****
1124 Sat Apr 12 11:18:53 2014
new/usr/src/cmd/dtrace/demo/sctpbytes.d
3903 DTrace SCTP Provider
****

```
#!/usr/sbin/dtrace -s
/*
 * 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) 2010, Oracle and/or its affiliates. All rights reserved.
 */

sctp:::receive
{
        @bytes[args[2]->ip_saddr, args[4]->sctp_dport] =
            sum(args[4]->sctp_length);
}

sctp:::send
{
        @bytes[args[2]->ip_daddr, args[4]->sctp_sport] =
            sum(args[4]->sctp_length);
}
#endif /* ! codereview */
```

```
**********************************************************
    7008 Sat Apr 12 11:18:54 2014
new/usr/src/lib/libdtrace/Makefile.com
3903 DTrace SCTP Provider
**********************************************************
#
# 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) 2012 by Delphix. All rights reserved.
#

LIBRARY = libdtrace.a
VERS = .1

LIBSRCS = \
        dt_aggregate.c \
        dt_as.c \
        dt_buf.c \
        dt_cc.c \
        dt_cg.c \
        dt_consume.c \
        dt_decl.c \
        dt_dis.c \
        dt_dof.c \
        dt_error.c \
        dt_errtags.c \
        dt_handle.c \
        dt_ident.c \
        dt_inttab.c \
        dt_link.c \
        dt_list.c \
        dt_open.c \
        dt_options.c \
        dt_program.c \
        dt_map.c \
        dt_module.c \
        dt_names.c \
        dt_parser.c \
        dt_pcb.c \
        dt_pid.c \
        dt_pq.c \
        dt_pragma.c \
        dt_print.c \
        dt_printf.c \
        dt_proc.c \
        dt_provider.c \
        dt_regset.c \
```



```
        dt_string.c \
        dt_strtab.c \
        dt_subr.c \
        dt_work.c \
        dt_xlator.c

LIBISASRCS = \
        dt_isadep.c

OBJECTS = dt_lex.o dt_grammar.o $(MACHOBJS) $(LIBSRCS:%.c=%.o) $(LIBISASRCS:%.c=

DRTISRCS = dlink_init.c dlink_common.c
DRTIOBJS = $(DRTISRCS:%.c=pics/%.o)
DRTIOBJ = drti.o

LIBDAUDITSRCS = dlink_audit.c dlink_common.c
LIBDAUDITOBJS = $(LIBDAUDITSRCS:%.c=pics/%.o)
LIBDAUDIT = libdtrace_forceload.so

DLINKSRCS = dlink_common.c dlink_init.c dlink_audit.c

DLIBSRCS += \
        errno.d \
        fc.d \
        io.d \
        ip.d \
        iscsit.d \
        net.d \
        nfs.d \
        nfssrv.d \
        procfs.d \
        regs.d \
        sched.d \
        signal.d \
        scsi.d \
        srp.d \
        sysevent.d \
        tcp.d \
        udp.d \
        sctp.d \
#endif /* ! codereview */
        unistd.d

include ../../Makefile.lib

SRCS = $(LIBSRCS:%.c=../common/%.c) $(LIBISASRCS:%.c=../$(MACH)/%.c)
LIBS = $(DYNLIB) $(LINTLIB)

SRCDIR = ../common

CLEANFILES += dt_lex.c dt_grammar.c dt_grammar.h y.output
CLEANFILES += ../common/procfs.sed ../common/procfs.d
CLEANFILES += ../common/io.sed ../common/io.d
CLEANFILES += ../common/ip.sed ../common/ip.d
CLEANFILES += ../common/net.sed ../common/net.d
CLEANFILES += ../common/errno.d ../common/signal.d
CLEANFILES += ../common/dt_errtags.c ../common/dt_names.c
CLEANFILES += ../common/sysevent.sed ../common/sysevent.d
CLEANFILES += ../common/tcp.sed ../common/tcp.d
CLEANFILES += ../common/udp.sed ../common/udp.d
CLEANFILES += ../common/sctp.sed ../common/sctp.d
#endif /* ! codereview */
CLEANFILES += $(LIBDAUDITOBJS) $(DRTIOBJS)

CLOBBERFILES += $(LIBDAUDIT) drti.o
```

```
128 CPPFLAGS += -I../common -I.
129 CFLAGS += $(CCVERBOSE) $(C_BIGPICFLAGS)
130 CFLAGS64 += $(CCVERBOSE) $(C_BIGPICFLAGS)

132 CERRWARN += -_gcc=-Wno-unused-label
133 CERRWARN += -_gcc=-Wno-unused-variable
134 CERRWARN += -_gcc=-Wno-parentheses
135 CERRWARN += -_gcc=-Wno-uninitialized
136 CERRWARN += -_gcc=-Wno-switch

138 YYCFLAGS =
139 LDLIBS += -lgen -lproc -lrtld_db -lnsl -lsocket -lctf -lelf -lc
140 DRTILDLIBS = $(LDLIBS.lib) -lc

142 yydebug := YYCFLAGS += -DYYDEBUG

144 $(LINTLIB) := SRCS = $(SRCDIR)/$(LINTSRC)

146 LFLAGS = -t -v
147 YFLAGS = -d -v

149 ROOTDLIBDIR = $(ROOT)/usr/lib/dtrace
150 ROOTDLIBDIR64 = $(ROOT)/usr/lib/dtrace/64

152 ROOTDLIBS = $(DLIBSRCS:%=$(ROOTDLIBDIR)/%)
153 ROOTDOBJS = $(ROOTDLIBDIR)/$(DRTIOBJ) $(ROOTDLIBDIR)/$(LIBDAUDIT)
154 ROOTDOBJS64 = $(ROOTDLIBDIR64)/$(DRTIOBJ) $(ROOTDLIBDIR64)/$(LIBDAUDIT)

156 $(ROOTDLIBDIR)/%.d := FILEMODE=444
157 $(ROOTDLIBDIR)/%.o := FILEMODE=444
158 $(ROOTDLIBDIR64)/%.o := FILEMODE=444
159 $(ROOTDLIBDIR)/%.so := FILEMODE=555
160 $(ROOTDLIBDIR64)/%.so := FILEMODE=555

162 .KEEP_STATE:

164 all: $(LIBS) $(DRTIOBJ) $(LIBDAUDIT)

166 lint: lintdlink lintcheck

168 lintdlink: $(DLINKSRCS:%.c=../common/%.c)
169         $(LINT.c) $(DLINKSRCS:%.c=../common/%.c) $(DRTILDLIBS)

171 dt_lex.c: $(SRCDIR)/dt_lex.l dt_grammar.h
172         $(LEX) $(LFLAGS) $(SRCDIR)/dt_lex.l > $@

174 dt_grammar.c dt_grammar.h: $(SRCDIR)/dt_grammar.y
175         $(YACC) $(YFLAGS) $(SRCDIR)/dt_grammar.y
176         @mv y.tab.h dt_grammar.h
177         @mv y.tab.c dt_grammar.c

179 pics/dt_lex.o pics/dt_grammar.o := CFLAGS += $(YYCFLAGS)
180 pics/dt_lex.o pics/dt_grammar.o := CFLAGS64 += $(YYCFLAGS)

182 pics/dt_lex.o pics/dt_grammar.o := CERRWARN += -erroff=E_STATEMENT_NOT_REACHED
183 pics/dt_lex.o pics/dt_grammar.o := CCVERBOSE =

185 ../common/dt_errtags.c: ../common/mkerrtags.sh ../common/dt_errtags.h
186         sh ../common/mkerrtags.sh < ../common/dt_errtags.h > $@

188 ../common/dt_names.c: ../common/mknames.sh $(SRC)/uts/common/sys/dtrace.h
189         sh ../common/mknames.sh < $(SRC)/uts/common/sys/dtrace.h > $@

191 ../common/errno.d: ../common/mkerrno.sh $(SRC)/uts/common/sys/errno.h
192         sh ../common/mkerrno.sh < $(SRC)/uts/common/sys/errno.h > $@
```



```
194 ../common/signal.d: ../common/mksignal.sh $(SRC)/uts/common/sys/iso/signal_iso.h
195         sh ../common/mksignal.sh < $(SRC)/uts/common/sys/iso/signal_iso.h > $@

197 ../common/%.sed: ../common/%.sed.in
198         $(COMPILE.cpp) -D_KERNEL $< | tr -d ' ' | tr '"' '@' | \
199             sed 's/\&/\\\&/g' | grep '^s/' > $@

201 ../common/procfs.d: ../common/procfs.sed ../common/procfs.d.in
202         sed -f ../common/procfs.sed < ../common/procfs.d.in > $@

204 ../common/io.d: ../common/io.sed ../common/io.d.in
205         sed -f ../common/io.sed < ../common/io.d.in > $@

207 ../common/ip.d: ../common/ip.sed ../common/ip.d.in
208         sed -f ../common/ip.sed < ../common/ip.d.in > $@

210 ../common/net.d: ../common/net.sed ../common/net.d.in
211         sed -f ../common/net.sed < ../common/net.d.in > $@

213 ../common/sysevent.d: ../common/sysevent.sed ../common/sysevent.d.in
214         sed -f ../common/sysevent.sed < ../common/sysevent.d.in > $@

216 ../common/tcp.d: ../common/tcp.sed ../common/tcp.d.in
101 ../common/tcp.d: ..//common/tcp.sed ../common/tcp.d.in
217         sed -f ../common/tcp.sed < ../common/tcp.d.in > $@

219 ../common/udp.d: ../common/udp.sed ../common/udp.d.in
220         sed -f ../common/udp.sed < ../common/udp.d.in > $@

222 ../common/sctp.d: ../common/sctp.sed ../common/sctp.d.in
223         sed -f ../common/sctp.sed < ../common/sctp.d.in > $@

225 #endif /* ! codereview */
226 pics/%.o: ../$(MACH)/%.c
227         $(COMPILE.c) -o $@ $<
228         $(POST_PROCESS_O)

230 pics/%.o: ../$(MACH)/%.s
231         $(COMPILE.s) -o $@ $<
232         $(POST_PROCESS_O)

234 $(DRTIOBJ): $(DRTIOBJS)
235         $(LD) -o $@ -r -Blocal -Breduce $(DRTIOBJS)
236         $(POST_PROCESS_O)

238 $(LIBDAUDIT): $(LIBDAUDITOBJS)
239         $(LINK.c) -o $@ $(GSHARED) -h$(LIBDAUDIT) $(ZTEXT) $(ZDEFS) $(BDIRECT) \
240             $(MAPFILE.PGA:%=-M%) $(MAPFILE.NED:%=-M%) $(LIBDAUDITOBJS) \
241             -lmapmalloc -lc -lproc
242         $(POST_PROCESS_SO)

244 $(ROOTDLIBDIR):
245         $(INS.dir)

247 $(ROOTDLIBDIR64): $(ROOTDLIBDIR)
248         $(INS.dir)

250 $(ROOTDLIBDIR)/%.d: ../common/%.d
251         $(INS.file)

253 $(ROOTDLIBDIR)/%.d: ../$(MACH)/%.d
254         $(INS.file)

256 $(ROOTDLIBDIR)/%.d: %.d
257         $(INS.file)
```

```
259 $(ROOTDLIBDIR)/%.o: %.o
260         $(INS.file)

262 $(ROOTDLIBDIR64)/%.o: %.o
263         $(INS.file)

265 $(ROOTDLIBDIR)/%.so: %.so
266         $(INS.file)

268 $(ROOTDLIBDIR64)/%.so: %.so
269         $(INS.file)

271 $(ROOTDLIBS): $(ROOTDLIBDIR)

273 $(ROOTDOBJS): $(ROOTDLIBDIR)

275 $(ROOTDOBJS64): $(ROOTDLIBDIR64)

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

```
**********************************************************
   53723 Sat Apr 12 11:18:54 2014
new/usr/src/lib/libdtrace/common/dt_open.c
3903 DTrace SCTP Provider
**********************************************************
______unchanged_portion_omitted_

  82 /*
  83  * The version number should be increased for every customer visible release
  84  * of DTrace. The major number should be incremented when a fundamental
  85  * change has been made that would affect all consumers, and would reflect
  86  * sweeping changes to DTrace or the D language. The minor number should be
  87  * incremented when a change is introduced that could break scripts that had
  88  * previously worked; for example, adding a new built-in variable could break
  89  * a script which was already using that identifier. The micro number should
  90  * be changed when introducing functionality changes or major bug fixes that
  91  * do not affect backward compatibility -- this is merely to make capabilities
  92  * easily determined from the version number. Minor bugs do not require any
  93  * modification to the version number.
  94  */
  95 #define DT_VERS_1_0     DT_VERSION_NUMBER(1, 0, 0)
  96 #define DT_VERS_1_1     DT_VERSION_NUMBER(1, 1, 0)
  97 #define DT_VERS_1_2     DT_VERSION_NUMBER(1, 2, 0)
  98 #define DT_VERS_1_2_1   DT_VERSION_NUMBER(1, 2, 1)
  99 #define DT_VERS_1_2_2   DT_VERSION_NUMBER(1, 2, 2)
 100 #define DT_VERS_1_3     DT_VERSION_NUMBER(1, 3, 0)
 101 #define DT_VERS_1_4     DT_VERSION_NUMBER(1, 4, 0)
 102 #define DT_VERS_1_4_1   DT_VERSION_NUMBER(1, 4, 1)
 103 #define DT_VERS_1_5     DT_VERSION_NUMBER(1, 5, 0)
 104 #define DT_VERS_1_6     DT_VERSION_NUMBER(1, 6, 0)
 105 #define DT_VERS_1_6_1   DT_VERSION_NUMBER(1, 6, 1)
 106 #define DT_VERS_1_6_2   DT_VERSION_NUMBER(1, 6, 2)
 107 #define DT_VERS_1_6_3   DT_VERSION_NUMBER(1, 6, 3)
 108 #define DT_VERS_1_7     DT_VERSION_NUMBER(1, 7, 0)
 109 #define DT_VERS_1_7_1   DT_VERSION_NUMBER(1, 7, 1)
 110 #define DT_VERS_1_8     DT_VERSION_NUMBER(1, 8, 0)
 111 #define DT_VERS_1_8_1   DT_VERSION_NUMBER(1, 8, 1)
 112 #define DT_VERS_1_9     DT_VERSION_NUMBER(1, 9, 0)
 113 #define DT_VERS_1_9_1   DT_VERSION_NUMBER(1, 9, 1)
 114 #define DT_VERS_1_9_2   DT_VERSION_NUMBER(1, 9, 2)
 115 #define DT_VERS_LATEST  DT_VERS_1_9_2
 116 #define DT_VERS_STRING  "Sun D 1.9.2"
 114 #define DT_VERS_LATEST  DT_VERS_1_9_1
 115 #define DT_VERS_STRING  "Sun D 1.9.1"

 118 const dt_version_t _dtrace_versions[] = {
 119         DT_VERS_1_0,    /* D API 1.0.0 (PSARC 2001/466) Solaris 10 FCS */
 120         DT_VERS_1_1,    /* D API 1.1.0 Solaris Express 6/05 */
 121         DT_VERS_1_2,    /* D API 1.2.0 Solaris 10 Update 1 */
 122         DT_VERS_1_2_1,  /* D API 1.2.1 Solaris Express 4/06 */
 123         DT_VERS_1_2_2,  /* D API 1.2.2 Solaris Express 6/06 */
 124         DT_VERS_1_3,    /* D API 1.3 Solaris Express 10/06 */
 125         DT_VERS_1_4,    /* D API 1.4 Solaris Express 2/07 */
 126         DT_VERS_1_4_1,  /* D API 1.4.1 Solaris Express 4/07 */
 127         DT_VERS_1_5,    /* D API 1.5 Solaris Express 7/07 */
 128         DT_VERS_1_6,    /* D API 1.6 */
 129         DT_VERS_1_6_1,  /* D API 1.6.1 */
 130         DT_VERS_1_6_2,  /* D API 1.6.2 */
 131         DT_VERS_1_6_3,  /* D API 1.6.3 */
 132         DT_VERS_1_7,    /* D API 1.7 */
 133         DT_VERS_1_7_1,  /* D API 1.7.1 */
 134         DT_VERS_1_8,    /* D API 1.8 */
 135         DT_VERS_1_8_1,  /* D API 1.8.1 */
 136         DT_VERS_1_9,    /* D API 1.9 */
 137         DT_VERS_1_9_1,  /* D API 1.9.1 */
 138         DT_VERS_1_9_2,  /* D API 1.9.2 */
```



```
 139 #endif /* ! codereview */
 140         0
 141 };

