEVELSHIFT)

504 #define DTRACE_LQUANTIZE_BASE(x)                \
505         (int32_t)(((x) & DTRACE_LQUANTIZE_BASEMASK) >> \
506         DTRACE_LQUANTIZE_BASESHIFT)
```

```
508 #define DTRACE_LLQUANTIZE_FACTORSHIFT           48
509 #define DTRACE_LLQUANTIZE_FACTORMASK            ((uint64_t)UINT16_MAX << 48)
510 #define DTRACE_LLQUANTIZE_LOWSHIFT              32
511 #define DTRACE_LLQUANTIZE_LOWMASK               ((uint64_t)UINT16_MAX << 32)
512 #define DTRACE_LLQUANTIZE_HIGHSHIFT             16
513 #define DTRACE_LLQUANTIZE_HIGHMASK              ((uint64_t)UINT16_MAX << 16)
514 #define DTRACE_LLQUANTIZE_NSTEPSHIFT            0
515 #define DTRACE_LLQUANTIZE_NSTEPMASK             UINT16_MAX

517 #define DTRACE_LLQUANTIZE_FACTOR(x)             \
518         (uint16_t)(((x) & DTRACE_LLQUANTIZE_FACTORMASK) >> \
519         DTRACE_LLQUANTIZE_FACTORSHIFT)

521 #define DTRACE_LLQUANTIZE_LOW(x)                \
522         (uint16_t)(((x) & DTRACE_LLQUANTIZE_LOWMASK) >> \
523         DTRACE_LLQUANTIZE_LOWSHIFT)

525 #define DTRACE_LLQUANTIZE_HIGH(x)               \
526         (uint16_t)(((x) & DTRACE_LLQUANTIZE_HIGHMASK) >> \
527         DTRACE_LLQUANTIZE_HIGHSHIFT)

529 #define DTRACE_LLQUANTIZE_NSTEP(x)              \
530         (uint16_t)(((x) & DTRACE_LLQUANTIZE_NSTEPMASK) >> \
531         DTRACE_LLQUANTIZE_NSTEPSHIFT)

533 #define DTRACE_USTACK_NFRAMES(x)        (uint32_t)((x) & UINT32_MAX)
534 #define DTRACE_USTACK_STRSIZE(x)        (uint32_t)((x) >> 32)
535 #define DTRACE_USTACK_ARG(x, y)         \
536         ((((uint64_t)(y)) << 32) | ((x) & UINT32_MAX))

538 #ifndef _LP64
539 #ifndef _LITTLE_ENDIAN
540 #define DTRACE_PTR(type, name)  uint32_t name##pad; type *name
541 #else
542 #define DTRACE_PTR(type, name)  type *name; uint32_t name##pad
543 #endif
544 #else
545 #define DTRACE_PTR(type, name)  type *name
546 #endif

548 /*
549  * DTrace Object Format (DOF)
550  *
551  * DTrace programs can be persistently encoded in the DOF format so that they
552  * may be embedded in other programs (for example, in an ELF file) or in the
553  * dtrace driver configuration file for use in anonymous tracing.  The DOF
554  * format is versioned and extensible so that it can be revised and so that
555  * internal data structures can be modified or extended compatibly.  All DOF
556  * structures use fixed-size types, so the 32-bit and 64-bit representations
557  * are identical and consumers can use either data model transparently.
558  *
559  * The file layout is structured as follows:
560  *
561  * +---------------+------------------+----- ... ----+---- ... ------+
562  * |   dof_hdr_t   |  dof_sec_t[ ... ] |   loadable   | non-loadable  |
563  * | (file header) | (section headers) | section data | section data  |
564  * +---------------+------------------+----- ... ----+---- ... ------+
565  * |<------------ dof_hdr.dofh_loadsz --------------->|               |
566  * |<------------ dof_hdr.dofh_filesz ------------------------------->|
567  *
568  * The file header stores meta-data including a magic number, data model for
569  * the instrumentation, data encoding, and properties of the DIF code within.
570  * The header describes its own size and the size of the section headers.  By
571  * convention, an array of section headers follows the file header, and then
572  * the data for all loadable sections and unloadable sections.  This permits
573  * consumer code to easily download the headers and all loadable data into the
```



```
574  * DTrace driver in one contiguous chunk, omitting other extraneous sections.
575  *
576  * The section headers describe the size, offset, alignment, and section type
577  * for each section.  Sections are described using a set of #defines that tell
578  * the consumer what kind of data is expected.  Sections can contain links to
579  * other sections by storing a dof_secidx_t, an index into the section header
580  * array, inside of the section data structures.  The section header includes
581  * an entry size so that sections with data arrays can grow their structures.
582  *
583  * The DOF data itself can contain many snippets of DIF (i.e. >1 DIFOs), which
584  * are represented themselves as a collection of related DOF sections.  This
585  * permits us to change the set of sections associated with a DIFO over time,
586  * and also permits us to encode DIFOs that contain different sets of sections.
587  * When a DOF section wants to refer to a DIFO, it stores the dof_secidx_t of a
588  * section of type DOF_SECT_DIFOHDR.  This section's data is then an array of
589  * dof_secidx_t's which in turn denote the sections associated with this DIFO.
590  *
591  * This loose coupling of the file structure (header and sections) to the
592  * structure of the DTrace program itself (ECB descriptions, action
593  * descriptions, and DIFOs) permits activities such as relocation processing
594  * to occur in a single pass without having to understand D program structure.
595  *
596  * Finally, strings are always stored in ELF-style string tables along with a
597  * string table section index and string table offset.  Therefore strings in
598  * DOF are always arbitrary-length and not bound to the current implementation.
599  */

601 #define DOF_ID_SIZE     16      /* total size of dofh_ident[] in bytes */

603 typedef struct dof_hdr {
604         uint8_t dofh_ident[DOF_ID_SIZE]; /* identification bytes (see below) */
605         uint32_t dofh_flags;            /* file attribute flags (if any) */
606         uint32_t dofh_hdrsize;          /* size of file header in bytes */
607         uint32_t dofh_secsize;          /* size of section header in bytes */
608         uint32_t dofh_secnum;           /* number of section headers */
609         uint64_t dofh_secoff;           /* file offset of section headers */
610         uint64_t dofh_loadsz;           /* file size of loadable portion */
611         uint64_t dofh_filesz;           /* file size of entire DOF file */
612         uint64_t dofh_pad;              /* reserved for future use */
613 } dof_hdr_t;

615 #define DOF_ID_MAG0     0       /* first byte of magic number */
616 #define DOF_ID_MAG1     1       /* second byte of magic number */
617 #define DOF_ID_MAG2     2       /* third byte of magic number */
618 #define DOF_ID_MAG3     3       /* fourth byte of magic number */
619 #define DOF_ID_MODEL    4       /* DOF data model (see below) */
620 #define DOF_ID_ENCODING 5       /* DOF data encoding (see below) */
621 #define DOF_ID_VERSION  6       /* DOF file format major version (see below) */
622 #define DOF_ID_DIFVERS  7       /* DIF instruction set version */
623 #define DOF_ID_DIFIREG  8       /* DIF integer registers used by compiler */
624 #define DOF_ID_DIFTREG  9       /* DIF tuple registers used by compiler */
625 #define DOF_ID_PAD      10      /* start of padding bytes (all zeroes) */

627 #define DOF_MAG_MAG0    0x7F    /* DOF_ID_MAG[0-3] */
628 #define DOF_MAG_MAG1    'D'
629 #define DOF_MAG_MAG2    'O'
630 #define DOF_MAG_MAG3    'F'

632 #define DOF_MAG_STRING  "\177DOF"
633 #define DOF_MAG_STRLEN  4

635 #define DOF_MODEL_NONE  0       /* DOF_ID_MODEL */
636 #define DOF_MODEL_ILP32 1
637 #define DOF_MODEL_LP64  2

639 #ifdef _LP64
```

```
640 #define DOF_MODEL_NATIVE        DOF_MODEL_LP64
641 #else
642 #define DOF_MODEL_NATIVE        DOF_MODEL_ILP32
643 #endif

645 #define DOF_ENCODE_NONE 0       /* DOF_ID_ENCODING */
646 #define DOF_ENCODE_LSB  1
647 #define DOF_ENCODE_MSB  2

649 #ifdef _BIG_ENDIAN
650 #define DOF_ENCODE_NATIVE       DOF_ENCODE_MSB
651 #else
652 #define DOF_ENCODE_NATIVE       DOF_ENCODE_LSB
653 #endif

655 #define DOF_VERSION_1   1       /* DOF version 1: Solaris 10 FCS */
656 #define DOF_VERSION_2   2       /* DOF version 2: Solaris Express 6/06 */
657 #define DOF_VERSION     DOF_VERSION_2   /* Latest DOF version */

659 #define DOF_FL_VALID    0       /* mask of all valid dofh_flags bits */

661 typedef uint32_t dof_secidx_t;  /* section header table index type */
662 typedef uint32_t dof_stridx_t;  /* string table index type */

664 #define DOF_SECIDX_NONE (-1U)   /* null value for section indices */
665 #define DOF_STRIDX_NONE (-1U)   /* null value for string indices */

667 typedef struct dof_sec {
668         uint32_t dofs_type;     /* section type (see below) */
669         uint32_t dofs_align;    /* section data memory alignment */
670         uint32_t dofs_flags;    /* section flags (if any) */
671         uint32_t dofs_entsize;  /* size of section entry (if table) */
672         uint64_t dofs_offset;   /* offset of section data within file */
673         uint64_t dofs_size;     /* size of section data in bytes */
674 } dof_sec_t;

676 #define DOF_SECT_NONE           0       /* null section */
677 #define DOF_SECT_COMMENTS       1       /* compiler comments */
678 #define DOF_SECT_SOURCE         2       /* D program source code */
679 #define DOF_SECT_ECBDESC        3       /* dof_ecbdesc_t */
680 #define DOF_SECT_PROBEDESC      4       /* dof_probedesc_t */
681 #define DOF_SECT_ACTDESC        5       /* dof_actdesc_t array */
682 #define DOF_SECT_DIFOHDR        6       /* dof_difohdr_t (variable length) */
683 #define DOF_SECT_DIF            7       /* uint32_t array of byte code */
684 #define DOF_SECT_STRTAB         8       /* string table */
685 #define DOF_SECT_VARTAB         9       /* dtrace_difv_t array */
686 #define DOF_SECT_RELTAB         10      /* dof_relodesc_t array */
687 #define DOF_SECT_TYPTAB         11      /* dtrace_diftype_t array */
688 #define DOF_SECT_URELHDR        12      /* dof_relohdr_t (user relocations) */
689 #define DOF_SECT_KRELHDR        13      /* dof_relohdr_t (kernel relocations) */
690 #define DOF_SECT_OPTDESC        14      /* dof_optdesc_t array */
691 #define DOF_SECT_PROVIDER       15      /* dof_provider_t */
692 #define DOF_SECT_PROBES         16      /* dof_probe_t array */
693 #define DOF_SECT_PRARGS         17      /* uint8_t array (probe arg mappings) */
694 #define DOF_SECT_PROFFS         18      /* uint32_t array (probe arg offsets) */
695 #define DOF_SECT_INTTAB         19      /* uint64_t array */
696 #define DOF_SECT_UTSNAME        20      /* struct utsname */
697 #define DOF_SECT_XLTAB          21      /* dof_xlref_t array */
698 #define DOF_SECT_XLMEMBERS      22      /* dof_xlmember_t array */
699 #define DOF_SECT_XLIMPORT       23      /* dof_xlator_t */
700 #define DOF_SECT_XLEXPORT       24      /* dof_xlator_t */
701 #define DOF_SECT_PREXPORT       25      /* dof_secidx_t array (exported objs) */
702 #define DOF_SECT_PRENOFFS       26      /* uint32_t array (enabled offsets) */