 143 /*
 144  * Table of global identifiers.  This is used to populate the global identifier
 145  * hash when a new dtrace client open occurs.  For more info see dt_ident.h.
 146  * The global identifiers that represent functions use the dt_idops_func ops
 147  * and specify the private data pointer as a prototype string which is parsed
 148  * when the identifier is first encountered.  These prototypes look like ANSI
 149  * C function prototypes except that the special symbol "@" can be used as a
 150  * wildcard to represent a single parameter of any type (i.e. any dt_node_t).
 151  * The standard "..." notation can also be used to represent varargs.  An empty
 152  * parameter list is taken to mean void (that is, no arguments are permitted).
 153  * A parameter enclosed in square brackets (e.g. "[int]") denotes an optional
 154  * argument.
 155  */
 156 static const dt_ident_t _dtrace_globals[] = {
 157 { "alloca", DT_IDENT_FUNC, 0, DIF_SUBR_ALLOCA, DT_ATTR_STABCMN, DT_VERS_1_0,
 158         &dt_idops_func, "void *(size_t)" },
 159 { "arg0", DT_IDENT_SCALAR, 0, DIF_VAR_ARG0, DT_ATTR_STABCMN, DT_VERS_1_0,
 160         &dt_idops_type, "int64_t" },
 161 { "arg1", DT_IDENT_SCALAR, 0, DIF_VAR_ARG1, DT_ATTR_STABCMN, DT_VERS_1_0,
 162         &dt_idops_type, "int64_t" },
 163 { "arg2", DT_IDENT_SCALAR, 0, DIF_VAR_ARG2, DT_ATTR_STABCMN, DT_VERS_1_0,
 164         &dt_idops_type, "int64_t" },
 165 { "arg3", DT_IDENT_SCALAR, 0, DIF_VAR_ARG3, DT_ATTR_STABCMN, DT_VERS_1_0,
 166         &dt_idops_type, "int64_t" },
 167 { "arg4", DT_IDENT_SCALAR, 0, DIF_VAR_ARG4, DT_ATTR_STABCMN, DT_VERS_1_0,
 168         &dt_idops_type, "int64_t" },
 169 { "arg5", DT_IDENT_SCALAR, 0, DIF_VAR_ARG5, DT_ATTR_STABCMN, DT_VERS_1_0,
 170         &dt_idops_type, "int64_t" },
 171 { "arg6", DT_IDENT_SCALAR, 0, DIF_VAR_ARG6, DT_ATTR_STABCMN, DT_VERS_1_0,
 172         &dt_idops_type, "int64_t" },
 173 { "arg7", DT_IDENT_SCALAR, 0, DIF_VAR_ARG7, DT_ATTR_STABCMN, DT_VERS_1_0,
 174         &dt_idops_type, "int64_t" },
 175 { "arg8", DT_IDENT_SCALAR, 0, DIF_VAR_ARG8, DT_ATTR_STABCMN, DT_VERS_1_0,
 176         &dt_idops_type, "int64_t" },
 177 { "arg9", DT_IDENT_SCALAR, 0, DIF_VAR_ARG9, DT_ATTR_STABCMN, DT_VERS_1_0,
 178         &dt_idops_type, "int64_t" },
 179 { "args", DT_IDENT_ARRAY, 0, DIF_VAR_ARGS, DT_ATTR_STABCMN, DT_VERS_1_0,
 180         &dt_idops_args, NULL },
 181 { "avg", DT_IDENT_AGGFUNC, 0, DTRACEAGG_AVG, DT_ATTR_STABCMN, DT_VERS_1_0,
 182         &dt_idops_func, "void(@)" },
 183 { "basename", DT_IDENT_FUNC, 0, DIF_SUBR_BASENAME, DT_ATTR_STABCMN, DT_VERS_1_0,
 184         &dt_idops_func, "string(const char *)" },
 185 { "bcopy", DT_IDENT_FUNC, 0, DIF_SUBR_BCOPY, DT_ATTR_STABCMN, DT_VERS_1_0,
 186         &dt_idops_func, "void(void *, void *, size_t)" },
 187 { "breakpoint", DT_IDENT_ACTFUNC, 0, DT_ACT_BREAKPOINT,
 188         DT_ATTR_STABCMN, DT_VERS_1_0,
 189         &dt_idops_func, "void()" },
 190 { "caller", DT_IDENT_SCALAR, 0, DIF_VAR_CALLER, DT_ATTR_STABCMN, DT_VERS_1_0,
 191         &dt_idops_type, "uintptr_t" },
 192 { "chill", DT_IDENT_ACTFUNC, 0, DT_ACT_CHILL, DT_ATTR_STABCMN, DT_VERS_1_0,
 193         &dt_idops_func, "void(int)" },
 194 { "cleanpath", DT_IDENT_FUNC, 0, DIF_SUBR_CLEANPATH, DT_ATTR_STABCMN,
 195         DT_VERS_1_0, &dt_idops_func, "string(const char *)" },
 196 { "clear", DT_IDENT_ACTFUNC, 0, DT_ACT_CLEAR, DT_ATTR_STABCMN, DT_VERS_1_0,
 197         &dt_idops_func, "void(...)" },
 198 { "commit", DT_IDENT_ACTFUNC, 0, DT_ACT_COMMIT, DT_ATTR_STABCMN, DT_VERS_1_0,
 199         &dt_idops_func, "void(int)" },
 200 { "copyin", DT_IDENT_FUNC, 0, DIF_SUBR_COPYIN, DT_ATTR_STABCMN, DT_VERS_1_0,
 201         &dt_idops_func, "void *(uintptr_t, size_t)" },
 202 { "copyinstr", DT_IDENT_FUNC, 0, DIF_SUBR_COPYINSTR,
 203         DT_ATTR_STABCMN, DT_VERS_1_0,
 204         &dt_idops_func, "string(uintptr_t, [size_t])" },
```

```
{ "copyinto", DT_IDENT_FUNC, 0, DIF_SUBR_COPYINTO, DT_ATTR_STABCMN,
        DT_VERS_1_0, &dt_idops_func, "void(uintptr_t, size_t, void *)" },
{ "copyout", DT_IDENT_FUNC, 0, DIF_SUBR_COPYOUT, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(void *, uintptr_t, size_t)" },
{ "copyoutstr", DT_IDENT_FUNC, 0, DIF_SUBR_COPYOUTSTR,
        DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(char *, uintptr_t, size_t)" },
{ "count", DT_IDENT_AGGFUNC, 0, DTRACEAGG_COUNT, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void()" },
{ "curthread", DT_IDENT_SCALAR, 0, DIF_VAR_CURTHREAD,
        { DTRACE_STABILITY_STABLE, DTRACE_STABILITY_PRIVATE,
        DTRACE_CLASS_COMMON }, DT_VERS_1_0,
        &dt_idops_type, "genunix`kthread_t *" },
{ "ddi_pathname", DT_IDENT_FUNC, 0, DIF_SUBR_DDI_PATHNAME,
        DT_ATTR_EVOLCMN, DT_VERS_1_0,
        &dt_idops_func, "string(void *, int64_t)" },
{ "denormalize", DT_IDENT_ACTFUNC, 0, DT_ACT_DENORMALIZE, DT_ATTR_STABCMN,
        DT_VERS_1_0, &dt_idops_func, "void(...)" },
{ "dirname", DT_IDENT_FUNC, 0, DIF_SUBR_DIRNAME, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "string(const char *)" },
{ "discard", DT_IDENT_ACTFUNC, 0, DT_ACT_DISCARD, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(int)" },
{ "epid", DT_IDENT_SCALAR, 0, DIF_VAR_EPID, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_type, "uint_t" },
{ "errno", DT_IDENT_SCALAR, 0, DIF_VAR_ERRNO, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_type, "int" },
{ "execname", DT_IDENT_SCALAR, 0, DIF_VAR_EXECNAME,
        DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
{ "exit", DT_IDENT_ACTFUNC, 0, DT_ACT_EXIT, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(int)" },
{ "freopen", DT_IDENT_ACTFUNC, 0, DT_ACT_FREOPEN, DT_ATTR_STABCMN,
        DT_VERS_1_1, &dt_idops_func, "void(@, ...)" },
{ "ftruncate", DT_IDENT_ACTFUNC, 0, DT_ACT_FTRUNCATE, DT_ATTR_STABCMN,
        DT_VERS_1_0, &dt_idops_func, "void()" },
{ "func", DT_IDENT_ACTFUNC, 0, DT_ACT_SYM, DT_ATTR_STABCMN,
        DT_VERS_1_2, &dt_idops_func, "_symaddr(uintptr_t)" },
{ "getmajor", DT_IDENT_FUNC, 0, DIF_SUBR_GETMAJOR,
        DT_ATTR_EVOLCMN, DT_VERS_1_0,
        &dt_idops_func, "genunix`major_t(genunix`dev_t)" },
{ "getminor", DT_IDENT_FUNC, 0, DIF_SUBR_GETMINOR,
        DT_ATTR_EVOLCMN, DT_VERS_1_0,
        &dt_idops_func, "genunix`minor_t(genunix`dev_t)" },
{ "htonl", DT_IDENT_FUNC, 0, DIF_SUBR_HTONL, DT_ATTR_EVOLCMN, DT_VERS_1_3,
        &dt_idops_func, "uint32_t(uint32_t)" },
{ "htonll", DT_IDENT_FUNC, 0, DIF_SUBR_HTONLL, DT_ATTR_EVOLCMN, DT_VERS_1_3,
        &dt_idops_func, "uint64_t(uint64_t)" },
{ "htons", DT_IDENT_FUNC, 0, DIF_SUBR_HTONS, DT_ATTR_EVOLCMN, DT_VERS_1_3,
        &dt_idops_func, "uint16_t(uint16_t)" },
{ "gid", DT_IDENT_SCALAR, 0, DIF_VAR_GID, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_type, "gid_t" },
{ "id", DT_IDENT_SCALAR, 0, DIF_VAR_ID, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_type, "uint_t" },
{ "index", DT_IDENT_FUNC, 0, DIF_SUBR_INDEX, DT_ATTR_STABCMN, DT_VERS_1_1,
        &dt_idops_func, "int(const char *, const char *, [int])" },
{ "inet_ntoa", DT_IDENT_FUNC, 0, DIF_SUBR_INET_NTOA, DT_ATTR_STABCMN,
        DT_VERS_1_5, &dt_idops_func, "string(ipaddr_t *)" },
{ "inet_ntoa6", DT_IDENT_FUNC, 0, DIF_SUBR_INET_NTOA6, DT_ATTR_STABCMN,
        DT_VERS_1_5, &dt_idops_func, "string(in6_addr_t *)" },
{ "inet_ntop", DT_IDENT_FUNC, 0, DIF_SUBR_INET_NTOP, DT_ATTR_STABCMN,
        DT_VERS_1_5, &dt_idops_func, "string(int, void *)" },
{ "ipl", DT_IDENT_SCALAR, 0, DIF_VAR_IPL, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_type, "uint_t" },
{ "jstack", DT_IDENT_ACTFUNC, 0, DT_ACT_JSTACK, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "stack(...)" },
{ "lltostr", DT_IDENT_FUNC, 0, DIF_SUBR_LLTOSTR, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "string(int64_t, [int])" },
```



```
{ "llquantize", DT_IDENT_AGGFUNC, 0, DTRACEAGG_LLQUANTIZE, DT_ATTR_STABCMN,
        DT_VERS_1_7, &dt_idops_func,
        "void(@, int32_t, int32_t, int32_t, int32_t, ...)" },
{ "lquantize", DT_IDENT_AGGFUNC, 0, DTRACEAGG_LQUANTIZE,
        DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(@, int32_t, int32_t, ...)" },
{ "max", DT_IDENT_AGGFUNC, 0, DTRACEAGG_MAX, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(@)" },
{ "min", DT_IDENT_AGGFUNC, 0, DTRACEAGG_MIN, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(@)" },
{ "mod", DT_IDENT_ACTFUNC, 0, DT_ACT_MOD, DT_ATTR_STABCMN,
        DT_VERS_1_2, &dt_idops_func, "_symaddr(uintptr_t)" },
{ "msgdsize", DT_IDENT_FUNC, 0, DIF_SUBR_MSGDSIZE,
        DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "size_t(mblk_t *)" },
{ "msgsize", DT_IDENT_FUNC, 0, DIF_SUBR_MSGSIZE,
        DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "size_t(mblk_t *)" },
{ "mutex_owned", DT_IDENT_FUNC, 0, DIF_SUBR_MUTEX_OWNED,
        DT_ATTR_EVOLCMN, DT_VERS_1_0,
        &dt_idops_func, "int(genunix`kmutex_t *)" },
{ "mutex_owner", DT_IDENT_FUNC, 0, DIF_SUBR_MUTEX_OWNER,
        DT_ATTR_EVOLCMN, DT_VERS_1_0,
        &dt_idops_func, "genunix`kthread_t *(genunix`kmutex_t *)" },
{ "mutex_type_adaptive", DT_IDENT_FUNC, 0, DIF_SUBR_MUTEX_TYPE_ADAPTIVE,
        DT_ATTR_EVOLCMN, DT_VERS_1_0,
        &dt_idops_func, "int(genunix`kmutex_t *)" },
{ "mutex_type_spin", DT_IDENT_FUNC, 0, DIF_SUBR_MUTEX_TYPE_SPIN,
        DT_ATTR_EVOLCMN, DT_VERS_1_0,
        &dt_idops_func, "int(genunix`kmutex_t *)" },
{ "ntohl", DT_IDENT_FUNC, 0, DIF_SUBR_NTOHL, DT_ATTR_EVOLCMN, DT_VERS_1_3,
        &dt_idops_func, "uint32_t(uint32_t)" },
{ "ntohll", DT_IDENT_FUNC, 0, DIF_SUBR_NTOHLL, DT_ATTR_EVOLCMN, DT_VERS_1_3,
        &dt_idops_func, "uint64_t(uint64_t)" },
{ "ntohs", DT_IDENT_FUNC, 0, DIF_SUBR_NTOHS, DT_ATTR_EVOLCMN, DT_VERS_1_3,
        &dt_idops_func, "uint16_t(uint16_t)" },
{ "normalize", DT_IDENT_ACTFUNC, 0, DT_ACT_NORMALIZE, DT_ATTR_STABCMN,
        DT_VERS_1_0, &dt_idops_func, "void(...)" },
{ "panic", DT_IDENT_ACTFUNC, 0, DT_ACT_PANIC, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void()" },
{ "pid", DT_IDENT_SCALAR, 0, DIF_VAR_PID, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_type, "pid_t" },
{ "ppid", DT_IDENT_SCALAR, 0, DIF_VAR_PPID, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_type, "pid_t" },
{ "print", DT_IDENT_ACTFUNC, 0, DT_ACT_PRINT, DT_ATTR_STABCMN, DT_VERS_1_9,
        &dt_idops_func, "void(@)" },
{ "printa", DT_IDENT_ACTFUNC, 0, DT_ACT_PRINTA, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(@, ...)" },
{ "printf", DT_IDENT_ACTFUNC, 0, DT_ACT_PRINTF, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(@, ...)" },
{ "probefunc", DT_IDENT_SCALAR, 0, DIF_VAR_PROBEFUNC,
        DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
{ "probemod", DT_IDENT_SCALAR, 0, DIF_VAR_PROBEMOD,
        DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
{ "probename", DT_IDENT_SCALAR, 0, DIF_VAR_PROBENAME,
        DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
{ "probeprov", DT_IDENT_SCALAR, 0, DIF_VAR_PROBEPROV,
        DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
{ "progenyof", DT_IDENT_FUNC, 0, DIF_SUBR_PROGENYOF,
        DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "int(pid_t)" },
{ "quantize", DT_IDENT_AGGFUNC, 0, DTRACEAGG_QUANTIZE,
        DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(@, ...)" },
{ "raise", DT_IDENT_ACTFUNC, 0, DT_ACT_RAISE, DT_ATTR_STABCMN, DT_VERS_1_0,
        &dt_idops_func, "void(int)" },
```

```
337 { "rand", DT_IDENT_FUNC, 0, DIF_SUBR_RAND, DT_ATTR_STABCMN, DT_VERS_1_0,
338         &dt_idops_func, "int()" },
339 { "rindex", DT_IDENT_FUNC, 0, DIF_SUBR_RINDEX, DT_ATTR_STABCMN, DT_VERS_1_1,
340         &dt_idops_func, "int(const char *, const char *, [int])" },
341 { "rw_iswriter", DT_IDENT_FUNC, 0, DIF_SUBR_RW_ISWRITER,
342         DT_ATTR_EVOLCMN, DT_VERS_1_0,
343         &dt_idops_func, "int(genunix`krwlock_t *)" },
344 { "rw_read_held", DT_IDENT_FUNC, 0, DIF_SUBR_RW_READ_HELD,
345         DT_ATTR_EVOLCMN, DT_VERS_1_0,
346         &dt_idops_func, "int(genunix`krwlock_t *)" },
347 { "rw_write_held", DT_IDENT_FUNC, 0, DIF_SUBR_RW_WRITE_HELD,
348         DT_ATTR_EVOLCMN, DT_VERS_1_0,
349         &dt_idops_func, "int(genunix`krwlock_t *)" },
350 { "self", DT_IDENT_PTR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0,
351         &dt_idops_type, "void" },
352 { "setopt", DT_IDENT_ACTFUNC, 0, DT_ACT_SETOPT, DT_ATTR_STABCMN,
353         DT_VERS_1_2, &dt_idops_func, "void(const char *, [const char *])" },
354 { "speculate", DT_IDENT_ACTFUNC, 0, DT_ACT_SPECULATE,
355         DT_ATTR_STABCMN, DT_VERS_1_0,
356         &dt_idops_func, "void(int)" },
357 { "speculation", DT_IDENT_FUNC, 0, DIF_SUBR_SPECULATION,
358         DT_ATTR_STABCMN, DT_VERS_1_0,
359         &dt_idops_func, "int()" },
360 { "stack", DT_IDENT_ACTFUNC, 0, DT_ACT_STACK, DT_ATTR_STABCMN, DT_VERS_1_0,
361         &dt_idops_func, "stack(...)" },
362 { "stackdepth", DT_IDENT_SCALAR, 0, DIF_VAR_STACKDEPTH,
363         DT_ATTR_STABCMN, DT_VERS_1_0,
364         &dt_idops_type, "uint32_t" },
365 { "stddev", DT_IDENT_AGGFUNC, 0, DTRACEAGG_STDDEV, DT_ATTR_STABCMN,
366         DT_VERS_1_6, &dt_idops_func, "void(@)" },
367 { "stop", DT_IDENT_ACTFUNC, 0, DT_ACT_STOP, DT_ATTR_STABCMN, DT_VERS_1_0,
368         &dt_idops_func, "void()" },
369 { "strchr", DT_IDENT_FUNC, 0, DIF_SUBR_STRCHR, DT_ATTR_STABCMN, DT_VERS_1_1,
370         &dt_idops_func, "string(const char *, char)" },
371 { "strlen", DT_IDENT_FUNC, 0, DIF_SUBR_STRLEN, DT_ATTR_STABCMN, DT_VERS_1_0,
372         &dt_idops_func, "size_t(const char *)" },
373 { "strjoin", DT_IDENT_FUNC, 0, DIF_SUBR_STRJOIN, DT_ATTR_STABCMN, DT_VERS_1_0,
374         &dt_idops_func, "string(const char *, const char *)" },
375 { "strrchr", DT_IDENT_FUNC, 0, DIF_SUBR_STRRCHR, DT_ATTR_STABCMN, DT_VERS_1_1,
376         &dt_idops_func, "string(const char *, char)" },
377 { "strstr", DT_IDENT_FUNC, 0, DIF_SUBR_STRSTR, DT_ATTR_STABCMN, DT_VERS_1_1,
378         &dt_idops_func, "string(const char *, const char *)" },
379 { "strtok", DT_IDENT_FUNC, 0, DIF_SUBR_STRTOK, DT_ATTR_STABCMN, DT_VERS_1_1,
380         &dt_idops_func, "string(const char *, const char *)" },
381 { "substr", DT_IDENT_FUNC, 0, DIF_SUBR_SUBSTR, DT_ATTR_STABCMN, DT_VERS_1_1,
382         &dt_idops_func, "string(const char *, int, [int])" },
383 { "sum", DT_IDENT_AGGFUNC, 0, DTRACEAGG_SUM, DT_ATTR_STABCMN, DT_VERS_1_0,
384         &dt_idops_func, "void(@)" },
385 { "sym", DT_IDENT_ACTFUNC, 0, DT_ACT_SYM, DT_ATTR_STABCMN,
386         DT_VERS_1_2, &dt_idops_func, "_symaddr(uintptr_t)" },
387 { "system", DT_IDENT_ACTFUNC, 0, DT_ACT_SYSTEM, DT_ATTR_STABCMN, DT_VERS_1_0,
388         &dt_idops_func, "void(@, ...)" },
389 { "this", DT_IDENT_PTR, 0, 0, DT_ATTR_STABCMN, DT_VERS_1_0,
390         &dt_idops_type, "void" },
391 { "tid", DT_IDENT_SCALAR, 0, DIF_VAR_TID, DT_ATTR_STABCMN, DT_VERS_1_0,
392         &dt_idops_type, "id_t" },
393 { "timestamp", DT_IDENT_SCALAR, 0, DIF_VAR_TIMESTAMP,
394         DT_ATTR_STABCMN, DT_VERS_1_0,
395         &dt_idops_type, "uint64_t" },
396 { "tolower", DT_IDENT_FUNC, 0, DIF_SUBR_TOLOWER, DT_ATTR_STABCMN, DT_VERS_1_8,
397         &dt_idops_func, "string(const char *)" },
398 { "toupper", DT_IDENT_FUNC, 0, DIF_SUBR_TOUPPER, DT_ATTR_STABCMN, DT_VERS_1_8,
399         &dt_idops_func, "string(const char *)" },
400 { "trace", DT_IDENT_ACTFUNC, 0, DT_ACT_TRACE, DT_ATTR_STABCMN, DT_VERS_1_0,
401         &dt_idops_func, "void(@)" },
402 { "tracemem", DT_IDENT_ACTFUNC, 0, DT_ACT_TRACEMEM,
```



```
403         DT_ATTR_STABCMN, DT_VERS_1_0,
404         &dt_idops_func, "void(@, size_t, ...)" },
405 { "trunc", DT_IDENT_ACTFUNC, 0, DT_ACT_TRUNC, DT_ATTR_STABCMN,
406         DT_VERS_1_0, &dt_idops_func, "void(...)" },
407 { "uaddr", DT_IDENT_ACTFUNC, 0, DT_ACT_UADDR, DT_ATTR_STABCMN,
408         DT_VERS_1_2, &dt_idops_func, "_usymaddr(uintptr_t)" },
409 { "ucaller", DT_IDENT_SCALAR, 0, DIF_VAR_UCALLER, DT_ATTR_STABCMN,
410         DT_VERS_1_2, &dt_idops_type, "uint64_t" },
411 { "ufunc", DT_IDENT_ACTFUNC, 0, DT_ACT_USYM, DT_ATTR_STABCMN,
412         DT_VERS_1_2, &dt_idops_func, "_usymaddr(uintptr_t)" },
413 { "uid", DT_IDENT_SCALAR, 0, DIF_VAR_UID, DT_ATTR_STABCMN, DT_VERS_1_0,
414         &dt_idops_type, "uid_t" },
415 { "umod", DT_IDENT_ACTFUNC, 0, DT_ACT_UMOD, DT_ATTR_STABCMN,
416         DT_VERS_1_2, &dt_idops_func, "_usymaddr(uintptr_t)" },
417 { "uregs", DT_IDENT_ARRAY, 0, DIF_VAR_UREGS, DT_ATTR_STABCMN, DT_VERS_1_0,
418         &dt_idops_regs, NULL },
419 { "ustack", DT_IDENT_ACTFUNC, 0, DT_ACT_USTACK, DT_ATTR_STABCMN, DT_VERS_1_0,
420         &dt_idops_func, "stack(...)" },
421 { "ustackdepth", DT_IDENT_SCALAR, 0, DIF_VAR_USTACKDEPTH,
422         DT_ATTR_STABCMN, DT_VERS_1_2,
423         &dt_idops_type, "uint32_t" },
424 { "usym", DT_IDENT_ACTFUNC, 0, DT_ACT_USYM, DT_ATTR_STABCMN,
425         DT_VERS_1_2, &dt_idops_func, "_usymaddr(uintptr_t)" },
426 { "vmregs", DT_IDENT_ARRAY, 0, DIF_VAR_VMREGS, DT_ATTR_STABCMN, DT_VERS_1_7,
427         &dt_idops_regs, NULL },
428 { "vtimestamp", DT_IDENT_SCALAR, 0, DIF_VAR_VTIMESTAMP,
429         DT_ATTR_STABCMN, DT_VERS_1_0,
430         &dt_idops_type, "uint64_t" },
431 { "walltimestamp", DT_IDENT_SCALAR, 0, DIF_VAR_WALLTIMESTAMP,
432         DT_ATTR_STABCMN, DT_VERS_1_0,
433         &dt_idops_type, "int64_t" },
434 { "zonename", DT_IDENT_SCALAR, 0, DIF_VAR_ZONENAME,
435         DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
436 { NULL, 0, 0, 0, { 0, 0, 0 }, 0, NULL, NULL }
437 };

439 /*
440  * Tables of ILP32 intrinsic integer and floating-point type templates to use
441  * to populate the dynamic "C" CTF type container.
442  */
443 static const dt_intrinsic_t _dtrace_intrinsics_32[] = {
444 { "void", { CTF_INT_SIGNED, 0, 0 }, CTF_K_INTEGER },
445 { "signed", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
446 { "unsigned", { 0, 0, 32 }, CTF_K_INTEGER },
447 { "char", { CTF_INT_SIGNED | CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
448 { "short", { CTF_INT_SIGNED, 0, 16 }, CTF_K_INTEGER },
449 { "int", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
450 { "long", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
451 { "long long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
452 { "signed char", { CTF_INT_SIGNED | CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
453 { "signed short", { CTF_INT_SIGNED, 0, 16 }, CTF_K_INTEGER },
454 { "signed int", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
455 { "signed long", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
456 { "signed long long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
457 { "unsigned char", { CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
458 { "unsigned short", { 0, 0, 16 }, CTF_K_INTEGER },
459 { "unsigned int", { 0, 0, 32 }, CTF_K_INTEGER },
460 { "unsigned long", { 0, 0, 32 }, CTF_K_INTEGER },
461 { "unsigned long long", { 0, 0, 64 }, CTF_K_INTEGER },
462 { "_Bool", { CTF_INT_BOOL, 0, 8 }, CTF_K_INTEGER },
463 { "float", { CTF_FP_SINGLE, 0, 32 }, CTF_K_FLOAT },
464 { "double", { CTF_FP_DOUBLE, 0, 64 }, CTF_K_FLOAT },
465 { "long double", { CTF_FP_LDOUBLE, 0, 128 }, CTF_K_FLOAT },
466 { "float imaginary", { CTF_FP_IMAGRY, 0, 32 }, CTF_K_FLOAT },
467 { "double imaginary", { CTF_FP_DIMAGRY, 0, 64 }, CTF_K_FLOAT },
468 { "long double imaginary", { CTF_FP_LDIMAGRY, 0, 128 }, CTF_K_FLOAT },
```

```
{ "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 LP64 intrinsic integer and floating-point type templates to use
 * to populate the dynamic "C" CTF type container.
 */
static const dt_intrinsic_t _dtrace_intrinsics_64[] = {
{ "void", { CTF_INT_SIGNED, 0, 0 }, CTF_K_INTEGER },
{ "signed", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
{ "unsigned", { 0, 0, 32 }, CTF_K_INTEGER },
{ "char", { CTF_INT_SIGNED | CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
{ "short", { CTF_INT_SIGNED, 0, 16 }, CTF_K_INTEGER },
{ "int", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
{ "long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
{ "long long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
{ "signed char", { CTF_INT_SIGNED | CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
{ "signed short", { CTF_INT_SIGNED, 0, 16 }, CTF_K_INTEGER },
{ "signed int", { CTF_INT_SIGNED, 0, 32 }, CTF_K_INTEGER },
{ "signed long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
{ "signed long long", { CTF_INT_SIGNED, 0, 64 }, CTF_K_INTEGER },
{ "unsigned char", { CTF_INT_CHAR, 0, 8 }, CTF_K_INTEGER },
{ "unsigned short", { 0, 0, 16 }, CTF_K_INTEGER },
{ "unsigned int", { 0, 0, 32 }, CTF_K_INTEGER },
{ "unsigned long", { 0, 0, 64 }, CTF_K_INTEGER },
{ "unsigned long long", { 0, 0, 64 }, CTF_K_INTEGER },
{ "_Bool", { CTF_INT_BOOL, 0, 8 }, CTF_K_INTEGER },
{ "float", { CTF_FP_SINGLE, 0, 32 }, CTF_K_FLOAT },
{ "double", { CTF_FP_DOUBLE, 0, 64 }, CTF_K_FLOAT },
{ "long double", { CTF_FP_LDOUBLE, 0, 128 }, CTF_K_FLOAT },
{ "float imaginary", { CTF_FP_IMAGRY, 0, 32 }, CTF_K_FLOAT },
{ "double imaginary", { CTF_FP_DIMAGRY, 0, 64 }, CTF_K_FLOAT },
{ "long double imaginary", { CTF_FP_LDIMAGRY, 0, 128 }, CTF_K_FLOAT },
{ "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 = {
        DTRACE_STABILITY_MAX,
        DTRACE_STABILITY_MAX,
        DTRACE_CLASS_MAX
};

const dtrace_attribute_t _dtrace_defattr = {
        DTRACE_STABILITY_STABLE,
        DTRACE_STABILITY_STABLE,
        DTRACE_CLASS_COMMON
};

const dtrace_attribute_t _dtrace_symattr = {
        DTRACE_STABILITY_PRIVATE,
        DTRACE_STABILITY_PRIVATE,
        DTRACE_CLASS_UNKNOWN
};

const dtrace_attribute_t _dtrace_typattr = {
        DTRACE_STABILITY_PRIVATE,
        DTRACE_STABILITY_PRIVATE,
        DTRACE_CLASS_UNKNOWN
};

const dtrace_attribute_t _dtrace_prvattr = {
        DTRACE_STABILITY_PRIVATE,
        DTRACE_STABILITY_PRIVATE,
        DTRACE_CLASS_UNKNOWN
};
```



```
const dtrace_pattr_t _dtrace_prvdesc = {
{ DTRACE_STABILITY_UNSTABLE, DTRACE_STABILITY_UNSTABLE, DTRACE_CLASS_COMMON },
{ DTRACE_STABILITY_UNSTABLE, DTRACE_STABILITY_UNSTABLE, DTRACE_CLASS_COMMON },
{ DTRACE_STABILITY_UNSTABLE, DTRACE_STABILITY_UNSTABLE, DTRACE_CLASS_COMMON },
{ DTRACE_STABILITY_UNSTABLE, DTRACE_STABILITY_UNSTABLE, DTRACE_CLASS_COMMON },
{ DTRACE_STABILITY_UNSTABLE, DTRACE_STABILITY_UNSTABLE, DTRACE_CLASS_COMMON },
};

const char *_dtrace_defcpp = "/usr/ccs/lib/cpp"; /* default cpp(1) to invoke */
const char *_dtrace_defld = "/usr/ccs/bin/ld";   /* default ld(1) to invoke */

const char *_dtrace_libdir = "/usr/lib/dtrace"; /* default library directory */
const char *_dtrace_provdir = "/dev/dtrace/provider"; /* provider directory */

int _dtrace_strbuckets = 211;   /* default number of hash buckets (prime) */
int _dtrace_intbuckets = 256;   /* default number of integer buckets (Pof2) */
uint_t _dtrace_strsize = 256;   /* default size of string intrinsic type */
uint_t _dtrace_stkindent = 14;  /* default whitespace indent for stack/ustack */
uint_t _dtrace_pidbuckets = 64; /* default number of pid hash buckets */
uint_t _dtrace_pidlrulim = 8;   /* default number of pid handles to cache */
size_t _dtrace_bufsize = 512;   /* default dt_buf_create() size */
int _dtrace_argmax = 32;        /* default maximum number of probe arguments */

int _dtrace_debug = 0;          /* debug messages enabled (off) */
const char *const _dtrace_version = DT_VERS_STRING; /* API version string */
int _dtrace_rdvers = RD_VERSION; /* rtld_db feature version */

typedef struct dt_fdlist {
        int *df_fds;            /* array of provider driver file descriptors */
        uint_t df_ents;         /* number of valid elements in df_fds[] */
        uint_t df_size;         /* size of df_fds[] */
} dt_fdlist_t;

#pragma init(_dtrace_init)
void
_dtrace_init(void)
{
        _dtrace_debug = getenv("DTRACE_DEBUG") != NULL;

        for (; _dtrace_rdvers > 0; _dtrace_rdvers--) {
                if (rd_init(_dtrace_rdvers) == RD_OK)
                        break;
        }
}

static dtrace_hdl_t *
set_open_errno(dtrace_hdl_t *dtp, int *errp, int err)
{
        if (dtp != NULL)
                dtrace_close(dtp);
        if (errp != NULL)
                *errp = err;
        return (NULL);
}

static void
dt_provmod_open(dt_provmod_t **provmod, dt_fdlist_t *dfp)
{
        dt_provmod_t *prov;
        char path[PATH_MAX];
        struct dirent *dp, *ep;
        DIR *dirp;
        int fd;

        if ((dirp = opendir(_dtrace_provdir)) == NULL)
                return; /* failed to open directory; just skip it */
```

```
734         ep = alloca(sizeof (struct dirent) + PATH_MAX + 1);
735         bzero(ep, sizeof (struct dirent) + PATH_MAX + 1);
736
737         while (readdir_r(dirp, ep, &dp) == 0 && dp != NULL) {
738                 if (dp->d_name[0] == '.')
739                         continue; /* skip "." and ".." */
740
741                 if (dfp->df_ents == dfp->df_size) {
742                         uint_t size = dfp->df_size ? dfp->df_size * 2 : 16;
743                         int *fds = realloc(dfp->df_fds, size * sizeof (int));
744
745                         if (fds == NULL)
746                                 break; /* skip the rest of this directory */
747
748                         dfp->df_fds = fds;
749                         dfp->df_size = size;
750                 }
751
752                 (void) snprintf(path, sizeof (path), "%s/%s",
753                     _dtrace_provdir, dp->d_name);
754
755                 if ((fd = open(path, O_RDONLY)) == -1)
756                         continue; /* failed to open driver; just skip it */
757
758                 if (((prov = malloc(sizeof (dt_provmod_t))) == NULL) ||
759                     (prov->dp_name = malloc(strlen(dp->d_name) + 1)) == NULL) {
760                         free(prov);
761                         (void) close(fd);
762                         break;
763                 }
764
765                 (void) strcpy(prov->dp_name, dp->d_name);
766                 prov->dp_next = *provmod;
767                 *provmod = prov;
768
769                 dt_dprintf("opened provider %s\n", dp->d_name);
770                 dfp->df_fds[dfp->df_ents++] = fd;
771         }
772
773         (void) closedir(dirp);
774 }
775
776 static void
777 dt_provmod_destroy(dt_provmod_t **provmod)
778 {
779         dt_provmod_t *next, *current;
780
781         for (current = *provmod; current != NULL; current = next) {
782                 next = current->dp_next;
783                 free(current->dp_name);
784                 free(current);
785         }
786
787         *provmod = NULL;
788 }
789
790 static const char *
791 dt_get_sysinfo(int cmd, char *buf, size_t len)
792 {
793         ssize_t rv = sysinfo(cmd, buf, len);
794         char *p = buf;
795
796         if (rv < 0 || rv > len)
797                 (void) snprintf(buf, len, "%s", "Unknown");
```



```
799         while ((p = strchr(p, '.')) != NULL)
800                 *p++ = '_';
801
802         return (buf);
803 }
804
805 static dtrace_hdl_t *
806 dt_vopen(int version, int flags, int *errp,
807     const dtrace_vector_t *vector, void *arg)
808 {
809         dtrace_hdl_t *dtp = NULL;
810         int dtfd = -1, ftfd = -1, fterr = 0;
811         dtrace_prog_t *pgp;
812         dt_module_t *dmp;
813         dt_provmod_t *provmod = NULL;
814         int i, err;
815         struct rlimit rl;
816
817         const dt_intrinsic_t *dinp;
818         const dt_typedef_t *dtyp;
819         const dt_ident_t *idp;
820
821         dtrace_typeinfo_t dtt;
822         ctf_funcinfo_t ctc;
823         ctf_arinfo_t ctr;
824
825         dt_fdlist_t df = { NULL, 0, 0 };
826
827         char isadef[32], utsdef[32];
828         char s1[64], s2[64];
829
830         if (version <= 0)
831                 return (set_open_errno(dtp, errp, EINVAL));
832
833         if (version > DTRACE_VERSION)
834                 return (set_open_errno(dtp, errp, EDT_VERSION));
835
836         if (version < DTRACE_VERSION) {
837                 /*
838                  * Currently, increasing the library version number is used to
839                  * denote a binary incompatible change.  That is, a consumer
840                  * of the library cannot run on a version of the library with
841                  * a higher DTRACE_VERSION number than the consumer compiled
842                  * against.  Once the library API has been committed to,
843                  * backwards binary compatibility will be required; at that
844                  * time, this check should change to return EDT_OVERSION only
845                  * if the specified version number is less than the version
846                  * number at the time of interface commitment.
847                  */
848                 return (set_open_errno(dtp, errp, EDT_OVERSION));
849         }
850
851         if (flags & ~DTRACE_O_MASK)
852                 return (set_open_errno(dtp, errp, EINVAL));
853
854         if ((flags & DTRACE_O_LP64) && (flags & DTRACE_O_ILP32))
855                 return (set_open_errno(dtp, errp, EINVAL));
856
857         if (vector == NULL && arg != NULL)
858                 return (set_open_errno(dtp, errp, EINVAL));
859
860         if (elf_version(EV_CURRENT) == EV_NONE)
861                 return (set_open_errno(dtp, errp, EDT_ELFVERSION));
862
863         if (vector != NULL || (flags & DTRACE_O_NODEV))
864                 goto alloc; /* do not attempt to open dtrace device */
```