704 #define DOF_SECF_LOAD           1       /* section should be loaded */
```



```
706 #define DOF_SEC_ISLOADABLE(x)                                   \
707         (((x) == DOF_SECT_ECBDESC) || ((x) == DOF_SECT_PROBEDESC) ||    \
708         ((x) == DOF_SECT_ACTDESC) || ((x) == DOF_SECT_DIFOHDR) ||       \
709         ((x) == DOF_SECT_DIF) || ((x) == DOF_SECT_STRTAB) ||            \
710         ((x) == DOF_SECT_VARTAB) || ((x) == DOF_SECT_RELTAB) ||         \
711         ((x) == DOF_SECT_TYPTAB) || ((x) == DOF_SECT_URELHDR) ||        \
712         ((x) == DOF_SECT_KRELHDR) || ((x) == DOF_SECT_OPTDESC) ||       \
713         ((x) == DOF_SECT_PROVIDER) || ((x) == DOF_SECT_PROBES) ||       \
714         ((x) == DOF_SECT_PRARGS) || ((x) == DOF_SECT_PROFFS) ||         \
715         ((x) == DOF_SECT_INTTAB) || ((x) == DOF_SECT_XLTAB) ||          \
716         ((x) == DOF_SECT_XLMEMBERS) || ((x) == DOF_SECT_XLIMPORT) ||    \
717         ((x) == DOF_SECT_XLIMPORT) || ((x) == DOF_SECT_XLEXPORT) ||     \
718         ((x) == DOF_SECT_PREXPORT) || ((x) == DOF_SECT_PRENOFFS))

720 typedef struct dof_ecbdesc {
721         dof_secidx_t dofe_probes;       /* link to DOF_SECT_PROBEDESC */
722         dof_secidx_t dofe_pred;         /* link to DOF_SECT_DIFOHDR */
723         dof_secidx_t dofe_actions;      /* link to DOF_SECT_ACTDESC */
724         uint32_t dofe_pad;              /* reserved for future use */
725         uint64_t dofe_uarg;             /* user-supplied library argument */
726 } dof_ecbdesc_t;

728 typedef struct dof_probedesc {
729         dof_secidx_t dofp_strtab;       /* link to DOF_SECT_STRTAB section */
730         dof_stridx_t dofp_provider;     /* provider string */
731         dof_stridx_t dofp_mod;          /* module string */
732         dof_stridx_t dofp_func;         /* function string */
733         dof_stridx_t dofp_name;         /* name string */
734         uint32_t dofp_id;               /* probe identifier (or zero) */
735 } dof_probedesc_t;

737 typedef struct dof_actdesc {
738         dof_secidx_t dofa_difo;         /* link to DOF_SECT_DIFOHDR */
739         dof_secidx_t dofa_strtab;       /* link to DOF_SECT_STRTAB section */
740         uint32_t dofa_kind;             /* action kind (DTRACEACT_* constant) */
741         uint32_t dofa_ntuple;           /* number of subsequent tuple actions */
742         uint64_t dofa_arg;              /* kind-specific argument */
743         uint64_t dofa_uarg;             /* user-supplied argument */
744 } dof_actdesc_t;

746 typedef struct dof_difohdr {
747         dtrace_diftype_t dofd_rtype;    /* return type for this fragment */
748         dof_secidx_t dofd_links[1];     /* variable length array of indices */
749 } dof_difohdr_t;

751 typedef struct dof_relohdr {
752         dof_secidx_t dofr_strtab;       /* link to DOF_SECT_STRTAB for names */
753         dof_secidx_t dofr_relsec;       /* link to DOF_SECT_RELTAB for relos */
754         dof_secidx_t dofr_tgtsec;       /* link to section we are relocating */
755 } dof_relohdr_t;

757 typedef struct dof_relodesc {
758         dof_stridx_t dofr_name;         /* string name of relocation symbol */
759         uint32_t dofr_type;             /* relo type (DOF_RELO_* constant) */
760         uint64_t dofr_offset;           /* byte offset for relocation */
761         uint64_t dofr_data;             /* additional type-specific data */
762 } dof_relodesc_t;

764 #define DOF_RELO_NONE   0               /* empty relocation entry */
765 #define DOF_RELO_SETX   1               /* relocate setx value */

767 typedef struct dof_optdesc {
768         uint32_t dofo_option;           /* option identifier */
769         dof_secidx_t dofo_strtab;       /* string table, if string option */
770         uint64_t dofo_value;            /* option value or string index */
771 } dof_optdesc_t;
```

```
773 typedef uint32_t dof_attr_t;           /* encoded stability attributes */

775 #define DOF_ATTR(n, d, c)       (((n) << 24) | ((d) << 16) | ((c) << 8))
776 #define DOF_ATTR_NAME(a)        (((a) >> 24) & 0xff)
777 #define DOF_ATTR_DATA(a)        (((a) >> 16) & 0xff)
778 #define DOF_ATTR_CLASS(a)       (((a) >>  8) & 0xff)

780 typedef struct dof_provider {
781         dof_secidx_t dofpv_strtab;      /* link to DOF_SECT_STRTAB section */
782         dof_secidx_t dofpv_probes;      /* link to DOF_SECT_PROBES section */
783         dof_secidx_t dofpv_prargs;      /* link to DOF_SECT_PRARGS section */
784         dof_secidx_t dofpv_proffs;      /* link to DOF_SECT_PROFFS section */
785         dof_stridx_t dofpv_name;        /* provider name string */
786         dof_attr_t dofpv_provattr;      /* provider attributes */
787         dof_attr_t dofpv_modattr;       /* module attributes */
788         dof_attr_t dofpv_funcattr;      /* function attributes */
789         dof_attr_t dofpv_nameattr;      /* name attributes */
790         dof_attr_t dofpv_argsattr;      /* args attributes */
791         dof_secidx_t dofpv_prenoffs;    /* link to DOF_SECT_PRENOFFS section */
792 } dof_provider_t;

794 typedef struct dof_probe {
795         uint64_t dofpr_addr;            /* probe base address or offset */
796         dof_stridx_t dofpr_func;        /* probe function string */
797         dof_stridx_t dofpr_name;        /* probe name string */
798         dof_stridx_t dofpr_nargv;       /* native argument type strings */
799         dof_stridx_t dofpr_xargv;       /* translated argument type strings */
800         uint32_t dofpr_argidx;          /* index of first argument mapping */
801         uint32_t dofpr_offidx;          /* index of first offset entry */
802         uint8_t dofpr_nargc;            /* native argument count */
803         uint8_t dofpr_xargc;            /* translated argument count */
804         uint16_t dofpr_noffs;           /* number of offset entries for probe */
805         uint32_t dofpr_enoffidx;        /* index of first is-enabled offset */
806         uint16_t dofpr_nenoffs;         /* number of is-enabled offsets */
807         uint16_t dofpr_pad1;            /* reserved for future use */
808         uint32_t dofpr_pad2;            /* reserved for future use */
809 } dof_probe_t;

811 typedef struct dof_xlator {
812         dof_secidx_t dofxl_members;     /* link to DOF_SECT_XLMEMBERS section */
813         dof_secidx_t dofxl_strtab;      /* link to DOF_SECT_STRTAB section */
814         dof_stridx_t dofxl_argv;        /* input parameter type strings */
815         uint32_t dofxl_argc;            /* input parameter list length */
816         dof_stridx_t dofxl_type;        /* output type string name */
817         dof_attr_t dofxl_attr;          /* output stability attributes */
818 } dof_xlator_t;

820 typedef struct dof_xlmember {
821         dof_secidx_t dofxm_difo;        /* member link to DOF_SECT_DIFOHDR */
822         dof_stridx_t dofxm_name;        /* member name */
823         dtrace_diftype_t dofxm_type;    /* member type */
824 } dof_xlmember_t;

826 typedef struct dof_xlref {
827         dof_secidx_t dofxr_xlator;      /* link to DOF_SECT_XLATORS section */
828         uint32_t dofxr_member;          /* index of referenced dof_xlmember */
829         uint32_t dofxr_argn;            /* index of argument for DIF_OP_XLARG */
830 } dof_xlref_t;

832 /*
833  * DTrace Intermediate Format Object (DIFO)
834  *
835  * A DIFO is used to store the compiled DIF for a D expression, its return
836  * type, and its string and variable tables.  The string table is a single
837  * buffer of character data into which sets instructions and variable
```



```
838  * references can reference strings using a byte offset.  The variable table
839  * is an array of dtrace_difv_t structures that describe the name and type of
840  * each variable and the id used in the DIF code.  This structure is described
841  * above in the DIF section of this header file.  The DIFO is used at both
842  * user-level (in the library) and in the kernel, but the structure is never
843  * passed between the two: the DOF structures form the only interface.  As a
844  * result, the definition can change depending on the presence of _KERNEL.
845  */
846 typedef struct dtrace_difo {
847         dif_instr_t *dtdo_buf;          /* instruction buffer */
848         uint64_t *dtdo_inttab;          /* integer table (optional) */
849         char *dtdo_strtab;              /* string table (optional) */
850         dtrace_difv_t *dtdo_vartab;     /* variable table (optional) */
851         uint_t dtdo_len;                /* length of instruction buffer */
852         uint_t dtdo_intlen;             /* length of integer table */
853         uint_t dtdo_strlen;             /* length of string table */
854         uint_t dtdo_varlen;             /* length of variable table */
855         dtrace_diftype_t dtdo_rtype;    /* return type */
856         uint_t dtdo_refcnt;             /* owner reference count */
857         uint_t dtdo_destructive;        /* invokes destructive subroutines */
858 #ifndef _KERNEL
859         dof_relodesc_t *dtdo_kreltab;   /* kernel relocations */
860         dof_relodesc_t *dtdo_ureltab;   /* user relocations */
861         struct dt_node **dtdo_xlmtab;   /* translator references */
862         uint_t dtdo_krelen;             /* length of krelo table */
863         uint_t dtdo_urelen;             /* length of urelo table */
864         uint_t dtdo_xlmlen;             /* length of translator table */
865 #endif
866 } dtrace_difo_t;

868 /*
869  * DTrace Enabling Description Structures
870  *
871  * When DTrace is tracking the description of a DTrace enabling entity (probe,
872  * predicate, action, ECB, record, etc.), it does so in a description
873  * structure.  These structures all end in "desc", and are used at both
874  * user-level and in the kernel -- but (with the exception of
875  * dtrace_probedesc_t) they are never passed between them.  Typically,
876  * user-level will use the description structures when assembling an enabling.
877  * It will then distill those description structures into a DOF object (see
878  * above), and send it into the kernel.  The kernel will again use the
879  * description structures to create a description of the enabling as it reads
880  * the DOF.  When the description is complete, the enabling will be actually
881  * created -- turning it into the structures that represent the enabling
882  * instead of merely describing it.  Not surprisingly, the description
883  * structures bear a strong resemblance to the DOF structures that act as their
884  * conduit.
885  */
886 struct dtrace_predicate;

888 typedef struct dtrace_probedesc {
889         dtrace_id_t dtpd_id;                    /* probe identifier */
890         char dtpd_provider[DTRACE_PROVNAMELEN]; /* probe provider name */
891         char dtpd_mod[DTRACE_MODNAMELEN];       /* probe module name */
892         char dtpd_func[DTRACE_FUNCNAMELEN];     /* probe function name */
893         char dtpd_name[DTRACE_NAMELEN];         /* probe name */
894 } dtrace_probedesc_t;

896 typedef struct dtrace_repldesc {
897         dtrace_probedesc_t dtrpd_match;         /* probe descr. to match */
898         dtrace_probedesc_t dtrpd_create;        /* probe descr. to create */
899 } dtrace_repldesc_t;

901 typedef struct dtrace_preddesc {
902         dtrace_difo_t *dtpdd_difo;              /* pointer to DIF object */
903         struct dtrace_predicate *dtpdd_predicate; /* pointer to predicate */
```