```
866         /*
867          * Before we get going, crank our limit on file descriptors up to the
868          * hard limit.  This is to allow for the fact that libproc keeps file
869          * descriptors to objects open for the lifetime of the proc handle;
870          * without raising our hard limit, we would have an acceptably small
871          * bound on the number of processes that we could concurrently
872          * instrument with the pid provider.
873          */
874         if (getrlimit(RLIMIT_NOFILE, &rl) == 0) {
875                 rl.rlim_cur = rl.rlim_max;
876                 (void) setrlimit(RLIMIT_NOFILE, &rl);
877         }
878
879         /*
880          * Get the device path of each of the providers.  We hold them open
881          * in the df.df_fds list until we open the DTrace driver itself,
882          * allowing us to see all of the probes provided on this system.  Once
883          * we have the DTrace driver open, we can safely close all the providers
884          * now that they have registered with the framework.
885          */
886         dt_provmod_open(&provmod, &df);
887
888         dtfd = open("/dev/dtrace/dtrace", O_RDWR);
889         err = errno; /* save errno from opening dtfd */
890
891         ftfd = open("/dev/dtrace/provider/fasttrap", O_RDWR);
892         fterr = ftfd == -1 ? errno : 0; /* save errno from open ftfd */
893
894         while (df.df_ents-- != 0)
895                 (void) close(df.df_fds[df.df_ents]);
896
897         free(df.df_fds);
898
899         /*
900          * If we failed to open the dtrace device, fail dtrace_open().
901          * We convert some kernel errnos to custom libdtrace errnos to
902          * improve the resulting message from the usual strerror().
903          */
904         if (dtfd == -1) {
905                 dt_provmod_destroy(&provmod);
906                 switch (err) {
907                 case ENOENT:
908                         err = EDT_NOENT;
909                         break;
910                 case EBUSY:
911                         err = EDT_BUSY;
912                         break;
913                 case EACCES:
914                         err = EDT_ACCESS;
915                         break;
916                 }
917                 return (set_open_errno(dtp, errp, err));
918         }
919
920         (void) fcntl(dtfd, F_SETFD, FD_CLOEXEC);
921         (void) fcntl(ftfd, F_SETFD, FD_CLOEXEC);
922
923 alloc:
924         if ((dtp = malloc(sizeof (dtrace_hdl_t))) == NULL)
925                 return (set_open_errno(dtp, errp, EDT_NOMEM));
926
927         bzero(dtp, sizeof (dtrace_hdl_t));
928         dtp->dt_oflags = flags;
929         dtp->dt_prcmode = DT_PROC_STOP_PREINIT;
930         dtp->dt_linkmode = DT_LINK_KERNEL;
```



```
931         dtp->dt_linktype = DT_LTYP_ELF;
932         dtp->dt_xlatemode = DT_XL_STATIC;
933         dtp->dt_stdcmode = DT_STDC_XA;
934         dtp->dt_version = version;
935         dtp->dt_fd = dtfd;
936         dtp->dt_ftfd = ftfd;
937         dtp->dt_fterr = fterr;
938         dtp->dt_cdefs_fd = -1;
939         dtp->dt_ddefs_fd = -1;
940         dtp->dt_stdout_fd = -1;
941         dtp->dt_modbuckets = _dtrace_strbuckets;
942         dtp->dt_mods = calloc(dtp->dt_modbuckets, sizeof (dt_module_t *));
943         dtp->dt_provbuckets = _dtrace_strbuckets;
944         dtp->dt_provs = calloc(dtp->dt_provbuckets, sizeof (dt_provider_t *));
945         dt_proc_init(dtp);
946         dtp->dt_vmax = DT_VERS_LATEST;
947         dtp->dt_cpp_path = strdup(_dtrace_defcpp);
948         dtp->dt_cpp_argv = malloc(sizeof (char *));
949         dtp->dt_cpp_argc = 1;
950         dtp->dt_cpp_args = 1;
951         dtp->dt_ld_path = strdup(_dtrace_defld);
952         dtp->dt_provmod = provmod;
953         dtp->dt_vector = vector;
954         dtp->dt_varg = arg;
955         dt_dof_init(dtp);
956         (void) uname(&dtp->dt_uts);
957
958         if (dtp->dt_mods == NULL || dtp->dt_provs == NULL ||
959             dtp->dt_procs == NULL || dtp->dt_proc_env == NULL ||
960             dtp->dt_ld_path == NULL || dtp->dt_cpp_path == NULL ||
961             dtp->dt_cpp_argv == NULL)
962                 return (set_open_errno(dtp, errp, EDT_NOMEM));
963
964         for (i = 0; i < DTRACEOPT_MAX; i++)
965                 dtp->dt_options[i] = DTRACEOPT_UNSET;
966
967         dtp->dt_cpp_argv[0] = (char *)strbasename(dtp->dt_cpp_path);
968
969         (void) snprintf(isadef, sizeof (isadef), "-D__SUNW_D_%u",
970             (uint_t)(sizeof (void *) * NBBY));
971
972         (void) snprintf(utsdef, sizeof (utsdef), "-D__%s_%s",
973             dt_get_sysinfo(SI_SYSNAME, s1, sizeof (s1)),
974             dt_get_sysinfo(SI_RELEASE, s2, sizeof (s2)));
975
976         if (dt_cpp_add_arg(dtp, "-D__sun") == NULL ||
977             dt_cpp_add_arg(dtp, "-D__unix") == NULL ||
978             dt_cpp_add_arg(dtp, "-D__SVR4") == NULL ||
979             dt_cpp_add_arg(dtp, "-D__SUNW_D=1") == NULL ||
980             dt_cpp_add_arg(dtp, isadef) == NULL ||
981             dt_cpp_add_arg(dtp, utsdef) == NULL)
982                 return (set_open_errno(dtp, errp, EDT_NOMEM));
983
984         if (flags & DTRACE_O_NODEV)
985                 bcopy(&_dtrace_conf, &dtp->dt_conf, sizeof (_dtrace_conf));
986         else if (dt_ioctl(dtp, DTRACEIOC_CONF, &dtp->dt_conf) != 0)
987                 return (set_open_errno(dtp, errp, errno));
988
989         if (flags & DTRACE_O_LP64)
990                 dtp->dt_conf.dtc_ctfmodel = CTF_MODEL_LP64;
991         else if (flags & DTRACE_O_ILP32)
992                 dtp->dt_conf.dtc_ctfmodel = CTF_MODEL_ILP32;
993
994 #ifdef __sparc
995         /*
996          * On SPARC systems, __sparc is always defined for <sys/isa_defs.h>
```

```
 997          * and __sparcv9 is defined if we are doing a 64-bit compile.
 998          */
 999         if (dt_cpp_add_arg(dtp, "-D__sparc") == NULL)
1000                 return (set_open_errno(dtp, errp, EDT_NOMEM));

1002         if (dtp->dt_conf.dtc_ctfmodel == CTF_MODEL_LP64 &&
1003             dt_cpp_add_arg(dtp, "-D__sparcv9") == NULL)
1004                 return (set_open_errno(dtp, errp, EDT_NOMEM));
1005 #endif

1007 #ifdef __x86
1008         /*
1009          * On x86 systems, __i386 is defined for <sys/isa_defs.h> for 32-bit
1010          * compiles and __amd64 is defined for 64-bit compiles.  Unlike SPARC,
1011          * they are defined exclusive of one another (see PSARC 2004/619).
1012          */
1013         if (dtp->dt_conf.dtc_ctfmodel == CTF_MODEL_LP64) {
1014                 if (dt_cpp_add_arg(dtp, "-D__amd64") == NULL)
1015                         return (set_open_errno(dtp, errp, EDT_NOMEM));
1016         } else {
1017                 if (dt_cpp_add_arg(dtp, "-D__i386") == NULL)
1018                         return (set_open_errno(dtp, errp, EDT_NOMEM));
1019         }
1020 #endif

1022         if (dtp->dt_conf.dtc_difversion < DIF_VERSION)
1023                 return (set_open_errno(dtp, errp, EDT_DIFVERS));

1025         if (dtp->dt_conf.dtc_ctfmodel == CTF_MODEL_ILP32)
1026                 bcopy(_dtrace_ints_32, dtp->dt_ints, sizeof (_dtrace_ints_32));
1027         else
1028                 bcopy(_dtrace_ints_64, dtp->dt_ints, sizeof (_dtrace_ints_64));

1030         dtp->dt_macros = dt_idhash_create("macro", NULL, 0, UINT_MAX);
1031         dtp->dt_aggs = dt_idhash_create("aggregation", NULL,
1032             DTRACE_AGGVARIDNONE + 1, UINT_MAX);

1034         dtp->dt_globals = dt_idhash_create("global", _dtrace_globals,
1035             DIF_VAR_OTHER_UBASE, DIF_VAR_OTHER_MAX);

1037         dtp->dt_tls = dt_idhash_create("thread local", NULL,
1038             DIF_VAR_OTHER_UBASE, DIF_VAR_OTHER_MAX);

1040         if (dtp->dt_macros == NULL || dtp->dt_aggs == NULL ||
1041             dtp->dt_globals == NULL || dtp->dt_tls == NULL)
1042                 return (set_open_errno(dtp, errp, EDT_NOMEM));

1044         /*
1045          * Populate the dt_macros identifier hash table by hand: we can't use
1046          * the dt_idhash_populate() mechanism because we're not yet compiling
1047          * and dtrace_update() needs to immediately reference these idents.
1048          */
1049         for (idp = _dtrace_macros; idp->di_name != NULL; idp++) {
1050                 if (dt_idhash_insert(dtp->dt_macros, idp->di_name,
1051                     idp->di_kind, idp->di_flags, idp->di_id, idp->di_attr,
1052                     idp->di_vers, idp->di_ops ? idp->di_ops : &dt_idops_thaw,
1053                     idp->di_iarg, 0) == NULL)
1054                         return (set_open_errno(dtp, errp, EDT_NOMEM));
1055         }

1057         /*
1058          * Update the module list using /system/object and load the values for
1059          * the macro variable definitions according to the current process.
1060          */
1061         dtrace_update(dtp);
```



```
1063         /*
1064          * Select the intrinsics and typedefs we want based on the data model.
1065          * The intrinsics are under "C".  The typedefs are added under "D".
1066          */
1067         if (dtp->dt_conf.dtc_ctfmodel == CTF_MODEL_ILP32) {
1068                 dinp = _dtrace_intrinsics_32;
1069                 dtyp = _dtrace_typedefs_32;
1070         } else {
1071                 dinp = _dtrace_intrinsics_64;
1072                 dtyp = _dtrace_typedefs_64;
1073         }

1075         /*
1076          * Create a dynamic CTF container under the "C" scope for intrinsic
1077          * types and types defined in ANSI-C header files that are included.
1078          */
1079         if ((dmp = dtp->dt_cdefs = dt_module_create(dtp, "C")) == NULL)
1080                 return (set_open_errno(dtp, errp, EDT_NOMEM));

1082         if ((dmp->dm_ctfp = ctf_create(&dtp->dt_ctferr)) == NULL)
1083                 return (set_open_errno(dtp, errp, EDT_CTF));

1085         dt_dprintf("created CTF container for %s (%p)\n",
1086             dmp->dm_name, (void *)dmp->dm_ctfp);

1088         (void) ctf_setmodel(dmp->dm_ctfp, dtp->dt_conf.dtc_ctfmodel);
1089         ctf_setspecific(dmp->dm_ctfp, dmp);

1091         dmp->dm_flags = DT_DM_LOADED; /* fake up loaded bit */
1092         dmp->dm_modid = -1; /* no module ID */

1094         /*
1095          * Fill the dynamic "C" CTF container with all of the intrinsic
1096          * integer and floating-point types appropriate for this data model.
1097          */
1098         for (; dinp->din_name != NULL; dinp++) {
1099                 if (dinp->din_kind == CTF_K_INTEGER) {
1100                         err = ctf_add_integer(dmp->dm_ctfp, CTF_ADD_ROOT,
1101                             dinp->din_name, &dinp->din_data);
1102                 } else {
1103                         err = ctf_add_float(dmp->dm_ctfp, CTF_ADD_ROOT,
1104                             dinp->din_name, &dinp->din_data);
1105                 }

1107                 if (err == CTF_ERR) {
1108                         dt_dprintf("failed to add %s to C container: %s\n",
1109                             dinp->din_name, ctf_errmsg(
1110                             ctf_errno(dmp->dm_ctfp)));
1111                         return (set_open_errno(dtp, errp, EDT_CTF));
1112                 }
1113         }

1115         if (ctf_update(dmp->dm_ctfp) != 0) {
1116                 dt_dprintf("failed to update C container: %s\n",
1117                     ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1118                 return (set_open_errno(dtp, errp, EDT_CTF));
1119         }

1121         /*
1122          * Add intrinsic pointer types that are needed to initialize printf
1123          * format dictionary types (see table in dt_printf.c).
1124          */
1125         (void) ctf_add_pointer(dmp->dm_ctfp, CTF_ADD_ROOT,
1126             ctf_lookup_by_name(dmp->dm_ctfp, "void"));

1128         (void) ctf_add_pointer(dmp->dm_ctfp, CTF_ADD_ROOT,
```

```
1129            ctf_lookup_by_name(dmp->dm_ctfp, "char"));

1131        (void) ctf_add_pointer(dmp->dm_ctfp, CTF_ADD_ROOT,
1132            ctf_lookup_by_name(dmp->dm_ctfp, "int"));

1134        if (ctf_update(dmp->dm_ctfp) != 0) {
1135                dt_dprintf("failed to update C container: %s\n",
1136                    ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1137                return (set_open_errno(dtp, errp, EDT_CTF));
1138        }

1140        /*
1141         * Create a dynamic CTF container under the "D" scope for types that
1142         * are defined by the D program itself or on-the-fly by the D compiler.
1143         * The "D" CTF container is a child of the "C" CTF container.
1144         */
1145        if ((dmp = dtp->dt_ddefs = dt_module_create(dtp, "D")) == NULL)
1146                return (set_open_errno(dtp, errp, EDT_NOMEM));

1148        if ((dmp->dm_ctfp = ctf_create(&dtp->dt_ctferr)) == NULL)
1149                return (set_open_errno(dtp, errp, EDT_CTF));

1151        dt_dprintf("created CTF container for %s (%p)\n",
1152            dmp->dm_name, (void *)dmp->dm_ctfp);

1154        (void) ctf_setmodel(dmp->dm_ctfp, dtp->dt_conf.dtc_ctfmodel);
1155        ctf_setspecific(dmp->dm_ctfp, dmp);

1157        dmp->dm_flags = DT_DM_LOADED; /* fake up loaded bit */
1158        dmp->dm_modid = -1; /* no module ID */

1160        if (ctf_import(dmp->dm_ctfp, dtp->dt_cdefs->dm_ctfp) == CTF_ERR) {
1161                dt_dprintf("failed to import D parent container: %s\n",
1162                    ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1163                return (set_open_errno(dtp, errp, EDT_CTF));
1164        }

1166        /*
1167         * Fill the dynamic "D" CTF container with all of the built-in typedefs
1168         * that we need to use for our D variable and function definitions.
1169         * This ensures that basic inttypes.h names are always available to us.
1170         */
1171        for (; dtyp->dty_src != NULL; dtyp++) {
1172                if (ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1173                    dtyp->dty_dst, ctf_lookup_by_name(dmp->dm_ctfp,
1174                    dtyp->dty_src)) == CTF_ERR) {
1175                        dt_dprintf("failed to add typedef %s %s to D "
1176                            "container: %s", dtyp->dty_src, dtyp->dty_dst,
1177                            ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1178                        return (set_open_errno(dtp, errp, EDT_CTF));
1179                }
1180        }

1182        /*
1183         * Insert a CTF ID corresponding to a pointer to a type of kind
1184         * CTF_K_FUNCTION we can use in the compiler for function pointers.
1185         * CTF treats all function pointers as "int (*)()" so we only need one.
1186         */
1187        ctc.ctc_return = ctf_lookup_by_name(dmp->dm_ctfp, "int");
1188        ctc.ctc_argc = 0;
1189        ctc.ctc_flags = 0;

1191        dtp->dt_type_func = ctf_add_function(dmp->dm_ctfp,
1192            CTF_ADD_ROOT, &ctc, NULL);

1194        dtp->dt_type_fptr = ctf_add_pointer(dmp->dm_ctfp,
```



```
1195            CTF_ADD_ROOT, dtp->dt_type_func);

1197        /*
1198         * We also insert CTF definitions for the special D intrinsic types
1199         * string and <DYN> into the D container.  The string type is added
1200         * as a typedef of char[n].  The <DYN> type is an alias for void.
1201         * We compare types to these special CTF ids throughout the compiler.
1202         */
1203        ctr.ctr_contents = ctf_lookup_by_name(dmp->dm_ctfp, "char");
1204        ctr.ctr_index = ctf_lookup_by_name(dmp->dm_ctfp, "long");
1205        ctr.ctr_nelems = _dtrace_strsize;

1207        dtp->dt_type_str = ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1208            "string", ctf_add_array(dmp->dm_ctfp, CTF_ADD_ROOT, &ctr));

1210        dtp->dt_type_dyn = ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1211            "<DYN>", ctf_lookup_by_name(dmp->dm_ctfp, "void"));

1213        dtp->dt_type_stack = ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1214            "stack", ctf_lookup_by_name(dmp->dm_ctfp, "void"));

1216        dtp->dt_type_symaddr = ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1217            "_symaddr", ctf_lookup_by_name(dmp->dm_ctfp, "void"));

1219        dtp->dt_type_usymaddr = ctf_add_typedef(dmp->dm_ctfp, CTF_ADD_ROOT,
1220            "_usymaddr", ctf_lookup_by_name(dmp->dm_ctfp, "void"));

1222        if (dtp->dt_type_func == CTF_ERR || dtp->dt_type_fptr == CTF_ERR ||
1223            dtp->dt_type_str == CTF_ERR || dtp->dt_type_dyn == CTF_ERR ||
1224            dtp->dt_type_stack == CTF_ERR || dtp->dt_type_symaddr == CTF_ERR ||
1225            dtp->dt_type_usymaddr == CTF_ERR) {
1226                dt_dprintf("failed to add intrinsic to D container: %s\n",
1227                    ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1228                return (set_open_errno(dtp, errp, EDT_CTF));
1229        }

1231        if (ctf_update(dmp->dm_ctfp) != 0) {
1232                dt_dprintf("failed update D container: %s\n",
1233                    ctf_errmsg(ctf_errno(dmp->dm_ctfp)));
1234                return (set_open_errno(dtp, errp, EDT_CTF));
1235        }

1237        /*
1238         * Initialize the integer description table used to convert integer
1239         * constants to the appropriate types.  Refer to the comments above
1240         * dt_node_int() for a complete description of how this table is used.
1241         */
1242        for (i = 0; i < sizeof (dtp->dt_ints) / sizeof (dtp->dt_ints[0]); i++) {
1243                if (dtrace_lookup_by_type(dtp, DTRACE_OBJ_EVERY,
1244                    dtp->dt_ints[i].did_name, &dtt) != 0) {
1245                        dt_dprintf("failed to lookup integer type %s: %s\n",
1246                            dtp->dt_ints[i].did_name,
1247                            dtrace_errmsg(dtp, dtrace_errno(dtp)));
1248                        return (set_open_errno(dtp, errp, dtp->dt_errno));
1249                }
1250                dtp->dt_ints[i].did_ctfp = dtt.dtt_ctfp;
1251                dtp->dt_ints[i].did_type = dtt.dtt_type;
1252        }