```
} dtrace_preddesc_t;

typedef struct dtrace_actdesc {
        dtrace_difo_t *dtad_difo;               /* pointer to DIF object */
        struct dtrace_actdesc *dtad_next;       /* next action */
        dtrace_actkind_t dtad_kind;             /* kind of action */
        uint32_t dtad_ntuple;                   /* number in tuple */
        uint64_t dtad_arg;                      /* action argument */
        uint64_t dtad_uarg;                     /* user argument */
        int dtad_refcnt;                        /* reference count */
} dtrace_actdesc_t;

typedef struct dtrace_ecbdesc {
        dtrace_actdesc_t *dted_action;          /* action description(s) */
        dtrace_preddesc_t dted_pred;            /* predicate description */
        dtrace_probedesc_t dted_probe;          /* probe description */
        uint64_t dted_uarg;                     /* library argument */
        int dted_refcnt;                        /* reference count */
} dtrace_ecbdesc_t;

/*
 * DTrace Metadata Description Structures
 *
 * DTrace separates the trace data stream from the metadata stream.  The only
 * metadata tokens placed in the data stream are the dtrace_rechdr_t (EPID +
 * timestamp) or (in the case of aggregations) aggregation identifiers.  To
 * determine the structure of the data, DTrace consumers pass the token to the
 * kernel, and receive in return a corresponding description of the enabled
 * probe (via the dtrace_eprobedesc structure) or the aggregation (via the
 * dtrace_aggdesc structure).  Both of these structures are expressed in terms
 * of record descriptions (via the dtrace_recdesc structure) that describe the
 * exact structure of the data.  Some record descriptions may also contain a
 * format identifier; this additional bit of metadata can be retrieved from the
 * kernel, for which a format description is returned via the dtrace_fmtdesc
 * structure.  Note that all four of these structures must be bitness-neutral
 * to allow for a 32-bit DTrace consumer on a 64-bit kernel.
 */
typedef struct dtrace_recdesc {
        dtrace_actkind_t dtrd_action;           /* kind of action */
        uint32_t dtrd_size;                     /* size of record */
        uint32_t dtrd_offset;                   /* offset in ECB's data */
        uint16_t dtrd_alignment;                /* required alignment */
        uint16_t dtrd_format;                   /* format, if any */
        uint64_t dtrd_arg;                      /* action argument */
        uint64_t dtrd_uarg;                     /* user argument */
} dtrace_recdesc_t;

typedef struct dtrace_eprobedesc {
        dtrace_epid_t dtepd_epid;               /* enabled probe ID */
        dtrace_id_t dtepd_probeid;              /* probe ID */
        uint64_t dtepd_uarg;                    /* library argument */
        uint32_t dtepd_size;                    /* total size */
        int dtepd_nrecs;                        /* number of records */
        dtrace_recdesc_t dtepd_rec[1];          /* records themselves */
} dtrace_eprobedesc_t;

typedef struct dtrace_aggdesc {
        DTRACE_PTR(char, dtagd_name);           /* not filled in by kernel */
        dtrace_aggvarid_t dtagd_varid;          /* not filled in by kernel */
        int dtagd_flags;                        /* not filled in by kernel */
        dtrace_aggid_t dtagd_id;                /* aggregation ID */
        dtrace_epid_t dtagd_epid;               /* enabled probe ID */
        uint32_t dtagd_size;                    /* size in bytes */
        int dtagd_nrecs;                        /* number of records */
        uint32_t dtagd_pad;                     /* explicit padding */
        dtrace_recdesc_t dtagd_rec[1];          /* record descriptions */
```



```
} dtrace_aggdesc_t;

typedef struct dtrace_fmtdesc {
        DTRACE_PTR(char, dtfd_string);          /* format string */
        int dtfd_length;                        /* length of format string */
        uint16_t dtfd_format;                   /* format identifier */
} dtrace_fmtdesc_t;

#define DTRACE_SIZEOF_EPROBEDESC(desc)                                  \
        (sizeof (dtrace_eprobedesc_t) + ((desc)->dtepd_nrecs ?  \
        (((desc)->dtepd_nrecs - 1) * sizeof (dtrace_recdesc_t)) : 0))

#define DTRACE_SIZEOF_AGGDESC(desc)                                     \
        (sizeof (dtrace_aggdesc_t) + ((desc)->dtagd_nrecs ?     \
        (((desc)->dtagd_nrecs - 1) * sizeof (dtrace_recdesc_t)) : 0))

/*
 * DTrace Option Interface
 *
 * Run-time DTrace options are set and retrieved via DOF_SECT_OPTDESC sections
 * in a DOF image.  The dof_optdesc structure contains an option identifier and
 * an option value.  The valid option identifiers are found below; the mapping
 * between option identifiers and option identifying strings is maintained at
 * user-level.  Note that the value of DTRACEOPT_UNSET is such that all of the
 * following are potentially valid option values:  all positive integers, zero
 * and negative one.  Some options (notably "bufpolicy" and "bufresize") take
 * predefined tokens as their values; these are defined with
 * DTRACEOPT_{option}_{token}.
 */
#define DTRACEOPT_BUFSIZE       0       /* buffer size */
#define DTRACEOPT_BUFPOLICY     1       /* buffer policy */
#define DTRACEOPT_DYNVARSIZE    2       /* dynamic variable size */
#define DTRACEOPT_AGGSIZE       3       /* aggregation size */
#define DTRACEOPT_SPECSIZE      4       /* speculation size */
#define DTRACEOPT_NSPEC         5       /* number of speculations */
#define DTRACEOPT_STRSIZE       6       /* string size */
#define DTRACEOPT_CLEANRATE     7       /* dynvar cleaning rate */
#define DTRACEOPT_CPU           8       /* CPU to trace */
#define DTRACEOPT_BUFRESIZE     9       /* buffer resizing policy */
#define DTRACEOPT_GRABANON      10      /* grab anonymous state, if any */
#define DTRACEOPT_FLOWINDENT    11      /* indent function entry/return */
#define DTRACEOPT_QUIET         12      /* only output explicitly traced data */
#define DTRACEOPT_STACKFRAMES   13      /* number of stack frames */
#define DTRACEOPT_USTACKFRAMES  14      /* number of user stack frames */
#define DTRACEOPT_AGGRATE       15      /* aggregation snapshot rate */
#define DTRACEOPT_SWITCHRATE    16      /* buffer switching rate */
#define DTRACEOPT_STATUSRATE    17      /* status rate */
#define DTRACEOPT_DESTRUCTIVE   18      /* destructive actions allowed */
#define DTRACEOPT_STACKINDENT   19      /* output indent for stack traces */
#define DTRACEOPT_RAWBYTES      20      /* always print bytes in raw form */
#define DTRACEOPT_JSTACKFRAMES  21      /* number of jstack() frames */
#define DTRACEOPT_JSTACKSTRSIZE 22      /* size of jstack() string table */
#define DTRACEOPT_AGGSORTKEY    23      /* sort aggregations by key */
#define DTRACEOPT_AGGSORTREV    24      /* reverse-sort aggregations */
#define DTRACEOPT_AGGSORTPOS    25      /* agg. position to sort on */
#define DTRACEOPT_AGGSORTKEYPOS 26      /* agg. key position to sort on */
#define DTRACEOPT_TEMPORAL      27      /* temporally ordered output */
#define DTRACEOPT_MAX           28      /* number of options */

#define DTRACEOPT_UNSET         (dtrace_optval_t)-2     /* unset option */

#define DTRACEOPT_BUFPOLICY_RING        0       /* ring buffer */
#define DTRACEOPT_BUFPOLICY_FILL        1       /* fill buffer, then stop */
#define DTRACEOPT_BUFPOLICY_SWITCH      2       /* switch buffers */

#define DTRACEOPT_BUFRESIZE_AUTO        0       /* automatic resizing */
```

```
#define DTRACEOPT_BUFRESIZE_MANUAL      1       /* manual resizing */

/*
 * DTrace Buffer Interface
 *
 * In order to get a snapshot of the principal or aggregation buffer,
 * user-level passes a buffer description to the kernel with the dtrace_bufdesc
 * structure.  This describes which CPU user-level is interested in, and
 * where user-level wishes the kernel to snapshot the buffer to (the
 * dtbd_data field).  The kernel uses the same structure to pass back some
 * information regarding the buffer:  the size of data actually copied out, the
 * number of drops, the number of errors, the offset of the oldest record,
 * and the time of the snapshot.
 *
 * If the buffer policy is a "switch" policy, taking a snapshot of the
 * principal buffer has the additional effect of switching the active and
 * inactive buffers.  Taking a snapshot of the aggregation buffer _always_ has
 * the additional effect of switching the active and inactive buffers.
 */
typedef struct dtrace_bufdesc {
        uint64_t dtbd_size;                     /* size of buffer */
        uint32_t dtbd_cpu;                      /* CPU or DTRACE_CPUALL */
        uint32_t dtbd_errors;                   /* number of errors */
        uint64_t dtbd_drops;                    /* number of drops */
        DTRACE_PTR(char, dtbd_data);            /* data */
        uint64_t dtbd_oldest;                   /* offset of oldest record */
        uint64_t dtbd_timestamp;                /* hrtime of snapshot */
} dtrace_bufdesc_t;

/*
 * Each record in the buffer (dtbd_data) begins with a header that includes
 * the epid and a timestamp.  The timestamp is split into two 4-byte parts
 * so that we do not require 8-byte alignment.
 */
typedef struct dtrace_rechdr {
        dtrace_epid_t dtrh_epid;                /* enabled probe id */
        uint32_t dtrh_timestamp_hi;             /* high bits of hrtime_t */
        uint32_t dtrh_timestamp_lo;             /* low bits of hrtime_t */
} dtrace_rechdr_t;

#define DTRACE_RECORD_LOAD_TIMESTAMP(dtrh)                      \
        ((dtrh)->dtrh_timestamp_lo +                            \
        ((uint64_t)(dtrh)->dtrh_timestamp_hi << 32))

#define DTRACE_RECORD_STORE_TIMESTAMP(dtrh, hrtime) {           \
        (dtrh)->dtrh_timestamp_lo = (uint32_t)hrtime;           \
        (dtrh)->dtrh_timestamp_hi = hrtime >> 32;               \
}

/*
 * DTrace Status
 *
 * The status of DTrace is relayed via the dtrace_status structure.  This
 * structure contains members to count drops other than the capacity drops
 * available via the buffer interface (see above).  This consists of dynamic
 * drops (including capacity dynamic drops, rinsing drops and dirty drops), and
 * speculative drops (including capacity speculative drops, drops due to busy
 * speculative buffers and drops due to unavailable speculative buffers).
 * Additionally, the status structure contains a field to indicate the number
 * of "fill"-policy buffers have been filled and a boolean field to indicate
 * that exit() has been called.  If the dtst_exiting field is non-zero, no
 * further data will be generated until tracing is stopped (at which time any
 * enablings of the END action will be processed); if user-level sees that
 * this field is non-zero, tracing should be stopped as soon as possible.
 */
typedef struct dtrace_status {
```



```
        uint64_t dtst_dyndrops;                 /* dynamic drops */
        uint64_t dtst_dyndrops_rinsing;         /* dyn drops due to rinsing */
        uint64_t dtst_dyndrops_dirty;           /* dyn drops due to dirty */
        uint64_t dtst_specdrops;                /* speculative drops */
        uint64_t dtst_specdrops_busy;           /* spec drops due to busy */
        uint64_t dtst_specdrops_unavail;        /* spec drops due to unavail */
        uint64_t dtst_errors;                   /* total errors */
        uint64_t dtst_filled;                   /* number of filled bufs */
        uint64_t dtst_stkstroverflows;          /* stack string tab overflows */
        uint64_t dtst_dblerrors;                /* errors in ERROR probes */
        char dtst_killed;                       /* non-zero if killed */
        char dtst_exiting;                      /* non-zero if exit() called */
        char dtst_pad[6];                       /* pad out to 64-bit align */
} dtrace_status_t;

/*
 * DTrace Configuration
 *
 * User-level may need to understand some elements of the kernel DTrace
 * configuration in order to generate correct DIF.  This information is
 * conveyed via the dtrace_conf structure.
 */
typedef struct dtrace_conf {
        uint_t dtc_difversion;                  /* supported DIF version */
        uint_t dtc_difintregs;                  /* # of DIF integer registers */
        uint_t dtc_diftupregs;                  /* # of DIF tuple registers */
        uint_t dtc_ctfmodel;                    /* CTF data model */
        uint_t dtc_pad[8];                      /* reserved for future use */
} dtrace_conf_t;

/*
 * DTrace Faults
 *
 * The constants below DTRACEFLT_LIBRARY indicate probe processing faults;
 * constants at or above DTRACEFLT_LIBRARY indicate faults in probe
 * postprocessing at user-level.  Probe processing faults induce an ERROR
 * probe and are replicated in unistd.d to allow users' ERROR probes to decode
 * the error condition using thse symbolic labels.
 */
#define DTRACEFLT_UNKNOWN               0       /* Unknown fault */
#define DTRACEFLT_BADADDR               1       /* Bad address */
#define DTRACEFLT_BADALIGN              2       /* Bad alignment */
#define DTRACEFLT_ILLOP                 3       /* Illegal operation */
#define DTRACEFLT_DIVZERO               4       /* Divide-by-zero */
#define DTRACEFLT_NOSCRATCH             5       /* Out of scratch space */
#define DTRACEFLT_KPRIV                 6       /* Illegal kernel access */
#define DTRACEFLT_UPRIV                 7       /* Illegal user access */
#define DTRACEFLT_TUPOFLOW              8       /* Tuple stack overflow */
#define DTRACEFLT_BADSTACK              9       /* Bad stack */

#define DTRACEFLT_LIBRARY               1000    /* Library-level fault */

/*
 * DTrace Argument Types
 *
 * Because it would waste both space and time, argument types do not reside
 * with the probe.  In order to determine argument types for args[X]
 * variables, the D compiler queries for argument types on a probe-by-probe
 * basis.  (This optimizes for the common case that arguments are either not
 * used or used in an untyped fashion.)  Typed arguments are specified with a
 * string of the type name in the dtragd_native member of the argument
 * description structure.  Typed arguments may be further translated to types
 * of greater stability; the provider indicates such a translated argument by
 * filling in the dtargd_xlate member with the string of the translated type.
 * Finally, the provider may indicate which argument value a given argument
 * maps to by setting the dtargd_mapping member -- allowing a single argument
```

```
 * to map to multiple args[X] variables.
 */
typedef struct dtrace_argdesc {
        dtrace_id_t dtargd_id;                  /* probe identifier */
        int dtargd_ndx;                         /* arg number (-1 iff none) */
        int dtargd_mapping;                     /* value mapping */
        char dtargd_native[DTRACE_ARGTYPELEN];  /* native type name */
        char dtargd_xlate[DTRACE_ARGTYPELEN];   /* translated type name */
} dtrace_argdesc_t;

/*
 * DTrace Stability Attributes
 *
 * Each DTrace provider advertises the name and data stability of each of its
 * probe description components, as well as its architectural dependencies.
 * The D compiler can query the provider attributes (dtrace_pattr_t below) in
 * order to compute the properties of an input program and report them.
 */
typedef uint8_t dtrace_stability_t;     /* stability code (see attributes(5)) */
typedef uint8_t dtrace_class_t;         /* architectural dependency class */

#define DTRACE_STABILITY_INTERNAL       0       /* private to DTrace itself */
#define DTRACE_STABILITY_PRIVATE        1       /* private to Sun (see docs) */
#define DTRACE_STABILITY_OBSOLETE       2       /* scheduled for removal */
#define DTRACE_STABILITY_EXTERNAL       3       /* not controlled by Sun */
#define DTRACE_STABILITY_UNSTABLE       4       /* new or rapidly changing */
#define DTRACE_STABILITY_EVOLVING       5       /* less rapidly changing */
#define DTRACE_STABILITY_STABLE         6       /* mature interface from Sun */
#define DTRACE_STABILITY_STANDARD       7       /* industry standard */
#define DTRACE_STABILITY_MAX            7       /* maximum valid stability */

#define DTRACE_CLASS_UNKNOWN    0       /* unknown architectural dependency */
#define DTRACE_CLASS_CPU        1       /* CPU-module-specific */
#define DTRACE_CLASS_PLATFORM   2       /* platform-specific (uname -i) */
#define DTRACE_CLASS_GROUP      3       /* hardware-group-specific (uname -m) */
#define DTRACE_CLASS_ISA        4       /* ISA-specific (uname -p) */
#define DTRACE_CLASS_COMMON     5       /* common to all systems */
#define DTRACE_CLASS_MAX        5       /* maximum valid class */

#define DTRACE_PRIV_NONE        0x0000
#define DTRACE_PRIV_KERNEL      0x0001
#define DTRACE_PRIV_USER        0x0002
#define DTRACE_PRIV_PROC        0x0004
#define DTRACE_PRIV_OWNER       0x0008
#define DTRACE_PRIV_ZONEOWNER   0x0010

#define DTRACE_PRIV_ALL \
        (DTRACE_PRIV_KERNEL | DTRACE_PRIV_USER | \
        DTRACE_PRIV_PROC | DTRACE_PRIV_OWNER | DTRACE_PRIV_ZONEOWNER)

typedef struct dtrace_ppriv {
        uint32_t dtpp_flags;                    /* privilege flags */
        uid_t dtpp_uid;                         /* user ID */
        zoneid_t dtpp_zoneid;                   /* zone ID */
} dtrace_ppriv_t;

typedef struct dtrace_attribute {
        dtrace_stability_t dtat_name;           /* entity name stability */
        dtrace_stability_t dtat_data;           /* entity data stability */
        dtrace_class_t dtat_class;              /* entity data dependency */
} dtrace_attribute_t;

typedef struct dtrace_pattr {
        dtrace_attribute_t dtpa_provider;       /* provider attributes */
        dtrace_attribute_t dtpa_mod;            /* module attributes */
        dtrace_attribute_t dtpa_func;           /* function attributes */
```



```
        dtrace_attribute_t dtpa_name;           /* name attributes */
        dtrace_attribute_t dtpa_args;           /* args[] attributes */
} dtrace_pattr_t;

typedef struct dtrace_providerdesc {
        char dtvd_name[DTRACE_PROVNAMELEN];     /* provider name */
        dtrace_pattr_t dtvd_attr;               /* stability attributes */
        dtrace_ppriv_t dtvd_priv;               /* privileges required */
} dtrace_providerdesc_t;

/*
 * DTrace Pseudodevice Interface
 *
 * DTrace is controlled through ioctl(2)'s to the in-kernel dtrace:dtrace
 * pseudodevice driver.  These ioctls comprise the user-kernel interface to
 * DTrace.
 */
#define DTRACEIOC               (('d' << 24) | ('t' << 16) | ('r' << 8))
#define DTRACEIOC_PROVIDER      (DTRACEIOC | 1)         /* provider query */
#define DTRACEIOC_PROBES        (DTRACEIOC | 2)         /* probe query */
#define DTRACEIOC_BUFSNAP       (DTRACEIOC | 4)         /* snapshot buffer */
#define DTRACEIOC_PROBEMATCH    (DTRACEIOC | 5)         /* match probes */
#define DTRACEIOC_ENABLE        (DTRACEIOC | 6)         /* enable probes */
#define DTRACEIOC_AGGSNAP       (DTRACEIOC | 7)         /* snapshot agg. */
#define DTRACEIOC_EPROBE        (DTRACEIOC | 8)         /* get eprobe desc. */
#define DTRACEIOC_PROBEARG      (DTRACEIOC | 9)         /* get probe arg */
#define DTRACEIOC_CONF          (DTRACEIOC | 10)        /* get config. */
#define DTRACEIOC_STATUS        (DTRACEIOC | 11)        /* get status */
#define DTRACEIOC_GO            (DTRACEIOC | 12)        /* start tracing */
#define DTRACEIOC_STOP          (DTRACEIOC | 13)        /* stop tracing */
#define DTRACEIOC_AGGDESC       (DTRACEIOC | 15)        /* get agg. desc. */
#define DTRACEIOC_FORMAT        (DTRACEIOC | 16)        /* get format str */
#define DTRACEIOC_DOFGET        (DTRACEIOC | 17)        /* get DOF */
#define DTRACEIOC_REPLICATE     (DTRACEIOC | 18)        /* replicate enab */

/*
 * DTrace Helpers
 *
 * In general, DTrace establishes probes in processes and takes actions on
 * processes without knowing their specific user-level structures.  Instead of
 * existing in the framework, process-specific knowledge is contained by the
 * enabling D program -- which can apply process-specific knowledge by making
 * appropriate use of DTrace primitives like copyin() and copyinstr() to
 * operate on user-level data.  However, there may exist some specific probes
 * of particular semantic relevance that the application developer may wish to
 * explicitly export.  For example, an application may wish to export a probe
 * at the point that it begins and ends certain well-defined transactions.  In
 * addition to providing probes, programs may wish to offer assistance for
 * certain actions.  For example, in highly dynamic environments (e.g., Java),
 * it may be difficult to obtain a stack trace in terms of meaningful symbol
 * names (the translation from instruction addresses to corresponding symbol
 * names may only be possible in situ); these environments may wish to define
 * a series of actions to be applied in situ to obtain a meaningful stack
 * trace.
 *
 * These two mechanisms -- user-level statically defined tracing and assisting
 * DTrace actions -- are provided via DTrace _helpers_.  Helpers are specified
 * via DOF, but unlike enabling DOF, helper DOF may contain definitions of
 * providers, probes and their arguments.  If a helper wishes to provide
 * action assistance, probe descriptions and corresponding DIF actions may be
 * specified in the helper DOF.  For such helper actions, however, the probe
 * description describes the specific helper:  all DTrace helpers have the
 * provider name "dtrace" and the module name "helper", and the name of the
 * helper is contained in the function name (for example, the ustack() helper
 * is named "ustack").  Any helper-specific name may be contained in the name
 * (for example, if a helper were to have a constructor, it might be named
```

```
 * "dtrace:helper:<helper>:init").  Helper actions are only called when the
 * action that they are helping is taken.  Helper actions may only return DIF
 * expressions, and may only call the following subroutines:
 *
 *    alloca()      <= Allocates memory out of the consumer's scratch space
 *    bcopy()       <= Copies memory to scratch space
 *    copyin()      <= Copies memory from user-level into consumer's scratch
 *    copyinto()    <= Copies memory into a specific location in scratch
 *    copyinstr()   <= Copies a string into a specific location in scratch
 *
 * Helper actions may only access the following built-in variables:
 *
 *    curthread     <= Current kthread_t pointer
 *    tid           <= Current thread identifier
 *    pid           <= Current process identifier
 *    ppid          <= Parent process identifier
 *    uid           <= Current user ID
 *    gid           <= Current group ID
 *    execname      <= Current executable name
 *    zonename      <= Current zone name
 *
 * Helper actions may not manipulate or allocate dynamic variables, but they
 * may have clause-local and statically-allocated global variables.  The
 * helper action variable state is specific to the helper action -- variables
 * used by the helper action may not be accessed outside of the helper
 * action, and the helper action may not access variables that like outside
 * of it.  Helper actions may not load from kernel memory at-large; they are
 * restricting to loading current user state (via copyin() and variants) and
 * scratch space.  As with probe enablings, helper actions are executed in
 * program order.  The result of the helper action is the result of the last
 * executing helper expression.
 *
 * Helpers -- composed of either providers/probes or probes/actions (or both)
 * -- are added by opening the "helper" minor node, and issuing an ioctl(2)
 * (DTRACEHIOC_ADDDOF) that specifies the dof_helper_t structure. This
 * encapsulates the name and base address of the user-level library or
 * executable publishing the helpers and probes as well as the DOF that
 * contains the definitions of those helpers and probes.
 *
 * The DTRACEHIOC_ADD and DTRACEHIOC_REMOVE are left in place for legacy
 * helpers and should no longer be used.  No other ioctls are valid on the
 * helper minor node.
 */
#define DTRACEHIOC              ('d' << 24) | ('t' << 16) | ('h' << 8))
#define DTRACEHIOC_ADD          (DTRACEHIOC | 1)        /* add helper */
#define DTRACEHIOC_REMOVE       (DTRACEHIOC | 2)        /* remove helper */
#define DTRACEHIOC_ADDDOF       (DTRACEHIOC | 3)        /* add helper DOF */

typedef struct dof_helper {
        char dofhp_mod[DTRACE_MODNAMELEN];      /* executable or library name */
        uint64_t dofhp_addr;                    /* base address of object */
        uint64_t dofhp_dof;                     /* address of helper DOF */
} dof_helper_t;