1254        /*
1255         * Now that we've created the "C" and "D" containers, move them to the
1256         * start of the module list so that these types and symbols are found
1257         * first (for stability) when iterating through the module list.
1258         */
1259        dt_list_delete(&dtp->dt_modlist, dtp->dt_ddefs);
1260        dt_list_prepend(&dtp->dt_modlist, dtp->dt_ddefs);
```

```
1262         dt_list_delete(&dtp->dt_modlist, dtp->dt_cdefs);
1263         dt_list_prepend(&dtp->dt_modlist, dtp->dt_cdefs);

1265         if (dt_pfdict_create(dtp) == -1)
1266                 return (set_open_errno(dtp, errp, dtp->dt_errno));

1268         /*
1269          * If we are opening libdtrace DTRACE_O_NODEV enable C_ZDEFS by default
1270          * because without /dev/dtrace open, we will not be able to load the
1271          * names and attributes of any providers or probes from the kernel.
1272          */
1273         if (flags & DTRACE_O_NODEV)
1274                 dtp->dt_cflags |= DTRACE_C_ZDEFS;

1276         /*
1277          * Load hard-wired inlines into the definition cache by calling the
1278          * compiler on the raw definition string defined above.
1279          */
1280         if ((pgp = dtrace_program_strcompile(dtp, _dtrace_hardwire,
1281             DTRACE_PROBESPEC_NONE, DTRACE_C_EMPTY, 0, NULL)) == NULL) {
1282                 dt_dprintf("failed to load hard-wired definitions: %s\n",
1283                     dtrace_errmsg(dtp, dtrace_errno(dtp)));
1284                 return (set_open_errno(dtp, errp, EDT_HARDWIRE));
1285         }

1287         dtrace_program_destroy(dtp, pgp);

1289         /*
1290          * Set up the default DTrace library path.  Once set, the next call to
1291          * dt_compile() will compile all the libraries.  We intentionally defer
1292          * library processing to improve overhead for clients that don't ever
1293          * compile, and to provide better error reporting (because the full
1294          * reporting of compiler errors requires dtrace_open() to succeed).
1295          */
1296         if (dtrace_setopt(dtp, "libdir", _dtrace_libdir) != 0)
1297                 return (set_open_errno(dtp, errp, dtp->dt_errno));

1299         return (dtp);
1300 }

1302 dtrace_hdl_t *
1303 dtrace_open(int version, int flags, int *errp)
1304 {
1305         return (dt_vopen(version, flags, errp, NULL, NULL));
1306 }

1308 dtrace_hdl_t *
1309 dtrace_vopen(int version, int flags, int *errp,
1310     const dtrace_vector_t *vector, void *arg)
1311 {
1312         return (dt_vopen(version, flags, errp, vector, arg));
1313 }

1315 void
1316 dtrace_close(dtrace_hdl_t *dtp)
1317 {
1318         dt_ident_t *idp, *ndp;
1319         dt_module_t *dmp;
1320         dt_provider_t *pvp;
1321         dtrace_prog_t *pgp;
1322         dt_xlator_t *dxp;
1323         dt_dirpath_t *dirp;
1324         int i;

1326         if (dtp->dt_procs != NULL)
```



```
1327                 dt_proc_fini(dtp);

1329         while ((pgp = dt_list_next(&dtp->dt_programs)) != NULL)
1330                 dt_program_destroy(dtp, pgp);

1332         while ((dxp = dt_list_next(&dtp->dt_xlators)) != NULL)
1333                 dt_xlator_destroy(dtp, dxp);

1335         dt_free(dtp, dtp->dt_xlatormap);

1337         for (idp = dtp->dt_externs; idp != NULL; idp = ndp) {
1338                 ndp = idp->di_next;
1339                 dt_ident_destroy(idp);
1340         }

1342         if (dtp->dt_macros != NULL)
1343                 dt_idhash_destroy(dtp->dt_macros);
1344         if (dtp->dt_aggs != NULL)
1345                 dt_idhash_destroy(dtp->dt_aggs);
1346         if (dtp->dt_globals != NULL)
1347                 dt_idhash_destroy(dtp->dt_globals);
1348         if (dtp->dt_tls != NULL)
1349                 dt_idhash_destroy(dtp->dt_tls);

1351         while ((dmp = dt_list_next(&dtp->dt_modlist)) != NULL)
1352                 dt_module_destroy(dtp, dmp);

1354         while ((pvp = dt_list_next(&dtp->dt_provlist)) != NULL)
1355                 dt_provider_destroy(dtp, pvp);

1357         if (dtp->dt_fd != -1)
1358                 (void) close(dtp->dt_fd);
1359         if (dtp->dt_ftfd != -1)
1360                 (void) close(dtp->dt_ftfd);
1361         if (dtp->dt_cdefs_fd != -1)
1362                 (void) close(dtp->dt_cdefs_fd);
1363         if (dtp->dt_ddefs_fd != -1)
1364                 (void) close(dtp->dt_ddefs_fd);
1365         if (dtp->dt_stdout_fd != -1)
1366                 (void) close(dtp->dt_stdout_fd);

1368         dt_epid_destroy(dtp);
1369         dt_aggid_destroy(dtp);
1370         dt_format_destroy(dtp);
1371         dt_strdata_destroy(dtp);
1372         dt_buffered_destroy(dtp);
1373         dt_aggregate_destroy(dtp);
1374         dt_pfdict_destroy(dtp);
1375         dt_provmod_destroy(&dtp->dt_provmod);
1376         dt_dof_fini(dtp);

1378         for (i = 1; i < dtp->dt_cpp_argc; i++)
1379                 free(dtp->dt_cpp_argv[i]);

1381         while ((dirp = dt_list_next(&dtp->dt_lib_path)) != NULL) {
1382                 dt_list_delete(&dtp->dt_lib_path, dirp);
1383                 free(dirp->dir_path);
1384                 free(dirp);
1385         }

1387         free(dtp->dt_cpp_argv);
1388         free(dtp->dt_cpp_path);
1389         free(dtp->dt_ld_path);

1391         free(dtp->dt_mods);
1392         free(dtp->dt_provs);
```

```
1393         free(dtp);
1394 }

1396 int
1397 dtrace_provider_modules(dtrace_hdl_t *dtp, const char **mods, int nmods)
1398 {
1399         dt_provmod_t *prov;
1400         int i = 0;

1402         for (prov = dtp->dt_provmod; prov != NULL; prov = prov->dp_next, i++) {
1403                 if (i < nmods)
1404                         mods[i] = prov->dp_name;
1405         }

1407         return (i);
1408 }

1410 int
1411 dtrace_ctlfd(dtrace_hdl_t *dtp)
1412 {
1413         return (dtp->dt_fd);
1414 }
```

****
14492 Sat Apr 12 11:18:54 2014
new/usr/src/lib/libdtrace/common/ip.d.in
3903 DTrace SCTP Provider
****
_____***unchanged_portion_omitted_***

```
171 /*
172  * void_ip_t is a void pointer to either an IPv4 or IPv6 header.  It has
173  * its own type name so that a translator can be determined.
174  */
175 typedef uintptr_t void_ip_t;

177 /*
178  * __dtrace_ipsr_ill_t is used by the translator to take an ill_t plus an
179  * additional arg6 from the ip:::send and ip:::receive probes, and translate
179  * additional arg6 from the ip:::send and ip:::recieve probes, and translate
180  * them to an ifinfo_t.
181  */
182 typedef ill_t __dtrace_ipsr_ill_t;

184 /*
185  * __dtrace_tcp_void_ip_t is used by the translator to take either the
186  * non-NULL void_ip_t * passed in or, if it is NULL, uses arg3 (tcp_t *)
187  * from the tcp:::send and tcp:::receive probes to translate to an ipinfo_t.
187  * from the tcp:::send and tcp:::recieve probes to translate to an ipinfo_t.
188  * When no headers are available in the TCP fusion case for tcp:::send
189  * and tcp:::receive case, this allows us to present the consumer with header
190  * data based on the tcp_t * content in order to hide the implementation
191  * details of TCP fusion.
192  */
193 typedef void * __dtrace_tcp_void_ip_t;

195 #pragma D binding "1.5" translator
196 translator pktinfo_t < mblk_t *M > {
197         pkt_addr = NULL;
198 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    3225 Sat Apr 12 11:18:54 2014
new/usr/src/lib/libdtrace/common/sctp.d.in
3903 DTrace SCTP Provider
**********************************************************
```

```
/*
 * 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 David Hoeppner.  All rights reserved.
 */

#pragma D depends_on module unix
#pragma D depends_on provider sctp

/*
 * SCTP connection states.
 */
inline int32_t SCTP_STATE_IDLE = @SCTPS_IDLE@;
#pragma D binding "1.9.2" SCTP_STATE_IDLE
inline int32_t SCTP_STATE_BOUND = @SCTPS_BOUND@;
#pragma D binding "1.9.2" SCTP_STATE_BOUND
inline int32_t SCTP_STATE_LISTEN = @SCTPS_LISTEN@;
#pragma D binding "1.9.2" SCTP_STATE_LISTEN
inline int32_t SCTP_STATE_COOKIE_WAIT = @SCTPS_COOKIE_WAIT@;
#pragma D binding "1.9.2" SCTP_STATE_COOKIE_WAIT
inline int32_t SCTP_STATE_COOKIE_ECHOED = @SCTPS_COOKIE_ECHOED@;
#pragma D binding "1.9.2" SCTP_STATE_COOKIE_ECHOED
inline int32_t SCTP_STATE_ESTABLISHED = @SCTPS_ESTABLISHED@;
#pragma D binding "1.9.2" SCTP_STATE_ESTABLISHED
inline int32_t SCTP_STATE_SHUTDOWN_PENDING = @SCTPS_SHUTDOWN_PENDING@;
#pragma D binding "1.9.2" SCTP_STATE_SHUTDOWN_PENDING
inline int32_t SCTP_STATE_SHUTDOWN_SENT = @SCTPS_SHUTDOWN_SENT@;
#pragma D binding "1.9.2" SCTP_STATE_SHUTDOWN_SENT
inline int32_t SCTP_STATE_SHUTDOWN_RECEIVED = @SCTPS_SHUTDOWN_RECEIVED@;
#pragma D binding "1.9.2" SCTP_STATE_SHUTDOWN_RECEIVED
inline int32_t SCTP_STATE_SHUTDOWN_ACK_SENT = @SCTPS_SHUTDOWN_ACK_SENT@;
#pragma D binding "1.9.2" SCTP_STATE_SHUTDOWN_ACK_SENT

/*
 * Convert a SCTP state value to a string.
 */
inline string sctp_state_string[int32_t state] =
        state == SCTP_STATE_IDLE ? "state-idle" :
        state == SCTP_STATE_BOUND ? "state-bound" :
        state == SCTP_STATE_LISTEN ? "state-listen" :
        state == SCTP_STATE_COOKIE_WAIT ? "state-cookie-wait" :
        state == SCTP_STATE_COOKIE_ECHOED ? "state-cookie-echoed" :
        state == SCTP_STATE_ESTABLISHED ? "state-established" :
        state == SCTP_STATE_SHUTDOWN_PENDING ? "state-shutdown-pending" :
        state == SCTP_STATE_SHUTDOWN_SENT ? "state-shutdown-sent" :
        state == SCTP_STATE_SHUTDOWN_RECEIVED ? "state-shutdown-received" :
        state == SCTP_STATE_SHUTDOWN_ACK_SENT ? "state-shutdown-ack-sent" :
        "<unknown>";
#pragma D binding "1.9.2" sctp_state_string

/*
 * sctpinfo is the SCTP header fields.
```



```
 */
typedef struct sctpinfo {
        uint16_t        sctp_sport;     /* source port */
        uint16_t        sctp_dport;     /* destination port */
        uint32_t        sctp_verify;    /* verification tag */
        uint32_t        sctp_checksum;  /* headers + data checksum */
        sctp_chunk_hdr_t sctp_chunk_hdr;
        sctp_hdr_t      *sctp_hdr;      /* raw SCTP header */
} sctpinfo_t;

/*
 * sctpsinfo sctp state info.
 */
typedef struct sctpsinfo {
        string  sctps_laddr;            /* local address */
        string  sctps_raddr;            /* remote address */
        int32_t sctps_state;            /* connection state */
} sctpsinfo_t;

#pragma D binding "1.9.2" translator
translator sctpinfo_t < sctp_hdr_t *S > {
        sctp_sport = ntohs(S->sh_sport);
        sctp_dport = ntohs(S->sh_dport);
        sctp_verify = ntohl(S->sh_verf);
        sctp_checksum = ntohl(S->sh_chksum);
        sctp_hdr = S;
};
#endif /* ! codereview */
```

****
562 Sat Apr 12 11:18:55 2014
new/usr/src/lib/libdtrace/common/sctp.sed.in
3903 DTrace SCTP Provider
****

```
/*
 * 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 David Hoeppner.  All rights reserved.
 */

#include <netinet/sctp.h>

#define SED_REPLACE(x)  s/#x/x/g

SED_REPLACE(SCTPS_IDLE)
#endif /* ! codereview */
```

****
10644 Sat Apr 12 11:18:55 2014
new/usr/src/lib/libdtrace/common/tcp.d.in
3903 DTrace SCTP Provider
****
_____***unchanged_portion_omitted_***

```
 145 /*
 146  * __dtrace_tcp_tcph_t is used by the tcpinfo_t * translator to take either
 147  * the non-NULL tcph_t * passed in or, if it is NULL, uses arg3 (tcp_t *)
 148  * from the tcp:::send and tcp:::receive probes and translates the tcp_t *
 148  * from the tcp:::send and tcp:::recieve probes and translates the tcp_t *
 149  * into the tcpinfo_t.  When no headers are available - as is the case for
 150  * TCP fusion tcp:::send and tcp:::receive - this allows us to present the
 151  * consumer with header data based on tcp_t * content and hide TCP fusion
 152  * implementation details.
 153  */
 154 typedef tcph_t * __dtrace_tcp_tcph_t;

 156 #pragma D binding "1.6.3" translator
 157 translator tcpinfo_t < tcph_t *T > {
 158         tcp_sport = ntohs(*(uint16_t *)T->th_lport);
 159         tcp_dport = ntohs(*(uint16_t *)T->th_fport);
 160         tcp_seq = ntohl(*(uint32_t *)T->th_seq);
 161         tcp_ack = ntohl(*(uint32_t *)T->th_ack);
 162         tcp_offset = (*(uint8_t *)T->th_offset_and_rsrvd & 0xf0) >> 2;
 163         tcp_flags = *(uint8_t *)T->th_flags;
 164         tcp_window = ntohs(*(uint16_t *)T->th_win);
 165         tcp_checksum = ntohs(*(uint16_t *)T->th_sum);
 166         tcp_urgent = ntohs(*(uint16_t *)T->th_urp);
 167         tcp_hdr = T;
 168 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   27077 Sat Apr 12 11:18:55 2014
new/usr/src/pkg/manifests/developer-dtrace.mf
3903 DTrace SCTP Provider
**********************************************************
#
# 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) 2010, Oracle and/or its affiliates. All rights reserved.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
#

set name=pkg.fmri value=pkg:/developer/dtrace@$(PKGVERS)
set name=pkg.description value="Dynamic Tracing (DTrace) Clients"
set name=pkg.summary value="DTrace Clients"
set name=info.classification \
    value=org.opensolaris.category.2008:Development/System
set name=variant.arch value=$(ARCH)
dir path=usr group=sys
dir path=usr/demo
dir path=usr/demo/dtrace
dir path=usr/include
dir path=usr/include/sys
dir path=usr/lib
dir path=usr/lib/$(ARCH64)
dir path=usr/lib/devfsadm group=sys
dir path=usr/lib/devfsadm/linkmod group=sys
dir path=usr/lib/dtrace
dir path=usr/lib/dtrace/64
dir path=usr/lib/mdb group=sys
dir path=usr/lib/mdb/kvm group=sys
dir path=usr/lib/mdb/kvm/$(ARCH64) group=sys
dir path=usr/lib/mdb/raw group=sys
dir path=usr/lib/mdb/raw/$(ARCH64) group=sys
dir path=usr/sbin
dir path=usr/sbin/$(ARCH32)
dir path=usr/sbin/$(ARCH64)
dir path=usr/share
dir path=usr/share/lib
dir path=usr/share/lib/java group=sys
dir path=usr/share/lib/java/javadoc group=other
dir path=usr/share/lib/java/javadoc/dtrace group=other
dir path=usr/share/lib/java/javadoc/dtrace/api group=other
dir path=usr/share/lib/java/javadoc/dtrace/api/org group=other
```



```
dir path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris group=other
dir path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os group=other
dir path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace \
    group=other
dir \
    path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
    group=other
dir path=usr/share/lib/java/javadoc/dtrace/api/resources group=other
dir path=usr/share/lib/java/javadoc/dtrace/examples group=other
dir path=usr/share/lib/java/javadoc/dtrace/html group=other
dir path=usr/share/lib/java/javadoc/dtrace/images group=other
dir path=usr/share/man/man1m
dir path=usr/share/man/man3lib
file path=usr/demo/dtrace/applicat.d
file path=usr/demo/dtrace/badopen.d
file path=usr/demo/dtrace/begin.d
file path=usr/demo/dtrace/callout.d
file path=usr/demo/dtrace/clause.d
file path=usr/demo/dtrace/clear.d
file path=usr/demo/dtrace/countdown.d
file path=usr/demo/dtrace/counter.d
file path=usr/demo/dtrace/dateprof.d
file path=usr/demo/dtrace/delay.d
file path=usr/demo/dtrace/denorm.d
file path=usr/demo/dtrace/end.d
file path=usr/demo/dtrace/error.d
file path=usr/demo/dtrace/errorpath.d
file path=usr/demo/dtrace/find.d
file path=usr/demo/dtrace/firebird.d
file path=usr/demo/dtrace/hello.d
file path=usr/demo/dtrace/howlong.d
file path=usr/demo/dtrace/index.html
file path=usr/demo/dtrace/interp.d
file path=usr/demo/dtrace/interval.d
file path=usr/demo/dtrace/intr.d
file path=usr/demo/dtrace/iocpu.d
file path=usr/demo/dtrace/iosnoop.d
file path=usr/demo/dtrace/iothrough.d
file path=usr/demo/dtrace/iotime.d
file path=usr/demo/dtrace/ipio.d
file path=usr/demo/dtrace/ipproto.d
$(i386_ONLY)file path=usr/demo/dtrace/iprb.d
file path=usr/demo/dtrace/kstat.d
file path=usr/demo/dtrace/ksyms.d
file path=usr/demo/dtrace/libc.d
file path=usr/demo/dtrace/lquantize.d
file path=usr/demo/dtrace/lwptime.d
file path=usr/demo/dtrace/normalize.d
file path=usr/demo/dtrace/nscd.d
file path=usr/demo/dtrace/pri.d
file path=usr/demo/dtrace/printa.d
file path=usr/demo/dtrace/pritime.d
file path=usr/demo/dtrace/prof.d
file path=usr/demo/dtrace/profpri.d
file path=usr/demo/dtrace/progtime.d
file path=usr/demo/dtrace/putnext.d
file path=usr/demo/dtrace/qlen.d
file path=usr/demo/dtrace/qtime.d
file path=usr/demo/dtrace/renormalize.d
file path=usr/demo/dtrace/restest.d
file path=usr/demo/dtrace/ring.d
file path=usr/demo/dtrace/rtime.d
file path=usr/demo/dtrace/rwinfo.d
file path=usr/demo/dtrace/rwtime.d
file path=usr/demo/dtrace/sig.d
file path=usr/demo/dtrace/soffice.d
```