#define DTRACEMNR_DTRACE        "dtrace"        /* node for DTrace ops */
#define DTRACEMNR_HELPER        "helper"        /* node for helpers */
#define DTRACEMNRN_DTRACE       0               /* minor for DTrace ops */
#define DTRACEMNRN_HELPER       1               /* minor for helpers */
#define DTRACEMNRN_CLONE        2               /* first clone minor */

#ifdef _KERNEL

/*
 * DTrace Provider API
 *
 * The following functions are implemented by the DTrace framework and are
```



```
 * used to implement separate in-kernel DTrace providers.  Common functions
 * are provided in uts/common/os/dtrace.c.  ISA-dependent subroutines are
 * defined in uts/<isa>/dtrace/dtrace_asm.s or uts/<isa>/dtrace/dtrace_isa.c.
 *
 * The provider API has two halves:  the API that the providers consume from
 * DTrace, and the API that providers make available to DTrace.
 *
 * 1 Framework-to-Provider API
 *
 * 1.1  Overview
 *
 * The Framework-to-Provider API is represented by the dtrace_pops structure
 * that the provider passes to the framework when registering itself.  This
 * structure consists of the following members:
 *
 *   dtps_provide()          <-- Provide all probes, all modules
 *   dtps_provide_module()   <-- Provide all probes in specified module
 *   dtps_enable()           <-- Enable specified probe
 *   dtps_disable()          <-- Disable specified probe
 *   dtps_suspend()          <-- Suspend specified probe
 *   dtps_resume()           <-- Resume specified probe
 *   dtps_getargdesc()       <-- Get the argument description for args[X]
 *   dtps_getargval()        <-- Get the value for an argX or args[X] variable
 *   dtps_mode()             <-- Return the mode of the fired probe
 *   dtps_destroy()          <-- Destroy all state associated with this probe
 *
 * 1.2  void dtps_provide(void *arg, const dtrace_probedesc_t *spec)
 *
 * 1.2.1  Overview
 *
 *   Called to indicate that the provider should provide all probes.  If the
 *   specified description is non-NULL, dtps_provide() is being called because
 *   no probe matched a specified probe -- if the provider has the ability to
 *   create custom probes, it may wish to create a probe that matches the
 *   specified description.
 *
 * 1.2.2  Arguments and notes
 *
 *   The first argument is the cookie as passed to dtrace_register().  The
 *   second argument is a pointer to a probe description that the provider may
 *   wish to consider when creating custom probes.  The provider is expected to
 *   call back into the DTrace framework via dtrace_probe_create() to create
 *   any necessary probes.  dtps_provide() may be called even if the provider
 *   has made available all probes; the provider should check the return value
 *   of dtrace_probe_create() to handle this case.  Note that the provider need
 *   not implement both dtps_provide() and dtps_provide_module(); see
 *   "Arguments and Notes" for dtrace_register(), below.
 *
 * 1.2.3  Return value
 *
 *   None.
 *
 * 1.2.4  Caller's context
 *
 *   dtps_provide() is typically called from open() or ioctl() context, but may
 *   be called from other contexts as well.  The DTrace framework is locked in
 *   such a way that providers may not register or unregister.  This means that
 *   the provider may not call any DTrace API that affects its registration with
 *   the framework, including dtrace_register(), dtrace_unregister(),
 *   dtrace_invalidate(), and dtrace_condense().  However, the context is such
 *   that the provider may (and indeed, is expected to) call probe-related
 *   DTrace routines, including dtrace_probe_create(), dtrace_probe_lookup(),
 *   and dtrace_probe_arg().
 *
 * 1.3  void dtps_provide_module(void *arg, struct modctl *mp)
 *
```

```
1432  * 1.3.1  Overview
1433  *
1434  *   Called to indicate that the provider should provide all probes in the
1435  *   specified module.
1436  *
1437  * 1.3.2  Arguments and notes
1438  *
1439  *   The first argument is the cookie as passed to dtrace_register().  The
1440  *   second argument is a pointer to a modctl structure that indicates the
1441  *   module for which probes should be created.
1442  *
1443  * 1.3.3  Return value
1444  *
1445  *   None.
1446  *
1447  * 1.3.4  Caller's context
1448  *
1449  *   dtps_provide_module() may be called from open() or ioctl() context, but
1450  *   may also be called from a module loading context.  mod_lock is held, and
1451  *   the DTrace framework is locked in such a way that providers may not
1452  *   register or unregister.  This means that the provider may not call any
1453  *   DTrace API that affects its registration with the framework, including
1454  *   dtrace_register(), dtrace_unregister(), dtrace_invalidate(), and
1455  *   dtrace_condense().  However, the context is such that the provider may (and
1456  *   indeed, is expected to) call probe-related DTrace routines, including
1457  *   dtrace_probe_create(), dtrace_probe_lookup(), and dtrace_probe_arg().  Note
1458  *   that the provider need not implement both dtps_provide() and
1459  *   dtps_provide_module(); see "Arguments and Notes" for dtrace_register(),
1460  *   below.
1461  *
1462  * 1.4  int dtps_enable(void *arg, dtrace_id_t id, void *parg)
1463  *
1464  * 1.4.1  Overview
1465  *
1466  *   Called to enable the specified probe.
1467  *
1468  * 1.4.2  Arguments and notes
1469  *
1470  *   The first argument is the cookie as passed to dtrace_register().  The
1471  *   second argument is the identifier of the probe to be enabled.  The third
1472  *   argument is the probe argument as passed to dtrace_probe_create().
1473  *   dtps_enable() will be called when a probe transitions from not being
1474  *   enabled at all to having one or more ECB.  The number of ECBs associated
1475  *   with the probe may change without subsequent calls into the provider.
1476  *   When the number of ECBs drops to zero, the provider will be explicitly
1477  *   told to disable the probe via dtps_disable().  dtrace_probe() should never
1478  *   be called for a probe identifier that hasn't been explicitly enabled via
1479  *   dtps_enable().
1480  *
1481  * 1.4.3  Return value
1482  *
1483  *   On success, dtps_enable() should return 0. On failure, -1 should be
1484  *   returned.
1485  *
1486  * 1.4.4  Caller's context
1487  *
1488  *   The DTrace framework is locked in such a way that it may not be called
1489  *   back into at all.  cpu_lock is held.  mod_lock is not held and may not
1490  *   be acquired.
1491  *
1492  * 1.5  void dtps_disable(void *arg, dtrace_id_t id, void *parg)
1493  *
1494  * 1.5.1  Overview
1495  *
1496  *   Called to disable the specified probe.
1497  *
```



```
1498  * 1.5.2  Arguments and notes
1499  *
1500  *   The first argument is the cookie as passed to dtrace_register().  The
1501  *   second argument is the identifier of the probe to be disabled.  The third
1502  *   argument is the probe argument as passed to dtrace_probe_create().
1503  *   dtps_disable() will be called when a probe transitions from being enabled
1504  *   to having zero ECBs.  dtrace_probe() should never be called for a probe
1505  *   identifier that has been explicitly enabled via dtps_disable().
1506  *
1507  * 1.5.3  Return value
1508  *
1509  *   None.
1510  *
1511  * 1.5.4  Caller's context
1512  *
1513  *   The DTrace framework is locked in such a way that it may not be called
1514  *   back into at all.  cpu_lock is held.  mod_lock is not held and may not
1515  *   be acquired.
1516  *
1517  * 1.6  void dtps_suspend(void *arg, dtrace_id_t id, void *parg)
1518  *
1519  * 1.6.1  Overview
1520  *
1521  *   Called to suspend the specified enabled probe.  This entry point is for
1522  *   providers that may need to suspend some or all of their probes when CPUs
1523  *   are being powered on or when the boot monitor is being entered for a
1524  *   prolonged period of time.
1525  *
1526  * 1.6.2  Arguments and notes
1527  *
1528  *   The first argument is the cookie as passed to dtrace_register().  The
1529  *   second argument is the identifier of the probe to be suspended.  The
1530  *   third argument is the probe argument as passed to dtrace_probe_create().
1531  *   dtps_suspend will only be called on an enabled probe.  Providers that
1532  *   provide a dtps_suspend entry point will want to take roughly the action
1533  *   that it takes for dtps_disable.
1534  *
1535  * 1.6.3  Return value
1536  *
1537  *   None.
1538  *
1539  * 1.6.4  Caller's context
1540  *
1541  *   Interrupts are disabled.  The DTrace framework is in a state such that the
1542  *   specified probe cannot be disabled or destroyed for the duration of
1543  *   dtps_suspend().  As interrupts are disabled, the provider is afforded
1544  *   little latitude; the provider is expected to do no more than a store to
1545  *   memory.
1546  *
1547  * 1.7  void dtps_resume(void *arg, dtrace_id_t id, void *parg)
1548  *
1549  * 1.7.1  Overview
1550  *
1551  *   Called to resume the specified enabled probe.  This entry point is for
1552  *   providers that may need to resume some or all of their probes after the
1553  *   completion of an event that induced a call to dtps_suspend().
1554  *
1555  * 1.7.2  Arguments and notes
1556  *
1557  *   The first argument is the cookie as passed to dtrace_register().  The
1558  *   second argument is the identifier of the probe to be resumed.  The
1559  *   third argument is the probe argument as passed to dtrace_probe_create().
1560  *   dtps_resume will only be called on an enabled probe.  Providers that
1561  *   provide a dtps_resume entry point will want to take roughly the action
1562  *   that it takes for dtps_enable.
1563  *
```

```
1564  * 1.7.3  Return value
1565  *
1566  *   None.
1567  *
1568  * 1.7.4  Caller's context
1569  *
1570  *   Interrupts are disabled.  The DTrace framework is in a state such that the
1571  *   specified probe cannot be disabled or destroyed for the duration of
1572  *   dtps_resume().  As interrupts are disabled, the provider is afforded
1573  *   little latitude; the provider is expected to do no more than a store to
1574  *   memory.
1575  *
1576  * 1.8  void dtps_getargdesc(void *arg, dtrace_id_t id, void *parg,
1577  *           dtrace_argdesc_t *desc)
1578  *
1579  * 1.8.1  Overview
1580  *
1581  *   Called to retrieve the argument description for an args[X] variable.
1582  *
1583  * 1.8.2  Arguments and notes
1584  *
1585  *   The first argument is the cookie as passed to dtrace_register(). The
1586  *   second argument is the identifier of the current probe. The third
1587  *   argument is the probe argument as passed to dtrace_probe_create(). The
1588  *   fourth argument is a pointer to the argument description.  This
1589  *   description is both an input and output parameter:  it contains the
1590  *   index of the desired argument in the dtargd_ndx field, and expects
1591  *   the other fields to be filled in upon return.  If there is no argument
1592  *   corresponding to the specified index, the dtargd_ndx field should be set
1593  *   to DTRACE_ARGNONE.
1594  *
1595  * 1.8.3  Return value
1596  *
1597  *   None.  The dtargd_ndx, dtargd_native, dtargd_xlate and dtargd_mapping
1598  *   members of the dtrace_argdesc_t structure are all output values.
1599  *
1600  * 1.8.4  Caller's context
1601  *
1602  *   dtps_getargdesc() is called from ioctl() context. mod_lock is held, and
1603  *   the DTrace framework is locked in such a way that providers may not
1604  *   register or unregister.  This means that the provider may not call any
1605  *   DTrace API that affects its registration with the framework, including
1606  *   dtrace_register(), dtrace_unregister(), dtrace_invalidate(), and
1607  *   dtrace_condense().
1608  *
1609  * 1.9  uint64_t dtps_getargval(void *arg, dtrace_id_t id, void *parg,
1610  *              int argno, int aframes)
1611  *
1612  * 1.9.1  Overview
1613  *
1614  *   Called to retrieve a value for an argX or args[X] variable.
1615  *
1616  * 1.9.2  Arguments and notes
1617  *
1618  *   The first argument is the cookie as passed to dtrace_register(). The
1619  *   second argument is the identifier of the current probe. The third
1620  *   argument is the probe argument as passed to dtrace_probe_create(). The
1621  *   fourth argument is the number of the argument (the X in the example in
1622  *   1.9.1). The fifth argument is the number of stack frames that were used
1623  *   to get from the actual place in the code that fired the probe to
1624  *   dtrace_probe() itself, the so-called artificial frames. This argument may
1625  *   be used to descend an appropriate number of frames to find the correct
1626  *   values. If this entry point is left NULL, the dtrace_getarg() built-in
1627  *   function is used.
1628  *
1629  * 1.9.3  Return value
```



```
1630  *
1631  *   The value of the argument.
1632  *
1633  * 1.9.4  Caller's context
1634  *
1635  *   This is called from within dtrace_probe() meaning that interrupts
1636  *   are disabled. No locks should be taken within this entry point.
1637  *
1638  * 1.10  int dtps_mode(void *arg, dtrace_id_t id, void *parg)
1639  *
1640  * 1.10.1  Overview
1641  *
1642  *   Called to determine the mode of a fired probe.
1643  *
1644  * 1.10.2  Arguments and notes
1645  *
1646  *   The first argument is the cookie as passed to dtrace_register(). The
1647  *   second argument is the identifier of the current probe.  The third
1648  *   argument is the probe argument as passed to dtrace_probe_create().  This
1649  *   entry point must not be left NULL for providers whose probes allow for
1650  *   mixed mode tracing, that is to say those unanchored probes that can fire
1651  *   during kernel- or user-mode execution.
1652  *
1653  * 1.10.3  Return value
1654  *
1655  *   A bitwise OR that encapsulates both the mode (either DTRACE_MODE_KERNEL
1656  *   or DTRACE_MODE_USER) and the policy when the privilege of the enabling
1657  *   is insufficient for that mode (a combination of DTRACE_MODE_NOPRIV_DROP,
1658  *   DTRACE_MODE_NOPRIV_RESTRICT, and DTRACE_MODE_LIMITEDPRIV_RESTRICT).  If
1659  *   DTRACE_MODE_NOPRIV_DROP bit is set, insufficient privilege will result
1660  *   in the probe firing being silently ignored for the enabling; if the
1661  *   DTRACE_NODE_NOPRIV_RESTRICT bit is set, insufficient privilege will not
1662  *   prevent probe processing for the enabling, but restrictions will be in
1663  *   place that induce a UPRIV fault upon attempt to examine probe arguments
1664  *   or current process state.  If the DTRACE_MODE_LIMITEDPRIV_RESTRICT bit
1665  *   is set, similar restrictions will be placed upon operation if the
1666  *   privilege is sufficient to process the enabling, but does not otherwise
1667  *   entitle the enabling to all zones.  The DTRACE_MODE_NOPRIV_DROP and
1668  *   DTRACE_MODE_NOPRIV_RESTRICT are mutually exclusive (and one of these
1669  *   two policies must be specified), but either may be combined (or not)
1670  *   with DTRACE_MODE_LIMITEDPRIV_RESTRICT.
1671  *
1672  * 1.10.4  Caller's context
1673  *
1674  *   This is called from within dtrace_probe() meaning that interrupts
1675  *   are disabled. No locks should be taken within this entry point.
1676  *
1677  * 1.11 void dtps_destroy(void *arg, dtrace_id_t id, void *parg)
1678  *
1679  * 1.11.1 Overview
1680  *
1681  *   Called to destroy the specified probe.
1682  *
1683  * 1.11.2 Arguments and notes
1684  *
1685  *   The first argument is the cookie as passed to dtrace_register().  The
1686  *   second argument is the identifier of the probe to be destroyed.  The third
1687  *   argument is the probe argument as passed to dtrace_probe_create().  The
1688  *   provider should free all state associated with the probe.  The framework
1689  *   guarantees that dtps_destroy() is only called for probes that have either
1690  *   been disabled via dtps_disable() or were never enabled via dtps_enable().
1691  *   Once dtps_disable() has been called for a probe, no further call will be
1692  *   made specifying the probe.
1693  *
1694  * 1.11.3 Return value
1695  *
```

```
1696  *   None.
1697  *
1698  * 1.11.4 Caller's context
1699  *
1700  *   The DTrace framework is locked in such a way that it may not be called
1701  *   back into at all.  mod_lock is held.  cpu_lock is not held, and may not be
1702  *   acquired.
1703  *
1704  *
1705  * 2 Provider-to-Framework API
1706  *
1707  * 2.1  Overview
1708  *
1709  * The Provider-to-Framework API provides the mechanism for the provider to
1710  * register itself with the DTrace framework, to create probes, to lookup
1711  * probes and (most importantly) to fire probes.  The Provider-to-Framework
1712  * consists of:
1713  *
1714  *   dtrace_register()       <-- Register a provider with the DTrace framework
1715  *   dtrace_unregister()     <-- Remove a provider's DTrace registration
1716  *   dtrace_invalidate()     <-- Invalidate the specified provider
1717  *   dtrace_condense()       <-- Remove a provider's unenabled probes
1718  *   dtrace_attached()       <-- Indicates whether or not DTrace has attached
1719  *   dtrace_probe_create()   <-- Create a DTrace probe
1720  *   dtrace_probe_lookup()   <-- Lookup a DTrace probe based on its name
1721  *   dtrace_probe_arg()      <-- Return the probe argument for a specific probe
1722  *   dtrace_probe()          <-- Fire the specified probe
1723  *
1724  * 2.2  int dtrace_register(const char *name, const dtrace_pattr_t *pap,
1725  *         uint32_t priv, cred_t *cr, const dtrace_pops_t *pops, void *arg,
1726  *         dtrace_provider_id_t *idp)
1727  *
1728  * 2.2.1  Overview
1729  *
1730  *   dtrace_register() registers the calling provider with the DTrace
1731  *   framework.  It should generally be called by DTrace providers in their
1732  *   attach(9E) entry point.
1733  *
1734  * 2.2.2  Arguments and Notes
1735  *
1736  *   The first argument is the name of the provider.  The second argument is a
1737  *   pointer to the stability attributes for the provider.  The third argument
1738  *   is the privilege flags for the provider, and must be some combination of:
1739  *
1740  *     DTRACE_PRIV_NONE     <= All users may enable probes from this provider
1741  *
1742  *     DTRACE_PRIV_PROC     <= Any user with privilege of PRIV_DTRACE_PROC may
1743  *                             enable probes from this provider
1744  *
1745  *     DTRACE_PRIV_USER     <= Any user with privilege of PRIV_DTRACE_USER may
1746  *                             enable probes from this provider
1747  *
1748  *     DTRACE_PRIV_KERNEL   <= Any user with privilege of PRIV_DTRACE_KERNEL
1749  *                             may enable probes from this provider
1750  *
1751  *     DTRACE_PRIV_OWNER    <= This flag places an additional constraint on
1752  *                             the privilege requirements above. These probes
1753  *                             require either (a) a user ID matching the user
1754  *                             ID of the cred passed in the fourth argument
1755  *                             or (b) the PRIV_PROC_OWNER privilege.
1756  *
1757  *     DTRACE_PRIV_ZONEOWNER<= This flag places an additional constraint on
1758  *                             the privilege requirements above. These probes
1759  *                             require either (a) a zone ID matching the zone
1760  *                             ID of the cred passed in the fourth argument
1761  *                             or (b) the PRIV_PROC_ZONE privilege.
```



```
1762  *
1763  *   Note that these flags designate the _visibility_ of the probes, not
1764  *   the conditions under which they may or may not fire.
1765  *
1766  *   The fourth argument is the credential that is associated with the
1767  *   provider.  This argument should be NULL if the privilege flags don't
1768  *   include DTRACE_PRIV_OWNER or DTRACE_PRIV_ZONEOWNER.  If non-NULL, the
1769  *   framework stashes the uid and zoneid represented by this credential
1770  *   for use at probe-time, in implicit predicates.  These limit visibility
1771  *   of the probes to users and/or zones which have sufficient privilege to
1772  *   access them.
1773  *
1774  *   The fifth argument is a DTrace provider operations vector, which provides
1775  *   the implementation for the Framework-to-Provider API.  (See Section 1,
1776  *   above.)  This must be non-NULL, and each member must be non-NULL.  The
1777  *   exceptions to this are (1) the dtps_provide() and dtps_provide_module()
1778  *   members (if the provider so desires, _one_ of these members may be left
1779  *   NULL -- denoting that the provider only implements the other) and (2)
1780  *   the dtps_suspend() and dtps_resume() members, which must either both be
1781  *   NULL or both be non-NULL.
1782  *
1783  *   The sixth argument is a cookie to be specified as the first argument for
1784  *   each function in the Framework-to-Provider API.  This argument may have
1785  *   any value.
1786  *
1787  *   The final argument is a pointer to dtrace_provider_id_t.  If
1788  *   dtrace_register() successfully completes, the provider identifier will be
1789  *   stored in the memory pointed to be this argument.  This argument must be
1790  *   non-NULL.
1791  *
1792  * 2.2.3  Return value
1793  *
1794  *   On success, dtrace_register() returns 0 and stores the new provider's
1795  *   identifier into the memory pointed to by the idp argument.  On failure,
1796  *   dtrace_register() returns an errno:
1797  *
1798  *     EINVAL   The arguments passed to dtrace_register() were somehow invalid.
1799  *              This may because a parameter that must be non-NULL was NULL,
1800  *              because the name was invalid (either empty or an illegal
1801  *              provider name) or because the attributes were invalid.
1802  *
1803  *   No other failure code is returned.
1804  *
1805  * 2.2.4  Caller's context
1806  *
1807  *   dtrace_register() may induce calls to dtrace_provide(); the provider must
1808  *   hold no locks across dtrace_register() that may also be acquired by
1809  *   dtrace_provide().  cpu_lock and mod_lock must not be held.
1810  *
1811  * 2.3  int dtrace_unregister(dtrace_provider_t id)
1812  *
1813  * 2.3.1  Overview
1814  *
1815  *   Unregisters the specified provider from the DTrace framework.  It should
1816  *   generally be called by DTrace providers in their detach(9E) entry point.
1817  *
1818  * 2.3.2  Arguments and Notes
1819  *
1820  *   The only argument is the provider identifier, as returned from a
1821  *   successful call to dtrace_register().  As a result of calling
1822  *   dtrace_unregister(), the DTrace framework will call back into the provider
1823  *   via the dtps_destroy() entry point.  Once dtrace_unregister() successfully
1824  *   completes, however, the DTrace framework will no longer make calls through
1825  *   the Framework-to-Provider API.
1826  *
1827  * 2.3.3  Return value
```