```
128 file path=usr/demo/dtrace/spec.d
129 file path=usr/demo/dtrace/specopen.d
130 file path=usr/demo/dtrace/ssd.d
131 file path=usr/demo/dtrace/syscall.d
132 file path=usr/demo/dtrace/tcp1stbyte.d
133 file path=usr/demo/dtrace/tcpbytes.d
134 file path=usr/demo/dtrace/tcpbytesstat.d
135 file path=usr/demo/dtrace/tcpconnlat.d
136 file path=usr/demo/dtrace/tcpio.d
137 file path=usr/demo/dtrace/tcpioflags.d
138 file path=usr/demo/dtrace/tcprst.d
139 file path=usr/demo/dtrace/tcpsnoop.d
140 file path=usr/demo/dtrace/tcpstate.d
141 file path=usr/demo/dtrace/tcptop.d
142 file path=usr/demo/dtrace/tick.d
143 file path=usr/demo/dtrace/ticktime.d
144 file path=usr/demo/dtrace/time.d
145 file path=usr/demo/dtrace/tracewrite.d
146 file path=usr/demo/dtrace/trunc.d
147 file path=usr/demo/dtrace/trussrw.d
148 file path=usr/demo/dtrace/udpbytes.d
149 file path=usr/demo/dtrace/udpbytesstat.d
150 file path=usr/demo/dtrace/udpio.d
151 file path=usr/demo/dtrace/udpsnoop.d
152 file path=usr/demo/dtrace/udptop.d
153 file path=usr/demo/dtrace/userfunc.d
154 file path=usr/demo/dtrace/whatfor.d
155 file path=usr/demo/dtrace/whatlock.d
156 file path=usr/demo/dtrace/where.d
157 file path=usr/demo/dtrace/whererun.d
158 file path=usr/demo/dtrace/whoexec.d
159 file path=usr/demo/dtrace/whofor.d
160 file path=usr/demo/dtrace/whoio.d
161 file path=usr/demo/dtrace/whopreempt.d
162 file path=usr/demo/dtrace/whoqueue.d
163 file path=usr/demo/dtrace/whosteal.d
164 file path=usr/demo/dtrace/whowrite.d
165 file path=usr/demo/dtrace/writes.d
166 file path=usr/demo/dtrace/writesbycmd.d
167 file path=usr/demo/dtrace/writesbycmdfd.d
168 file path=usr/demo/dtrace/writetime.d
169 file path=usr/demo/dtrace/writetimeq.d
170 file path=usr/demo/dtrace/xioctl.d
171 file path=usr/demo/dtrace/xterm.d
172 file path=usr/demo/dtrace/xwork.d
173 file path=usr/include/dtrace.h
174 file path=usr/include/sys/dtrace.h
175 file path=usr/include/sys/dtrace_impl.h
176 file path=usr/include/sys/fasttrap.h
177 file path=usr/include/sys/fasttrap_impl.h
178 file path=usr/include/sys/fasttrap_isa.h
179 file path=usr/include/sys/lockstat.h
180 file path=usr/include/sys/sdt.h
181 file path=usr/lib/$(ARCH64)/libdtrace.so.1
182 file path=usr/lib/$(ARCH64)/libdtrace_jni.so.1
183 file path=usr/lib/$(ARCH64)/llib-ldtrace.ln
184 file path=usr/lib/devfsadm/linkmod/SUNW_dtrace_link.so group=sys
185 file path=usr/lib/dtrace/64/drti.o mode=0444
186 file path=usr/lib/dtrace/64/libdtrace_forceload.so mode=0555
187 file path=usr/lib/dtrace/drti.o mode=0444
188 file path=usr/lib/dtrace/errno.d mode=0444
189 file path=usr/lib/dtrace/fc.d mode=0444
190 file path=usr/lib/dtrace/io.d mode=0444
191 file path=usr/lib/dtrace/ip.d mode=0444
192 file path=usr/lib/dtrace/iscsit.d mode=0444
193 file path=usr/lib/dtrace/libdtrace_forceload.so mode=0555
```



```
194 file path=usr/lib/dtrace/net.d mode=0444
195 file path=usr/lib/dtrace/nfs.d mode=0444
196 file path=usr/lib/dtrace/nfssrv.d mode=0444
197 file path=usr/lib/dtrace/procfs.d mode=0444
198 file path=usr/lib/dtrace/regs.d mode=0444
199 file path=usr/lib/dtrace/sched.d mode=0444
200 file path=usr/lib/dtrace/scsi.d mode=0444
201 file path=usr/lib/dtrace/sctp.d mode=0444
202 #endif /* ! codereview */
203 file path=usr/lib/dtrace/signal.d mode=0444
204 file path=usr/lib/dtrace/srp.d mode=0444
205 file path=usr/lib/dtrace/sysevent.d mode=0444
206 file path=usr/lib/dtrace/tcp.d mode=0444
207 file path=usr/lib/dtrace/udp.d mode=0444
208 file path=usr/lib/dtrace/unistd.d mode=0444
209 file path=usr/lib/libdtrace.so.1
210 file path=usr/lib/libdtrace_jni.so.1
211 file path=usr/lib/llib-ldtrace
212 file path=usr/lib/llib-ldtrace.ln
213 file path=usr/lib/mdb/kvm/$(ARCH64)/dtrace.so group=sys mode=0555
214 $(i386_ONLY)file path=usr/lib/mdb/kvm/dtrace.so group=sys mode=0555
215 file path=usr/lib/mdb/raw/$(ARCH64)/dof.so group=sys mode=0555
216 file path=usr/lib/mdb/raw/dof.so group=sys mode=0555
217 file path=usr/sbin/$(ARCH32)/dtrace mode=0555
218 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/intrstat mode=0555
219 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/lockstat mode=0555
220 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/plockstat mode=0555
221 file path=usr/sbin/$(ARCH64)/dtrace mode=0555
222 file path=usr/sbin/$(ARCH64)/intrstat mode=0555
223 file path=usr/sbin/$(ARCH64)/lockstat mode=0555
224 file path=usr/sbin/$(ARCH64)/plockstat mode=0555
225 file path=usr/share/lib/java/dtrace.jar group=sys
226 file path=usr/share/lib/java/javadoc/dtrace/api/allclasses-frame.html \
227     group=other
228 file path=usr/share/lib/java/javadoc/dtrace/api/allclasses-noframe.html \
229     group=other
230 file path=usr/share/lib/java/javadoc/dtrace/api/constant-values.html \
231     group=other
232 file path=usr/share/lib/java/javadoc/dtrace/api/deprecated-list.html \
233     group=other
234 file path=usr/share/lib/java/javadoc/dtrace/api/help-doc.html group=other
235 file path=usr/share/lib/java/javadoc/dtrace/api/index-all.html group=other
236 file path=usr/share/lib/java/javadoc/dtrace/api/index.html group=other
237 file \
238     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Aggrega
239     group=other
240 file \
241     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Aggrega
242     group=other
243 file \
244     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Aggrega
245     group=other
246 file \
247     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Aggrega
248     group=other
249 file \
250     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/AvgValu
251     group=other
252 file \
253     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Consume
254     group=other
255 file \
256     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Consume
257     group=other
258 file \
259     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Consume
```

```
260     group=other
261 file \
262     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Consume
263     group=other
264 file \
265     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Consume
266     group=other
267 file \
268     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Consume
269     group=other
270 file \
271     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/CountVa
272     group=other
273 file \
274     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/DTraceE
275     group=other
276 file \
277     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/DataEve
278     group=other
279 file \
280     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Distrib
281     group=other
282 file \
283     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Distrib
284     group=other
285 file \
286     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Drop.Ki
287     group=other
288 file \
289     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Drop.ht
290     group=other
291 file \
292     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/DropEve
293     group=other
294 file \
295     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Error.h
296     group=other
297 file \
298     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/ErrorEv
299     group=other
300 file \
301     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Excepti
302     group=other
303 file \
304     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/ExitRec
305     group=other
306 file \
307     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Flow.Ki
308     group=other
309 file \
310     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Flow.ht
311     group=other
312 file \
313     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Interfa
314     group=other
315 file \
316     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Interfa
317     group=other
318 file \
319     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Interfa
320     group=other
321 file \
322     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/KernelS
323     group=other
324 file \
325     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/KernelS
```



```
326     group=other
327 file \
328     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/LinearD
329     group=other
330 file \
331     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/LocalCo
332     group=other
333 file \
334     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/LogDist
335     group=other
336 file \
337     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/LogLine
338     group=other
339 file \
340     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/MaxValu
341     group=other
342 file \
343     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/MinValu
344     group=other
345 file \
346     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Option.
347     group=other
348 file \
349     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/PrintaR
350     group=other
351 file \
352     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/PrintfR
353     group=other
354 file \
355     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Probe.h
356     group=other
357 file \
358     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/ProbeDa
359     group=other
360 file \
361     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/ProbeDa
362     group=other
363 file \
364     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/ProbeDe
365     group=other
366 file \
367     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/ProbeDe
368     group=other
369 file \
370     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/ProbeIn
371     group=other
372 file \
373     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Process
374     group=other
375 file \
376     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Process
377     group=other
378 file \
379     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Process
380     group=other
381 file \
382     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Program
383     group=other
384 file \
385     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Program
386     group=other
387 file \
388     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Program
389     group=other
390 file \
391     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Record.
```

```
392     group=other
393 file \
394     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/ScalarR
395     group=other
396 file \
397     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/StackFr
398     group=other
399 file \
400     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/StackVa
401     group=other
402 file \
403     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/StddevV
404     group=other
405 file \
406     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/SumValu
407     group=other
408 file \
409     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/SymbolV
410     group=other
411 file \
412     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/Tuple.h
413     group=other
414 file \
415     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/UserSta
416     group=other
417 file \
418     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/UserSym
419     group=other
420 file \
421     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/UserSym
422     group=other
423 file \
424     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/ValueRe
425     group=other
426 file \
427     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
428     group=other
429 file \
430     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
431     group=other
432 file \
433     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
434     group=other
435 file \
436     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
437     group=other
438 file \
439     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
440     group=other
441 file \
442     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
443     group=other
444 file \
445     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
446     group=other
447 file \
448     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
449     group=other
450 file \
451     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
452     group=other
453 file \
454     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
455     group=other
456 file \
457     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
```



```
458     group=other
459 file \
460     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
461     group=other
462 file \
463     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
464     group=other
465 file \
466     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
467     group=other
468 file \
469     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
470     group=other
471 file \
472     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
473     group=other
474 file \
475     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
476     group=other
477 file \
478     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
479     group=other
480 file \
481     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
482     group=other
483 file \
484     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
485     group=other
486 file \
487     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
488     group=other
489 file \
490     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
491     group=other
492 file \
493     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
494     group=other
495 file \
496     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
497     group=other
498 file \
499     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
500     group=other
501 file \
502     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
503     group=other
504 file \
505     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
506     group=other
507 file \
508     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
509     group=other
510 file \
511     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
512     group=other
513 file \
514     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
515     group=other
516 file \
517     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
518     group=other
519 file \
520     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
521     group=other
522 file \
523     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
```

```
524     group=other
525 file \
526     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
527     group=other
528 file \
529     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
530     group=other
531 file \
532     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
533     group=other
534 file \
535     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
536     group=other
537 file \
538     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
539     group=other
540 file \
541     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
542     group=other
543 file \
544     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
545     group=other
546 file \
547     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
548     group=other
549 file \
550     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
551     group=other
552 file \
553     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
554     group=other
555 file \
556     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
557     group=other
558 file \
559     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
560     group=other
561 file \
562     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
563     group=other
564 file \
565     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
566     group=other
567 file \
568     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
569     group=other
570 file \
571     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
572     group=other
573 file \
574     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
575     group=other
576 file \
577     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
578     group=other
579 file \
580     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
581     group=other
582 file \
583     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
584     group=other
585 file \
586     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
587     group=other
588 file \
589     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
```



```
590     group=other
591 file \
592     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
593     group=other
594 file \
595     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
596     group=other
597 file \
598     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
599     group=other
600 file \
601     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
602     group=other
603 file \
604     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
605     group=other
606 file \
607     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
608     group=other
609 file \
610     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
611     group=other
612 file \
613     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/class-u
614     group=other
615 file \
616     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/package
617     group=other
618 file \
619     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/package
620     group=other
621 file \
622     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/package
623     group=other
624 file \
625     path=usr/share/lib/java/javadoc/dtrace/api/org/opensolaris/os/dtrace/package
626     group=other
627 file path=usr/share/lib/java/javadoc/dtrace/api/overview-tree.html group=other
628 file path=usr/share/lib/java/javadoc/dtrace/api/package-list group=other
629 file path=usr/share/lib/java/javadoc/dtrace/api/resources/inherit.gif \
630     group=other
631 file path=usr/share/lib/java/javadoc/dtrace/api/serialized-form.html \
632     group=other
633 file path=usr/share/lib/java/javadoc/dtrace/api/stylesheet.css group=other
634 file path=usr/share/lib/java/javadoc/dtrace/examples/TestAPI.java group=other
635 file path=usr/share/lib/java/javadoc/dtrace/examples/TestAPI2.java group=other
636 file path=usr/share/lib/java/javadoc/dtrace/examples/TestTarget.java \
637     group=other
638 file path=usr/share/lib/java/javadoc/dtrace/examples/hello.d group=other
639 file path=usr/share/lib/java/javadoc/dtrace/examples/intrstat.d group=other
640 file path=usr/share/lib/java/javadoc/dtrace/examples/syscall.d group=other
641 file path=usr/share/lib/java/javadoc/dtrace/examples/target.d group=other
642 file path=usr/share/lib/java/javadoc/dtrace/html/JavaDTraceAPI.html \
643     group=other
644 file path=usr/share/lib/java/javadoc/dtrace/html/fast.html group=other
645 file path=usr/share/lib/java/javadoc/dtrace/images/JavaDTraceAPI.gif \
646     group=other
647 file path=usr/share/man/man1m/dtrace.1m
648 file path=usr/share/man/man1m/intrstat.1m
649 file path=usr/share/man/man1m/lockstat.1m
650 file path=usr/share/man/man1m/plockstat.1m
651 file path=usr/share/man/man3lib/libdtrace.3lib
652 hardlink path=usr/sbin/dtrace target=../../usr/lib/isaexec
653 hardlink path=usr/sbin/intrstat target=../../usr/lib/isaexec
654 hardlink path=usr/sbin/lockstat target=../../usr/lib/isaexec
655 hardlink path=usr/sbin/plockstat target=../../usr/lib/isaexec
```

```
656 legacy pkg=SUNWdtrc desc="Dynamic Tracing (DTrace) Clients" \
657     name="DTrace Clients"
658 license cr_Sun license=cr_Sun
659 license lic_CDDL license=lic_CDDL
660 link path=usr/lib/$(ARCH64)/libdtrace.so target=libdtrace.so.1
661 link path=usr/lib/$(ARCH64)/libdtrace_jni.so target=libdtrace_jni.so.1
662 link path=usr/lib/libdtrace.so target=libdtrace.so.1
663 link path=usr/lib/libdtrace_jni.so target=libdtrace_jni.so.1
```

```
**********************************************************
   55165 Sat Apr 12 11:18:55 2014
new/usr/src/uts/common/dtrace/sdt_subr.c
3903 DTrace SCTP Provider
**********************************************************
______unchanged_portion_omitted_

  99 sdt_provider_t sdt_providers[] = {
 100         { "vtrace", "__vtrace_", &vtrace_attr, 0 },
 101         { "sysinfo", "__cpu_sysinfo_", &info_attr, 0 },
 102         { "vminfo", "__cpu_vminfo_", &info_attr, 0 },
 103         { "fpuinfo", "__fpuinfo_", &fpu_attr, 0 },
 104         { "sched", "__sched_", &stab_attr, 0 },
 105         { "proc", "__proc_", &stab_attr, 0 },
 106         { "io", "__io_", &stab_attr, 0 },
 107         { "ip", "__ip_", &stab_attr, 0 },
 108         { "tcp", "__tcp_", &stab_attr, 0 },
 109         { "udp", "__udp_", &stab_attr, 0 },
 110         { "sctp", "__sctp_", &stab_attr, 0 },
 111 #endif /* ! codereview */
 112         { "mib", "__mib_", &stab_attr, 0 },
 113         { "fsinfo", "__fsinfo_", &fsinfo_attr, 0 },
 114         { "iscsi", "__iscsi_", &iscsi_attr, 0 },
 115         { "nfsv3", "__nfsv3_", &stab_attr, 0 },
 116         { "nfsv4", "__nfsv4_", &stab_attr, 0 },
 117         { "xpv", "__xpv_", &xpv_attr, 0 },
 118         { "fc", "__fc_", &fc_attr, 0 },
 119         { "srp", "__srp_", &fc_attr, 0 },
 120         { "sysevent", "__sysevent_", &stab_attr, 0 },
 121         { "sdt", NULL, &sdt_attr, 0 },
 122         { NULL }
 123 };