```
1828  *
1829  *   On success, dtrace_unregister returns 0.  On failure, dtrace_unregister()
1830  *   returns an errno:
1831  *
1832  *     EBUSY    There are currently processes that have the DTrace pseudodevice
1833  *              open, or there exists an anonymous enabling that hasn't yet
1834  *              been claimed.
1835  *
1836  *   No other failure code is returned.
1837  *
1838  * 2.3.4  Caller's context
1839  *
1840  *   Because a call to dtrace_unregister() may induce calls through the
1841  *   Framework-to-Provider API, the caller may not hold any lock across
1842  *   dtrace_register() that is also acquired in any of the Framework-to-
1843  *   Provider API functions.  Additionally, mod_lock may not be held.
1844  *
1845  * 2.4  void dtrace_invalidate(dtrace_provider_id_t id)
1846  *
1847  * 2.4.1  Overview
1848  *
1849  *   Invalidates the specified provider.  All subsequent probe lookups for the
1850  *   specified provider will fail, but its probes will not be removed.
1851  *
1852  * 2.4.2  Arguments and note
1853  *
1854  *   The only argument is the provider identifier, as returned from a
1855  *   successful call to dtrace_register().  In general, a provider's probes
1856  *   always remain valid; dtrace_invalidate() is a mechanism for invalidating
1857  *   an entire provider, regardless of whether or not probes are enabled or
1858  *   not.  Note that dtrace_invalidate() will _not_ prevent already enabled
1859  *   probes from firing -- it will merely prevent any new enablings of the
1860  *   provider's probes.
1861  *
1862  * 2.5 int dtrace_condense(dtrace_provider_id_t id)
1863  *
1864  * 2.5.1  Overview
1865  *
1866  *   Removes all the unenabled probes for the given provider. This function is
1867  *   not unlike dtrace_unregister(), except that it doesn't remove the
1868  *   provider just as many of its associated probes as it can.
1869  *
1870  * 2.5.2  Arguments and Notes
1871  *
1872  *   As with dtrace_unregister(), the sole argument is the provider identifier
1873  *   as returned from a successful call to dtrace_register().  As a result of
1874  *   calling dtrace_condense(), the DTrace framework will call back into the
1875  *   given provider's dtps_destroy() entry point for each of the provider's
1876  *   unenabled probes.
1877  *
1878  * 2.5.3  Return value
1879  *
1880  *   Currently, dtrace_condense() always returns 0.  However, consumers of this
1881  *   function should check the return value as appropriate; its behavior may
1882  *   change in the future.
1883  *
1884  * 2.5.4  Caller's context
1885  *
1886  *   As with dtrace_unregister(), the caller may not hold any lock across
1887  *   dtrace_condense() that is also acquired in the provider's entry points.
1888  *   Also, mod_lock may not be held.
1889  *
1890  * 2.6 int dtrace_attached()
1891  *
1892  * 2.6.1  Overview
1893  *
```



```
1894  *   Indicates whether or not DTrace has attached.
1895  *
1896  * 2.6.2  Arguments and Notes
1897  *
1898  *   For most providers, DTrace makes initial contact beyond registration.
1899  *   That is, once a provider has registered with DTrace, it waits to hear
1900  *   from DTrace to create probes.  However, some providers may wish to
1901  *   proactively create probes without first being told by DTrace to do so.
1902  *   If providers wish to do this, they must first call dtrace_attached() to
1903  *   determine if DTrace itself has attached.  If dtrace_attached() returns 0,
1904  *   the provider must not make any other Provider-to-Framework API call.
1905  *
1906  * 2.6.3  Return value
1907  *
1908  *   dtrace_attached() returns 1 if DTrace has attached, 0 otherwise.
1909  *
1910  * 2.7  int dtrace_probe_create(dtrace_provider_t id, const char *mod,
1911  *         const char *func, const char *name, int aframes, void *arg)
1912  *
1913  * 2.7.1  Overview
1914  *
1915  *   Creates a probe with specified module name, function name, and name.
1916  *
1917  * 2.7.2  Arguments and Notes
1918  *
1919  *   The first argument is the provider identifier, as returned from a
1920  *   successful call to dtrace_register().  The second, third, and fourth
1921  *   arguments are the module name, function name, and probe name,
1922  *   respectively.  Of these, module name and function name may both be NULL
1923  *   (in which case the probe is considered to be unanchored), or they may both
1924  *   be non-NULL.  The name must be non-NULL, and must point to a non-empty
1925  *   string.
1926  *
1927  *   The fifth argument is the number of artificial stack frames that will be
1928  *   found on the stack when dtrace_probe() is called for the new probe.  These
1929  *   artificial frames will be automatically be pruned should the stack() or
1930  *   stackdepth() functions be called as part of one of the probe's ECBs.  If
1931  *   the parameter doesn't add an artificial frame, this parameter should be
1932  *   zero.
1933  *
1934  *   The final argument is a probe argument that will be passed back to the
1935  *   provider when a probe-specific operation is called.  (e.g., via
1936  *   dtps_enable(), dtps_disable(), etc.)
1937  *
1938  *   Note that it is up to the provider to be sure that the probe that it
1939  *   creates does not already exist -- if the provider is unsure of the probe's
1940  *   existence, it should assure its absence with dtrace_probe_lookup() before
1941  *   calling dtrace_probe_create().
1942  *
1943  * 2.7.3  Return value
1944  *
1945  *   dtrace_probe_create() always succeeds, and always returns the identifier
1946  *   of the newly-created probe.
1947  *
1948  * 2.7.4  Caller's context
1949  *
1950  *   While dtrace_probe_create() is generally expected to be called from
1951  *   dtps_provide() and/or dtps_provide_module(), it may be called from other
1952  *   non-DTrace contexts.  Neither cpu_lock nor mod_lock may be held.
1953  *
1954  * 2.8  dtrace_id_t dtrace_probe_lookup(dtrace_provider_t id, const char *mod,
1955  *         const char *func, const char *name)
1956  *
1957  * 2.8.1  Overview
1958  *
1959  *   Looks up a probe based on provdider and one or more of module name,
```

```
 *   function name and probe name.
 *
 * 2.8.2  Arguments and Notes
 *
 *   The first argument is the provider identifier, as returned from a
 *   successful call to dtrace_register().  The second, third, and fourth
 *   arguments are the module name, function name, and probe name,
 *   respectively.  Any of these may be NULL; dtrace_probe_lookup() will return
 *   the identifier of the first probe that is provided by the specified
 *   provider and matches all of the non-NULL matching criteria.
 *   dtrace_probe_lookup() is generally used by a provider to be check the
 *   existence of a probe before creating it with dtrace_probe_create().
 *
 * 2.8.3  Return value
 *
 *   If the probe exists, returns its identifier.  If the probe does not exist,
 *   return DTRACE_IDNONE.
 *
 * 2.8.4  Caller's context
 *
 *   While dtrace_probe_lookup() is generally expected to be called from
 *   dtps_provide() and/or dtps_provide_module(), it may also be called from
 *   other non-DTrace contexts.  Neither cpu_lock nor mod_lock may be held.
 *
 * 2.9  void *dtrace_probe_arg(dtrace_provider_t id, dtrace_id_t probe)
 *
 * 2.9.1  Overview
 *
 *   Returns the probe argument associated with the specified probe.
 *
 * 2.9.2  Arguments and Notes
 *
 *   The first argument is the provider identifier, as returned from a
 *   successful call to dtrace_register().  The second argument is a probe
 *   identifier, as returned from dtrace_probe_lookup() or
 *   dtrace_probe_create().  This is useful if a probe has multiple
 *   provider-specific components to it:  the provider can create the probe
 *   once with provider-specific state, and then add to the state by looking
 *   up the probe based on probe identifier.
 *
 * 2.9.3  Return value
 *
 *   Returns the argument associated with the specified probe.  If the
 *   specified probe does not exist, or if the specified probe is not provided
 *   by the specified provider, NULL is returned.
 *
 * 2.9.4  Caller's context
 *
 *   While dtrace_probe_arg() is generally expected to be called from
 *   dtps_provide() and/or dtps_provide_module(), it may also be called from
 *   other non-DTrace contexts.  Neither cpu_lock nor mod_lock may be held.
 *
 * 2.10  void dtrace_probe(dtrace_id_t probe, uintptr_t arg0, uintptr_t arg1,
 *             uintptr_t arg2, uintptr_t arg3, uintptr_t arg4)
 *
 * 2.10.1  Overview
 *
 *   The epicenter of DTrace:  fires the specified probes with the specified
 *   arguments.
 *
 * 2.10.2  Arguments and Notes
 *
 *   The first argument is a probe identifier as returned by
 *   dtrace_probe_create() or dtrace_probe_lookup().  The second through sixth
 *   arguments are the values to which the D variables "arg0" through "arg4"
 *   will be mapped.
```



```
 *
 *   dtrace_probe() should be called whenever the specified probe has fired --
 *   however the provider defines it.
 *
 * 2.10.3  Return value
 *
 *   None.
 *
 * 2.10.4  Caller's context
 *
 *   dtrace_probe() may be called in virtually any context:  kernel, user,
 *   interrupt, high-level interrupt, with arbitrary adaptive locks held, with
 *   dispatcher locks held, with interrupts disabled, etc.  The only latitude
 *   that must be afforded to DTrace is the ability to make calls within
 *   itself (and to its in-kernel subroutines) and the ability to access
 *   arbitrary (but mapped) memory.  On some platforms, this constrains
 *   context.  For example, on UltraSPARC, dtrace_probe() cannot be called
 *   from any context in which TL is greater than zero.  dtrace_probe() may
 *   also not be called from any routine which may be called by dtrace_probe()
 *   -- which includes functions in the DTrace framework and some in-kernel
 *   DTrace subroutines.  All such functions "dtrace_"; providers that
 *   instrument the kernel arbitrarily should be sure to not instrument these
 *   routines.
 */
typedef struct dtrace_pops {
        void (*dtps_provide)(void *arg, const dtrace_probedesc_t *spec);
        void (*dtps_provide_module)(void *arg, struct modctl *mp);
        int (*dtps_enable)(void *arg, dtrace_id_t id, void *parg);
        void (*dtps_disable)(void *arg, dtrace_id_t id, void *parg);
        void (*dtps_suspend)(void *arg, dtrace_id_t id, void *parg);
        void (*dtps_resume)(void *arg, dtrace_id_t id, void *parg);
        void (*dtps_getargdesc)(void *arg, dtrace_id_t id, void *parg,
            dtrace_argdesc_t *desc);
        uint64_t (*dtps_getargval)(void *arg, dtrace_id_t id, void *parg,
            int argno, int aframes);
        int (*dtps_mode)(void *arg, dtrace_id_t id, void *parg);
        void (*dtps_destroy)(void *arg, dtrace_id_t id, void *parg);
} dtrace_pops_t;

#define DTRACE_MODE_KERNEL                      0x01
#define DTRACE_MODE_USER                        0x02
#define DTRACE_MODE_NOPRIV_DROP                 0x10
#define DTRACE_MODE_NOPRIV_RESTRICT             0x20
#define DTRACE_MODE_LIMITEDPRIV_RESTRICT        0x40

typedef uintptr_t       dtrace_provider_id_t;

extern int dtrace_register(const char *, const dtrace_pattr_t *, uint32_t,
    cred_t *, const dtrace_pops_t *, void *, dtrace_provider_id_t *);
extern int dtrace_unregister(dtrace_provider_id_t);
extern int dtrace_condense(dtrace_provider_id_t);
extern void dtrace_invalidate(dtrace_provider_id_t);
extern dtrace_id_t dtrace_probe_lookup(dtrace_provider_id_t, const char *,
    const char *, const char *);
extern dtrace_id_t dtrace_probe_create(dtrace_provider_id_t, const char *,
    const char *, const char *, int, void *);
extern void *dtrace_probe_arg(dtrace_provider_id_t, dtrace_id_t);
extern void dtrace_probe(dtrace_id_t, uintptr_t arg0, uintptr_t arg1,
    uintptr_t arg2, uintptr_t arg3, uintptr_t arg4);

/*
 * DTrace Meta Provider API
 *
 * The following functions are implemented by the DTrace framework and are
 * used to implement meta providers. Meta providers plug into the DTrace
 * framework and are used to instantiate new providers on the fly. At
```