 125 sdt_argdesc_t sdt_args[] = {
 126         { "sched", "wakeup", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 127         { "sched", "wakeup", 1, 0, "kthread_t *", "psinfo_t *" },
 128         { "sched", "dequeue", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 129         { "sched", "dequeue", 1, 0, "kthread_t *", "psinfo_t *" },
 130         { "sched", "dequeue", 2, 1, "disp_t *", "cpuinfo_t *" },
 131         { "sched", "enqueue", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 132         { "sched", "enqueue", 1, 0, "kthread_t *", "psinfo_t *" },
 133         { "sched", "enqueue", 2, 1, "disp_t *", "cpuinfo_t *" },
 134         { "sched", "enqueue", 3, 2, "int" },
 135         { "sched", "off-cpu", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 136         { "sched", "off-cpu", 1, 0, "kthread_t *", "psinfo_t *" },
 137         { "sched", "tick", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 138         { "sched", "tick", 1, 0, "kthread_t *", "psinfo_t *" },
 139         { "sched", "change-pri", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 140         { "sched", "change-pri", 1, 0, "kthread_t *", "psinfo_t *" },
 141         { "sched", "change-pri", 2, 1, "pri_t" },
 142         { "sched", "schedctl-nopreempt", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 143         { "sched", "schedctl-nopreempt", 1, 0, "kthread_t *", "psinfo_t *" },
 144         { "sched", "schedctl-nopreempt", 2, 1, "int" },
 145         { "sched", "schedctl-preempt", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 146         { "sched", "schedctl-preempt", 1, 0, "kthread_t *", "psinfo_t *" },
 147         { "sched", "schedctl-yield", 0, 0, "int" },
 148         { "sched", "surrender", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 149         { "sched", "surrender", 1, 0, "kthread_t *", "psinfo_t *" },
 150         { "sched", "cpucaps-sleep", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 151         { "sched", "cpucaps-sleep", 1, 0, "kthread_t *", "psinfo_t *" },
 152         { "sched", "cpucaps-wakeup", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 153         { "sched", "cpucaps-wakeup", 1, 0, "kthread_t *", "psinfo_t *" },

 155         { "proc", "create", 0, 0, "proc_t *", "psinfo_t *" },
 156         { "proc", "exec", 0, 0, "string" },
 157         { "proc", "exec-failure", 0, 0, "int" },
```



```
 158         { "proc", "exit", 0, 0, "int" },
 159         { "proc", "fault", 0, 0, "int" },
 160         { "proc", "fault", 1, 1, "siginfo_t *" },
 161         { "proc", "lwp-create", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 162         { "proc", "lwp-create", 1, 0, "kthread_t *", "psinfo_t *" },
 163         { "proc", "signal-clear", 0, 0, "int" },
 164         { "proc", "signal-clear", 1, 1, "siginfo_t *" },
 165         { "proc", "signal-discard", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 166         { "proc", "signal-discard", 1, 1, "proc_t *", "psinfo_t *" },
 167         { "proc", "signal-discard", 2, 2, "int" },
 168         { "proc", "signal-handle", 0, 0, "int" },
 169         { "proc", "signal-handle", 1, 1, "siginfo_t *" },
 170         { "proc", "signal-handle", 2, 2, "void (*)(void)" },
 171         { "proc", "signal-send", 0, 0, "kthread_t *", "lwpsinfo_t *" },
 172         { "proc", "signal-send", 1, 0, "kthread_t *", "psinfo_t *" },
 173         { "proc", "signal-send", 2, 1, "int" },

 175         { "io", "start", 0, 0, "buf_t *", "bufinfo_t *" },
 176         { "io", "start", 1, 0, "buf_t *", "devinfo_t *" },
 177         { "io", "start", 2, 0, "buf_t *", "fileinfo_t *" },
 178         { "io", "done", 0, 0, "buf_t *", "bufinfo_t *" },
 179         { "io", "done", 1, 0, "buf_t *", "devinfo_t *" },
 180         { "io", "done", 2, 0, "buf_t *", "fileinfo_t *" },
 181         { "io", "wait-start", 0, 0, "buf_t *", "bufinfo_t *" },
 182         { "io", "wait-start", 1, 0, "buf_t *", "devinfo_t *" },
 183         { "io", "wait-start", 2, 0, "buf_t *", "fileinfo_t *" },
 184         { "io", "wait-done", 0, 0, "buf_t *", "bufinfo_t *" },
 185         { "io", "wait-done", 1, 0, "buf_t *", "devinfo_t *" },
 186         { "io", "wait-done", 2, 0, "buf_t *", "fileinfo_t *" },

 188         { "mib", NULL, 0, 0, "int" },

 190         { "fsinfo", NULL, 0, 0, "vnode_t *", "fileinfo_t *" },
 191         { "fsinfo", NULL, 1, 1, "int", "int" },

 193         { "iscsi", "async-send", 0, 0, "idm_conn_t *", "conninfo_t *" },
 194         { "iscsi", "async-send", 1, 1, "iscsi_async_evt_hdr_t *",
               "iscsiinfo_t *" },
 196         { "iscsi", "login-command", 0, 0, "idm_conn_t *", "conninfo_t *" },
 197         { "iscsi", "login-command", 1, 1, "iscsi_login_hdr_t *",
               "iscsiinfo_t *" },
 199         { "iscsi", "login-response", 0, 0, "idm_conn_t *", "conninfo_t *" },
 200         { "iscsi", "login-response", 1, 1, "iscsi_login_rsp_hdr_t *",
               "iscsiinfo_t *" },
 202         { "iscsi", "logout-command", 0, 0, "idm_conn_t *", "conninfo_t *" },
 203         { "iscsi", "logout-command", 1, 1, "iscsi_logout_hdr_t *",
               "iscsiinfo_t *" },
 205         { "iscsi", "logout-response", 0, 0, "idm_conn_t *", "conninfo_t *" },
 206         { "iscsi", "logout-response", 1, 1, "iscsi_logout_rsp_hdr_t *",
               "iscsiinfo_t *" },
 208         { "iscsi", "data-request", 0, 0, "idm_conn_t *", "conninfo_t *" },
 209         { "iscsi", "data-request", 1, 1, "iscsi_rtt_hdr_t *",
               "iscsiinfo_t *" },
 211         { "iscsi", "data-send", 0, 0, "idm_conn_t *", "conninfo_t *" },
 212         { "iscsi", "data-send", 1, 1, "iscsi_data_rsp_hdr_t *",
               "iscsiinfo_t *" },
 214         { "iscsi", "data-receive", 0, 0, "idm_conn_t *", "conninfo_t *" },
 215         { "iscsi", "data-receive", 1, 1, "iscsi_data_hdr_t *",
               "iscsiinfo_t *" },
 217         { "iscsi", "nop-send", 0, 0, "idm_conn_t *", "conninfo_t *" },
 218         { "iscsi", "nop-send", 1, 1, "iscsi_nop_in_hdr_t *", "iscsiinfo_t *" },
 219         { "iscsi", "nop-receive", 0, 0, "idm_conn_t *", "conninfo_t *" },
 220         { "iscsi", "nop-receive", 1, 1, "iscsi_nop_out_hdr_t *",
               "iscsiinfo_t *" },
 222         { "iscsi", "scsi-command", 0, 0, "idm_conn_t *", "conninfo_t *" },
 223         { "iscsi", "scsi-command", 1, 1, "iscsi_scsi_cmd_hdr_t *",
```

```
224             "iscsiinfo_t *" },
225         { "iscsi", "scsi-command", 2, 2, "scsi_task_t *", "scsicmd_t *" },
226         { "iscsi", "scsi-response", 0, 0, "idm_conn_t *", "conninfo_t *" },
227         { "iscsi", "scsi-response", 1, 1, "iscsi_scsi_rsp_hdr_t *",
228             "iscsiinfo_t *" },
229         { "iscsi", "task-command", 0, 0, "idm_conn_t *", "conninfo_t *" },
230         { "iscsi", "task-command", 1, 1, "iscsi_scsi_task_mgt_hdr_t *",
231             "iscsiinfo_t *" },
232         { "iscsi", "task-response", 0, 0, "idm_conn_t *", "conninfo_t *" },
233         { "iscsi", "task-response", 1, 1, "iscsi_scsi_task_mgt_rsp_hdr_t *",
234             "iscsiinfo_t *" },
235         { "iscsi", "text-command", 0, 0, "idm_conn_t *", "conninfo_t *" },
236         { "iscsi", "text-command", 1, 1, "iscsi_text_hdr_t *",
237             "iscsiinfo_t *" },
238         { "iscsi", "text-response", 0, 0, "idm_conn_t *", "conninfo_t *" },
239         { "iscsi", "text-response", 1, 1, "iscsi_text_rsp_hdr_t *",
240             "iscsiinfo_t *" },
241         { "iscsi", "xfer-start", 0, 0, "idm_conn_t *", "conninfo_t *" },
242         { "iscsi", "xfer-start", 1, 0, "idm_conn_t *", "iscsiinfo_t *" },
243         { "iscsi", "xfer-start", 2, 1, "uintptr_t", "xferinfo_t *" },
244         { "iscsi", "xfer-start", 3, 2, "uint32_t"},
245         { "iscsi", "xfer-start", 4, 3, "uintptr_t"},
246         { "iscsi", "xfer-start", 5, 4, "uint32_t"},
247         { "iscsi", "xfer-start", 6, 5, "uint32_t"},
248         { "iscsi", "xfer-start", 7, 6, "uint32_t"},
249         { "iscsi", "xfer-start", 8, 7, "int"},
250         { "iscsi", "xfer-done", 0, 0, "idm_conn_t *", "conninfo_t *" },
251         { "iscsi", "xfer-done", 1, 0, "idm_conn_t *", "iscsiinfo_t *" },
252         { "iscsi", "xfer-done", 2, 1, "uintptr_t", "xferinfo_t *" },
253         { "iscsi", "xfer-done", 3, 2, "uint32_t"},
254         { "iscsi", "xfer-done", 4, 3, "uintptr_t"},
255         { "iscsi", "xfer-done", 5, 4, "uint32_t"},
256         { "iscsi", "xfer-done", 6, 5, "uint32_t"},
257         { "iscsi", "xfer-done", 7, 6, "uint32_t"},
258         { "iscsi", "xfer-done", 8, 7, "int"},

260         { "nfsv3", "op-getattr-start", 0, 0, "struct svc_req *",
261             "conninfo_t *" },
262         { "nfsv3", "op-getattr-start", 1, 1, "nfsv3oparg_t *",
263             "nfsv3opinfo_t *" },
264         { "nfsv3", "op-getattr-start", 2, 3, "GETATTR3args *" },
265         { "nfsv3", "op-getattr-done", 0, 0, "struct svc_req *",
266             "conninfo_t *" },
267         { "nfsv3", "op-getattr-done", 1, 1, "nfsv3oparg_t *",
268             "nfsv3opinfo_t *" },
269         { "nfsv3", "op-getattr-done", 2, 3, "GETATTR3res *" },
270         { "nfsv3", "op-setattr-start", 0, 0, "struct svc_req *",
271             "conninfo_t *" },
272         { "nfsv3", "op-setattr-start", 1, 1, "nfsv3oparg_t *",
273             "nfsv3opinfo_t *" },
274         { "nfsv3", "op-setattr-start", 2, 3, "SETATTR3args *" },
275         { "nfsv3", "op-setattr-done", 0, 0, "struct svc_req *",
276             "conninfo_t *" },
277         { "nfsv3", "op-setattr-done", 1, 1, "nfsv3oparg_t *",
278             "nfsv3opinfo_t *" },
279         { "nfsv3", "op-setattr-done", 2, 3, "SETATTR3res *" },
280         { "nfsv3", "op-lookup-start", 0, 0, "struct svc_req *",
281             "conninfo_t *" },
282         { "nfsv3", "op-lookup-start", 1, 1, "nfsv3oparg_t *",
283             "nfsv3opinfo_t *" },
284         { "nfsv3", "op-lookup-start", 2, 3, "LOOKUP3args *" },
285         { "nfsv3", "op-lookup-done", 0, 0, "struct svc_req *",
286             "conninfo_t *" },
287         { "nfsv3", "op-lookup-done", 1, 1, "nfsv3oparg_t *",
288             "nfsv3opinfo_t *" },
289         { "nfsv3", "op-lookup-done", 2, 3, "LOOKUP3res *" },
```



```
290         { "nfsv3", "op-access-start", 0, 0, "struct svc_req *",
291             "conninfo_t *" },
292         { "nfsv3", "op-access-start", 1, 1, "nfsv3oparg_t *",
293             "nfsv3opinfo_t *" },
294         { "nfsv3", "op-access-start", 2, 3, "ACCESS3args *" },
295         { "nfsv3", "op-access-done", 0, 0, "struct svc_req *",
296             "conninfo_t *" },
297         { "nfsv3", "op-access-done", 1, 1, "nfsv3oparg_t *",
298             "nfsv3opinfo_t *" },
299         { "nfsv3", "op-access-done", 2, 3, "ACCESS3res *" },
300         { "nfsv3", "op-commit-start", 0, 0, "struct svc_req *",
301             "conninfo_t *" },
302         { "nfsv3", "op-commit-start", 1, 1, "nfsv3oparg_t *",
303             "nfsv3opinfo_t *" },
304         { "nfsv3", "op-commit-start", 2, 3, "COMMIT3args *" },
305         { "nfsv3", "op-commit-done", 0, 0, "struct svc_req *",
306             "conninfo_t *" },
307         { "nfsv3", "op-commit-done", 1, 1, "nfsv3oparg_t *",
308             "nfsv3opinfo_t *" },
309         { "nfsv3", "op-commit-done", 2, 3, "COMMIT3res *" },
310         { "nfsv3", "op-create-start", 0, 0, "struct svc_req *",
311             "conninfo_t *" },
312         { "nfsv3", "op-create-start", 1, 1, "nfsv3oparg_t *",
313             "nfsv3opinfo_t *" },
314         { "nfsv3", "op-create-start", 2, 3, "CREATE3args *" },
315         { "nfsv3", "op-create-done", 0, 0, "struct svc_req *",
316             "conninfo_t *" },
317         { "nfsv3", "op-create-done", 1, 1, "nfsv3oparg_t *",
318             "nfsv3opinfo_t *" },
319         { "nfsv3", "op-create-done", 2, 3, "CREATE3res *" },
320         { "nfsv3", "op-fsinfo-start", 0, 0, "struct svc_req *",
321             "conninfo_t *" },
322         { "nfsv3", "op-fsinfo-start", 1, 1, "nfsv3oparg_t *",
323             "nfsv3opinfo_t *" },
324         { "nfsv3", "op-fsinfo-start", 2, 3, "FSINFO3args *" },
325         { "nfsv3", "op-fsinfo-done", 0, 0, "struct svc_req *",
326             "conninfo_t *" },
327         { "nfsv3", "op-fsinfo-done", 1, 1, "nfsv3oparg_t *",
328             "nfsv3opinfo_t *" },
329         { "nfsv3", "op-fsinfo-done", 2, 3, "FSINFO3res *" },
330         { "nfsv3", "op-fsstat-start", 0, 0, "struct svc_req *",
331             "conninfo_t *" },
332         { "nfsv3", "op-fsstat-start", 1, 1, "nfsv3oparg_t *",
333             "nfsv3opinfo_t *" },
334         { "nfsv3", "op-fsstat-start", 2, 3, "FSSTAT3args *" },
335         { "nfsv3", "op-fsstat-done", 0, 0, "struct svc_req *",
336             "conninfo_t *" },
337         { "nfsv3", "op-fsstat-done", 1, 1, "nfsv3oparg_t *",
338             "nfsv3opinfo_t *" },
339         { "nfsv3", "op-fsstat-done", 2, 3, "FSSTAT3res *" },
340         { "nfsv3", "op-link-start", 0, 0, "struct svc_req *",
341             "conninfo_t *" },
342         { "nfsv3", "op-link-start", 1, 1, "nfsv3oparg_t *",
343             "nfsv3opinfo_t *" },
344         { "nfsv3", "op-link-start", 2, 3, "LINK3args *" },
345         { "nfsv3", "op-link-done", 0, 0, "struct svc_req *",
346             "conninfo_t *" },
347         { "nfsv3", "op-link-done", 1, 1, "nfsv3oparg_t *",
348             "nfsv3opinfo_t *" },
349         { "nfsv3", "op-link-done", 2, 3, "LINK3res *" },
350         { "nfsv3", "op-mkdir-start", 0, 0, "struct svc_req *",
351             "conninfo_t *" },
352         { "nfsv3", "op-mkdir-start", 1, 1, "nfsv3oparg_t *",
353             "nfsv3opinfo_t *" },
354         { "nfsv3", "op-mkdir-start", 2, 3, "MKDIR3args *" },
355         { "nfsv3", "op-mkdir-done", 0, 0, "struct svc_req *",
```