```
2092  * present, there is only one type of meta provider and only one meta
2093  * provider may be registered with the DTrace framework at a time. The
2094  * sole meta provider type provides user-land static tracing facilities
2095  * by taking meta probe descriptions and adding a corresponding provider
2096  * into the DTrace framework.
2097  *
2098  * 1 Framework-to-Provider
2099  *
2100  * 1.1 Overview
2101  *
2102  * The Framework-to-Provider API is represented by the dtrace_mops structure
2103  * that the meta provider passes to the framework when registering itself as
2104  * a meta provider. This structure consists of the following members:
2105  *
2106  *   dtms_create_probe()        <-- Add a new probe to a created provider
2107  *   dtms_provide_pid()         <-- Create a new provider for a given process
2108  *   dtms_remove_pid()          <-- Remove a previously created provider
2109  *
2110  * 1.2  void dtms_create_probe(void *arg, void *parg,
2111  *           dtrace_helper_probedesc_t *probedesc);
2112  *
2113  * 1.2.1  Overview
2114  *
2115  *   Called by the DTrace framework to create a new probe in a provider
2116  *   created by this meta provider.
2117  *
2118  * 1.2.2  Arguments and notes
2119  *
2120  *   The first argument is the cookie as passed to dtrace_meta_register().
2121  *   The second argument is the provider cookie for the associated provider;
2122  *   this is obtained from the return value of dtms_provide_pid(). The third
2123  *   argument is the helper probe description.
2124  *
2125  * 1.2.3  Return value
2126  *
2127  *   None
2128  *
2129  * 1.2.4  Caller's context
2130  *
2131  *   dtms_create_probe() is called from either ioctl() or module load context.
2132  *   The DTrace framework is locked in such a way that meta providers may not
2133  *   register or unregister. This means that the meta provider cannot call
2134  *   dtrace_meta_register() or dtrace_meta_unregister(). However, the context is
2135  *   such that the provider may (and is expected to) call provider-related
2136  *   DTrace provider APIs including dtrace_probe_create().
2137  *
2138  * 1.3  void *dtms_provide_pid(void *arg, dtrace_meta_provider_t *mprov,
2139  *            pid_t pid)
2140  *
2141  * 1.3.1  Overview
2142  *
2143  *   Called by the DTrace framework to instantiate a new provider given the
2144  *   description of the provider and probes in the mprov argument. The
2145  *   meta provider should call dtrace_register() to insert the new provider
2146  *   into the DTrace framework.
2147  *
2148  * 1.3.2  Arguments and notes
2149  *
2150  *   The first argument is the cookie as passed to dtrace_meta_register().
2151  *   The second argument is a pointer to a structure describing the new
2152  *   helper provider. The third argument is the process identifier for
2153  *   process associated with this new provider. Note that the name of the
2154  *   provider as passed to dtrace_register() should be the contatenation of
2155  *   the dtmpb_provname member of the mprov argument and the processs
2156  *   identifier as a string.
2157  *
```



```
2158  * 1.3.3  Return value
2159  *
2160  *   The cookie for the provider that the meta provider creates. This is
2161  *   the same value that it passed to dtrace_register().
2162  *
2163  * 1.3.4  Caller's context
2164  *
2165  *   dtms_provide_pid() is called from either ioctl() or module load context.
2166  *   The DTrace framework is locked in such a way that meta providers may not
2167  *   register or unregister. This means that the meta provider cannot call
2168  *   dtrace_meta_register() or dtrace_meta_unregister(). However, the context
2169  *   is such that the provider may -- and is expected to --  call
2170  *   provider-related DTrace provider APIs including dtrace_register().
2171  *
2172  * 1.4  void dtms_remove_pid(void *arg, dtrace_meta_provider_t *mprov,
2173  *           pid_t pid)
2174  *
2175  * 1.4.1  Overview
2176  *
2177  *   Called by the DTrace framework to remove a provider that had previously
2178  *   been instantiated via the dtms_provide_pid() entry point. The meta
2179  *   provider need not remove the provider immediately, but this entry
2180  *   point indicates that the provider should be removed as soon as possible
2181  *   using the dtrace_unregister() API.
2182  *
2183  * 1.4.2  Arguments and notes
2184  *
2185  *   The first argument is the cookie as passed to dtrace_meta_register().
2186  *   The second argument is a pointer to a structure describing the helper
2187  *   provider. The third argument is the process identifier for process
2188  *   associated with this new provider.
2189  *
2190  * 1.4.3  Return value
2191  *
2192  *   None
2193  *
2194  * 1.4.4  Caller's context
2195  *
2196  *   dtms_remove_pid() is called from either ioctl() or exit() context.
2197  *   The DTrace framework is locked in such a way that meta providers may not
2198  *   register or unregister. This means that the meta provider cannot call
2199  *   dtrace_meta_register() or dtrace_meta_unregister(). However, the context
2200  *   is such that the provider may -- and is expected to -- call
2201  *   provider-related DTrace provider APIs including dtrace_unregister().
2202  */
2203 typedef struct dtrace_helper_probedesc {
2204         char *dthpb_mod;                        /* probe module */
2205         char *dthpb_func;                       /* probe function */
2206         char *dthpb_name;                       /* probe name */
2207         uint64_t dthpb_base;                    /* base address */
2208         uint32_t *dthpb_offs;                   /* offsets array */
2209         uint32_t *dthpb_enoffs;                 /* is-enabled offsets array */
2210         uint32_t dthpb_noffs;                   /* offsets count */
2211         uint32_t dthpb_nenoffs;                 /* is-enabled offsets count */
2212         uint8_t *dthpb_args;                    /* argument mapping array */
2213         uint8_t dthpb_xargc;                    /* translated argument count */
2214         uint8_t dthpb_nargc;                    /* native argument count */
2215         char *dthpb_xtypes;                     /* translated types strings */
2216         char *dthpb_ntypes;                     /* native types strings */
2217 } dtrace_helper_probedesc_t;

2219 typedef struct dtrace_helper_provdesc {
2220         char *dthpv_provname;                   /* provider name */
2221         dtrace_pattr_t dthpv_pattr;             /* stability attributes */
2222 } dtrace_helper_provdesc_t;
```

```
2224 typedef struct dtrace_mops {
2225         void (*dtms_create_probe)(void *, void *, dtrace_helper_probedesc_t *);
2226         void *(*dtms_provide_pid)(void *, dtrace_helper_provdesc_t *, pid_t);
2227         void (*dtms_remove_pid)(void *, dtrace_helper_provdesc_t *, pid_t);
2228 } dtrace_mops_t;

2230 typedef uintptr_t       dtrace_meta_provider_id_t;

2232 extern int dtrace_meta_register(const char *, const dtrace_mops_t *, void *,
2233     dtrace_meta_provider_id_t *);
2234 extern int dtrace_meta_unregister(dtrace_meta_provider_id_t);

2236 /*
2237  * DTrace Kernel Hooks
2238  *
2239  * The following functions are implemented by the base kernel and form a set of
2240  * hooks used by the DTrace framework.  DTrace hooks are implemented in either
2241  * uts/common/os/dtrace_subr.c, an ISA-specific assembly file, or in a
2242  * uts/<platform>/os/dtrace_subr.c corresponding to each hardware platform.
2243  */

2245 typedef enum dtrace_vtime_state {
2246         DTRACE_VTIME_INACTIVE = 0,      /* No DTrace, no TNF */
2247         DTRACE_VTIME_ACTIVE,            /* DTrace virtual time, no TNF */
2248         DTRACE_VTIME_INACTIVE_TNF,      /* No DTrace, TNF active */
2249         DTRACE_VTIME_ACTIVE_TNF         /* DTrace virtual time _and_ TNF */
2250 } dtrace_vtime_state_t;

2252 extern dtrace_vtime_state_t dtrace_vtime_active;
2253 extern void dtrace_vtime_switch(kthread_t *next);
2254 extern void dtrace_vtime_enable_tnf(void);
2255 extern void dtrace_vtime_disable_tnf(void);
2256 extern void dtrace_vtime_enable(void);
2257 extern void dtrace_vtime_disable(void);

2259 struct regs;

2261 extern int (*dtrace_pid_probe_ptr)(struct regs *);
2262 extern int (*dtrace_return_probe_ptr)(struct regs *);
2263 extern void (*dtrace_fasttrap_fork_ptr)(proc_t *, proc_t *);
2264 extern void (*dtrace_fasttrap_exec_ptr)(proc_t *);
2265 extern void (*dtrace_fasttrap_exit_ptr)(proc_t *);
2266 extern void dtrace_fasttrap_fork(proc_t *, proc_t *);

2268 typedef uintptr_t dtrace_icookie_t;
2269 typedef void (*dtrace_xcall_t)(void *);

2271 extern dtrace_icookie_t dtrace_interrupt_disable(void);
2272 extern void dtrace_interrupt_enable(dtrace_icookie_t);

2274 extern void dtrace_membar_producer(void);
2275 extern void dtrace_membar_consumer(void);

2277 extern void (*dtrace_cpu_init)(processorid_t);
2278 extern void (*dtrace_modload)(struct modctl *);
2279 extern void (*dtrace_modunload)(struct modctl *);
2280 extern void (*dtrace_helpers_cleanup)();
2281 extern void (*dtrace_helpers_fork)(proc_t *parent, proc_t *child);
2282 extern void (*dtrace_cpustart_init)();
2283 extern void (*dtrace_cpustart_fini)();
2284 extern void (*dtrace_closef)();

2286 extern void (*dtrace_debugger_init)();
2287 extern void (*dtrace_debugger_fini)();
2288 extern dtrace_cacheid_t dtrace_predcache_id;
```



```
2290 extern hrtime_t dtrace_gethrtime(void);
2291 extern void dtrace_sync(void);
2292 extern void dtrace_toxic_ranges(void (*)(uintptr_t, uintptr_t));
2293 extern void dtrace_xcall(processorid_t, dtrace_xcall_t, void *);
2294 extern void dtrace_vpanic(const char *, __va_list);
2295 extern void dtrace_panic(const char *, ...);

2297 extern int dtrace_safe_defer_signal(void);
2298 extern void dtrace_safe_synchronous_signal(void);

2300 extern int dtrace_mach_aframes(void);

2302 #if defined(__i386) || defined(__amd64)
2303 extern int dtrace_instr_size(uchar_t *instr);
2304 extern int dtrace_instr_size_isa(uchar_t *, model_t, int *);
2305 extern void dtrace_invop_add(int (*)(uintptr_t, uintptr_t *, uintptr_t));
2306 extern void dtrace_invop_remove(int (*)(uintptr_t, uintptr_t *, uintptr_t));
2307 extern void dtrace_invop_callsite(void);
2308 #endif

2310 #ifdef __sparc
2311 extern int dtrace_blksuword32(uintptr_t, uint32_t *, int);
2312 extern void dtrace_getfsr(uint64_t *);
2313 #endif

2315 #define DTRACE_CPUFLAG_ISSET(flag) \
2316         (cpu_core[CPU->cpu_id].cpuc_dtrace_flags & (flag))

2318 #define DTRACE_CPUFLAG_SET(flag) \
2319         (cpu_core[CPU->cpu_id].cpuc_dtrace_flags |= (flag))

2321 #define DTRACE_CPUFLAG_CLEAR(flag) \
2322         (cpu_core[CPU->cpu_id].cpuc_dtrace_flags &= ~(flag))

2324 #endif /* _KERNEL */

2326 #endif  /* _ASM */

2328 #if defined(__i386) || defined(__amd64)

2330 #define DTRACE_INVOP_PUSHL_EBP          1
2331 #define DTRACE_INVOP_POPL_EBP           2
2332 #define DTRACE_INVOP_LEAVE              3
2333 #define DTRACE_INVOP_NOP                4
2334 #define DTRACE_INVOP_RET                5

2336 #endif

2338 #ifdef  __cplusplus
2339 }
2340 #endif

2342 #endif  /* _SYS_DTRACE_H */
```