```
356             "conninfo_t *" },
357         { "nfsv3", "op-mkdir-done", 1, 1, "nfsv3oparg_t *",
358             "nfsv3opinfo_t *" },
359         { "nfsv3", "op-mkdir-done", 2, 3, "MKDIR3res *" },
360         { "nfsv3", "op-mknod-start", 0, 0, "struct svc_req *",
361             "conninfo_t *" },
362         { "nfsv3", "op-mknod-start", 1, 1, "nfsv3oparg_t *",
363             "nfsv3opinfo_t *" },
364         { "nfsv3", "op-mknod-start", 2, 3, "MKNOD3args *" },
365         { "nfsv3", "op-mknod-done", 0, 0, "struct svc_req *",
366             "conninfo_t *" },
367         { "nfsv3", "op-mknod-done", 1, 1, "nfsv3oparg_t *",
368             "nfsv3opinfo_t *" },
369         { "nfsv3", "op-mknod-done", 2, 3, "MKNOD3res *" },
370         { "nfsv3", "op-null-start", 0, 0, "struct svc_req *",
371             "conninfo_t *" },
372         { "nfsv3", "op-null-start", 1, 1, "nfsv3oparg_t *",
373             "nfsv3opinfo_t *" },
374         { "nfsv3", "op-null-done", 0, 0, "struct svc_req *",
375             "conninfo_t *" },
376         { "nfsv3", "op-null-done", 1, 1, "nfsv3oparg_t *",
377             "nfsv3opinfo_t *" },
378         { "nfsv3", "op-pathconf-start", 0, 0, "struct svc_req *",
379             "conninfo_t *" },
380         { "nfsv3", "op-pathconf-start", 1, 1, "nfsv3oparg_t *",
381             "nfsv3opinfo_t *" },
382         { "nfsv3", "op-pathconf-start", 2, 3, "PATHCONF3args *" },
383         { "nfsv3", "op-pathconf-done", 0, 0, "struct svc_req *",
384             "conninfo_t *" },
385         { "nfsv3", "op-pathconf-done", 1, 1, "nfsv3oparg_t *",
386             "nfsv3opinfo_t *" },
387         { "nfsv3", "op-pathconf-done", 2, 3, "PATHCONF3res *" },
388         { "nfsv3", "op-read-start", 0, 0, "struct svc_req *",
389             "conninfo_t *" },
390         { "nfsv3", "op-read-start", 1, 1, "nfsv3oparg_t *",
391             "nfsv3opinfo_t *" },
392         { "nfsv3", "op-read-start", 2, 3, "READ3args *" },
393         { "nfsv3", "op-read-done", 0, 0, "struct svc_req *",
394             "conninfo_t *" },
395         { "nfsv3", "op-read-done", 1, 1, "nfsv3oparg_t *",
396             "nfsv3opinfo_t *" },
397         { "nfsv3", "op-read-done", 2, 3, "READ3res *" },
398         { "nfsv3", "op-readdir-start", 0, 0, "struct svc_req *",
399             "conninfo_t *" },
400         { "nfsv3", "op-readdir-start", 1, 1, "nfsv3oparg_t *",
401             "nfsv3opinfo_t *" },
402         { "nfsv3", "op-readdir-start", 2, 3, "READDIR3args *" },
403         { "nfsv3", "op-readdir-done", 0, 0, "struct svc_req *",
404             "conninfo_t *" },
405         { "nfsv3", "op-readdir-done", 1, 1, "nfsv3oparg_t *",
406             "nfsv3opinfo_t *" },
407         { "nfsv3", "op-readdir-done", 2, 3, "READDIR3res *" },
408         { "nfsv3", "op-readdirplus-start", 0, 0, "struct svc_req *",
409             "conninfo_t *" },
410         { "nfsv3", "op-readdirplus-start", 1, 1, "nfsv3oparg_t *",
411             "nfsv3opinfo_t *" },
412         { "nfsv3", "op-readdirplus-start", 2, 3, "READDIRPLUS3args *" },
413         { "nfsv3", "op-readdirplus-done", 0, 0, "struct svc_req *",
414             "conninfo_t *" },
415         { "nfsv3", "op-readdirplus-done", 1, 1, "nfsv3oparg_t *",
416             "nfsv3opinfo_t *" },
417         { "nfsv3", "op-readdirplus-done", 2, 3, "READDIRPLUS3res *" },
418         { "nfsv3", "op-readlink-start", 0, 0, "struct svc_req *",
419             "conninfo_t *" },
420         { "nfsv3", "op-readlink-start", 1, 1, "nfsv3oparg_t *",
421             "nfsv3opinfo_t *" },
```



```
422         { "nfsv3", "op-readlink-start", 2, 3, "READLINK3args *" },
423         { "nfsv3", "op-readlink-done", 0, 0, "struct svc_req *",
424             "conninfo_t *" },
425         { "nfsv3", "op-readlink-done", 1, 1, "nfsv3oparg_t *",
426             "nfsv3opinfo_t *" },
427         { "nfsv3", "op-readlink-done", 2, 3, "READLINK3res *" },
428         { "nfsv3", "op-remove-start", 0, 0, "struct svc_req *",
429             "conninfo_t *" },
430         { "nfsv3", "op-remove-start", 1, 1, "nfsv3oparg_t *",
431             "nfsv3opinfo_t *" },
432         { "nfsv3", "op-remove-start", 2, 3, "REMOVE3args *" },
433         { "nfsv3", "op-remove-done", 0, 0, "struct svc_req *",
434             "conninfo_t *" },
435         { "nfsv3", "op-remove-done", 1, 1, "nfsv3oparg_t *",
436             "nfsv3opinfo_t *" },
437         { "nfsv3", "op-remove-done", 2, 3, "REMOVE3res *" },
438         { "nfsv3", "op-rename-start", 0, 0, "struct svc_req *",
439             "conninfo_t *" },
440         { "nfsv3", "op-rename-start", 1, 1, "nfsv3oparg_t *",
441             "nfsv3opinfo_t *" },
442         { "nfsv3", "op-rename-start", 2, 3, "RENAME3args *" },
443         { "nfsv3", "op-rename-done", 0, 0, "struct svc_req *",
444             "conninfo_t *" },
445         { "nfsv3", "op-rename-done", 1, 1, "nfsv3oparg_t *",
446             "nfsv3opinfo_t *" },
447         { "nfsv3", "op-rename-done", 2, 3, "RENAME3res *" },
448         { "nfsv3", "op-rmdir-start", 0, 0, "struct svc_req *",
449             "conninfo_t *" },
450         { "nfsv3", "op-rmdir-start", 1, 1, "nfsv3oparg_t *",
451             "nfsv3opinfo_t *" },
452         { "nfsv3", "op-rmdir-start", 2, 3, "RMDIR3args *" },
453         { "nfsv3", "op-rmdir-done", 0, 0, "struct svc_req *",
454             "conninfo_t *" },
455         { "nfsv3", "op-rmdir-done", 1, 1, "nfsv3oparg_t *",
456             "nfsv3opinfo_t *" },
457         { "nfsv3", "op-rmdir-done", 2, 3, "RMDIR3res *" },
458         { "nfsv3", "op-setattr-start", 0, 0, "struct svc_req *",
459             "conninfo_t *" },
460         { "nfsv3", "op-setattr-start", 1, 1, "nfsv3oparg_t *",
461             "nfsv3opinfo_t *" },
462         { "nfsv3", "op-setattr-start", 2, 3, "SETATTR3args *" },
463         { "nfsv3", "op-setattr-done", 0, 0, "struct svc_req *",
464             "conninfo_t *" },
465         { "nfsv3", "op-setattr-done", 1, 1, "nfsv3oparg_t *",
466             "nfsv3opinfo_t *" },
467         { "nfsv3", "op-setattr-done", 2, 3, "SETATTR3res *" },
468         { "nfsv3", "op-symlink-start", 0, 0, "struct svc_req *",
469             "conninfo_t *" },
470         { "nfsv3", "op-symlink-start", 1, 1, "nfsv3oparg_t *",
471             "nfsv3opinfo_t *" },
472         { "nfsv3", "op-symlink-start", 2, 3, "SYMLINK3args *" },
473         { "nfsv3", "op-symlink-done", 0, 0, "struct svc_req *",
474             "conninfo_t *" },
475         { "nfsv3", "op-symlink-done", 1, 1, "nfsv3oparg_t *",
476             "nfsv3opinfo_t *" },
477         { "nfsv3", "op-symlink-done", 2, 3, "SYMLINK3res *" },
478         { "nfsv3", "op-write-start", 0, 0, "struct svc_req *",
479             "conninfo_t *" },
480         { "nfsv3", "op-write-start", 1, 1, "nfsv3oparg_t *",
481             "nfsv3opinfo_t *" },
482         { "nfsv3", "op-write-start", 2, 3, "WRITE3args *" },
483         { "nfsv3", "op-write-done", 0, 0, "struct svc_req *",
484             "conninfo_t *" },
485         { "nfsv3", "op-write-done", 1, 1, "nfsv3oparg_t *",
486             "nfsv3opinfo_t *" },
487         { "nfsv3", "op-write-done", 2, 3, "WRITE3res *" },
```

```
489         { "nfsv4", "null-start", 0, 0, "struct svc_req *", "conninfo_t *" },
490         { "nfsv4", "null-done", 0, 0, "struct svc_req *", "conninfo_t *" },
491         { "nfsv4", "compound-start", 0, 0, "struct compound_state *",
492           "conninfo_t *" },
493         { "nfsv4", "compound-start", 1, 0, "struct compound_state *",
494           "nfsv4opinfo_t *" },
495         { "nfsv4", "compound-start", 2, 1, "COMPOUND4args *" },
496         { "nfsv4", "compound-done", 0, 0, "struct compound_state *",
497           "conninfo_t *" },
498         { "nfsv4", "compound-done", 1, 0, "struct compound_state *",
499           "nfsv4opinfo_t *" },
500         { "nfsv4", "compound-done", 2, 1, "COMPOUND4res *" },
501         { "nfsv4", "op-access-start", 0, 0, "struct compound_state *",
502           "conninfo_t *"},
503         { "nfsv4", "op-access-start", 1, 0, "struct compound_state *",
504           "nfsv4opinfo_t *" },
505         { "nfsv4", "op-access-start", 2, 1, "ACCESS4args *" },
506         { "nfsv4", "op-access-done", 0, 0, "struct compound_state *",
507           "conninfo_t *" },
508         { "nfsv4", "op-access-done", 1, 0, "struct compound_state *",
509           "nfsv4opinfo_t *" },
510         { "nfsv4", "op-access-done", 2, 1, "ACCESS4res *" },
511         { "nfsv4", "op-close-start", 0, 0, "struct compound_state *",
512           "conninfo_t *" },
513         { "nfsv4", "op-close-start", 1, 0, "struct compound_state *",
514           "nfsv4opinfo_t *" },
515         { "nfsv4", "op-close-start", 2, 1, "CLOSE4args *" },
516         { "nfsv4", "op-close-done", 0, 0, "struct compound_state *",
517           "conninfo_t *" },
518         { "nfsv4", "op-close-done", 1, 0, "struct compound_state *",
519           "nfsv4opinfo_t *" },
520         { "nfsv4", "op-close-done", 2, 1, "CLOSE4res *" },
521         { "nfsv4", "op-commit-start", 0, 0, "struct compound_state *",
522           "conninfo_t *" },
523         { "nfsv4", "op-commit-start", 1, 0, "struct compound_state *",
524           "nfsv4opinfo_t *" },
525         { "nfsv4", "op-commit-start", 2, 1, "COMMIT4args *" },
526         { "nfsv4", "op-commit-done", 0, 0, "struct compound_state *",
527           "conninfo_t *" },
528         { "nfsv4", "op-commit-done", 1, 0, "struct compound_state *",
529           "nfsv4opinfo_t *" },
530         { "nfsv4", "op-commit-done", 2, 1, "COMMIT4res *" },
531         { "nfsv4", "op-create-start", 0, 0, "struct compound_state *",
532           "conninfo_t *" },
533         { "nfsv4", "op-create-start", 1, 0, "struct compound_state *",
534           "nfsv4opinfo_t *" },
535         { "nfsv4", "op-create-start", 2, 1, "CREATE4args *" },
536         { "nfsv4", "op-create-done", 0, 0, "struct compound_state *",
537           "conninfo_t *" },
538         { "nfsv4", "op-create-done", 1, 0, "struct compound_state *",
539           "nfsv4opinfo_t *" },
540         { "nfsv4", "op-create-done", 2, 1, "CREATE4res *" },
541         { "nfsv4", "op-delegpurge-start", 0, 0, "struct compound_state *",
542           "conninfo_t *" },
543         { "nfsv4", "op-delegpurge-start", 1, 0, "struct compound_state *",
544           "nfsv4opinfo_t *" },
545         { "nfsv4", "op-delegpurge-start", 2, 1, "DELEGPURGE4args *" },
546         { "nfsv4", "op-delegpurge-done", 0, 0, "struct compound_state *",
547           "conninfo_t *" },
548         { "nfsv4", "op-delegpurge-done", 1, 0, "struct compound_state *",
549           "nfsv4opinfo_t *" },
550         { "nfsv4", "op-delegpurge-done", 2, 1, "DELEGPURGE4res *" },
551         { "nfsv4", "op-delegreturn-start", 0, 0, "struct compound_state *",
552           "conninfo_t *" },
553         { "nfsv4", "op-delegreturn-start", 1, 0, "struct compound_state *",
```



```
554           "nfsv4opinfo_t *" },
555         { "nfsv4", "op-delegreturn-start", 2, 1, "DELEGRETURN4args *" },
556         { "nfsv4", "op-delegreturn-done", 0, 0, "struct compound_state *",
557           "conninfo_t *" },
558         { "nfsv4", "op-delegreturn-done", 1, 0, "struct compound_state *",
559           "nfsv4opinfo_t *" },
560         { "nfsv4", "op-delegreturn-done", 2, 1, "DELEGRETURN4res *" },
561         { "nfsv4", "op-getattr-start", 0, 0, "struct compound_state *",
562           "conninfo_t *" },
563         { "nfsv4", "op-getattr-start", 1, 0, "struct compound_state *",
564           "nfsv4opinfo_t *" },
565         { "nfsv4", "op-getattr-start", 2, 1, "GETATTR4args *" },
566         { "nfsv4", "op-getattr-done", 0, 0, "struct compound_state *",
567           "conninfo_t *" },
568         { "nfsv4", "op-getattr-done", 1, 0, "struct compound_state *",
569           "nfsv4opinfo_t *" },
570         { "nfsv4", "op-getattr-done", 2, 1, "GETATTR4res *" },
571         { "nfsv4", "op-getfh-start", 0, 0, "struct compound_state *",
572           "conninfo_t *" },
573         { "nfsv4", "op-getfh-start", 1, 0, "struct compound_state *",
574           "nfsv4opinfo_t *" },
575         { "nfsv4", "op-getfh-done", 0, 0, "struct compound_state *",
576           "conninfo_t *" },
577         { "nfsv4", "op-getfh-done", 1, 0, "struct compound_state *",
578           "nfsv4opinfo_t *" },
579         { "nfsv4", "op-getfh-done", 2, 1, "GETFH4res *" },
580         { "nfsv4", "op-link-start", 0, 0, "struct compound_state *",
581           "conninfo_t *" },
582         { "nfsv4", "op-link-start", 1, 0, "struct compound_state *",
583           "nfsv4opinfo_t *" },
584         { "nfsv4", "op-link-start", 2, 1, "LINK4args *" },
585         { "nfsv4", "op-link-done", 0, 0, "struct compound_state *",
586           "conninfo_t *" },
587         { "nfsv4", "op-link-done", 1, 0, "struct compound_state *",
588           "nfsv4opinfo_t *" },
589         { "nfsv4", "op-link-done", 2, 1, "LINK4res *" },
590         { "nfsv4", "op-lock-start", 0, 0, "struct compound_state *",
591           "conninfo_t *" },
592         { "nfsv4", "op-lock-start", 1, 0, "struct compound_state *",
593           "nfsv4opinfo_t *" },
594         { "nfsv4", "op-lock-start", 2, 1, "LOCK4args *" },
595         { "nfsv4", "op-lock-done", 0, 0, "struct compound_state *",
596           "conninfo_t *" },
597         { "nfsv4", "op-lock-done", 1, 0, "struct compound_state *",
598           "nfsv4opinfo_t *" },
599         { "nfsv4", "op-lock-done", 2, 1, "LOCK4res *" },
600         { "nfsv4", "op-lockt-start", 0, 0, "struct compound_state *",
601           "conninfo_t *" },
602         { "nfsv4", "op-lockt-start", 1, 0, "struct compound_state *",
603           "nfsv4opinfo_t *" },
604         { "nfsv4", "op-lockt-start", 2, 1, "LOCKT4args *" },
605         { "nfsv4", "op-lockt-done", 0, 0, "struct compound_state *",
606           "conninfo_t *" },
607         { "nfsv4", "op-lockt-done", 1, 0, "struct compound_state *",
608           "nfsv4opinfo_t *" },
609         { "nfsv4", "op-lockt-done", 2, 1, "LOCKT4res *" },
610         { "nfsv4", "op-locku-start", 0, 0, "struct compound_state *",
611           "conninfo_t *" },
612         { "nfsv4", "op-locku-start", 1, 0, "struct compound_state *",
613           "nfsv4opinfo_t *" },
614         { "nfsv4", "op-locku-start", 2, 1, "LOCKU4args *" },
615         { "nfsv4", "op-locku-done", 0, 0, "struct compound_state *",
616           "conninfo_t *" },
617         { "nfsv4", "op-locku-done", 1, 0, "struct compound_state *",
618           "nfsv4opinfo_t *" },
619         { "nfsv4", "op-locku-done", 2, 1, "LOCKU4res *" },
```

```
620         { "nfsv4", "op-lookup-start", 0, 0, "struct compound_state *",
621           "conninfo_t *" },
622         { "nfsv4", "op-lookup-start", 1, 0, "struct compound_state *",
623           "nfsv4opinfo_t *" },
624         { "nfsv4", "op-lookup-start", 2, 1, "LOOKUP4args *" },
625         { "nfsv4", "op-lookup-done", 0, 0, "struct compound_state *",
626           "conninfo_t *" },
627         { "nfsv4", "op-lookup-done", 1, 0, "struct compound_state *",
628           "nfsv4opinfo_t *" },
629         { "nfsv4", "op-lookup-done", 2, 1, "LOOKUP4res *" },
630         { "nfsv4", "op-lookupp-start", 0, 0, "struct compound_state *",
631           "conninfo_t *" },
632         { "nfsv4", "op-lookupp-start", 1, 0, "struct compound_state *",
633           "nfsv4opinfo_t *" },
634         { "nfsv4", "op-lookupp-done", 0, 0, "struct compound_state *",
635           "conninfo_t *" },
636         { "nfsv4", "op-lookupp-done", 1, 0, "struct compound_state *",
637           "nfsv4opinfo_t *" },
638         { "nfsv4", "op-lookupp-done", 2, 1, "LOOKUPP4res *" },
639         { "nfsv4", "op-nverify-start", 0, 0, "struct compound_state *",
640           "conninfo_t *" },
641         { "nfsv4", "op-nverify-start", 1, 0, "struct compound_state *",
642           "nfsv4opinfo_t *" },
643         { "nfsv4", "op-nverify-start", 2, 1, "NVERIFY4args *" },
644         { "nfsv4", "op-nverify-done", 0, 0, "struct compound_state *",
645           "conninfo_t *" },
646         { "nfsv4", "op-nverify-done", 1, 0, "struct compound_state *",
647           "nfsv4opinfo_t *" },
648         { "nfsv4", "op-nverify-done", 2, 1, "NVERIFY4res *" },
649         { "nfsv4", "op-open-start", 0, 0, "struct compound_state *",
650           "conninfo_t *" },
651         { "nfsv4", "op-open-start", 1, 0, "struct compound_state *",
652           "nfsv4opinfo_t *" },
653         { "nfsv4", "op-open-start", 2, 1, "OPEN4args *" },
654         { "nfsv4", "op-open-done", 0, 0, "struct compound_state *",
655           "conninfo_t *" },
656         { "nfsv4", "op-open-done", 1, 0, "struct compound_state *",
657           "nfsv4opinfo_t *" },
658         { "nfsv4", "op-open-done", 2, 1, "OPEN4res *" },
659         { "nfsv4", "op-open-confirm-start", 0, 0, "struct compound_state *",
660           "conninfo_t *" },
661         { "nfsv4", "op-open-confirm-start", 1, 0, "struct compound_state *",
662           "nfsv4opinfo_t *" },
663         { "nfsv4", "op-open-confirm-start", 2, 1, "OPEN_CONFIRM4args *" },
664         { "nfsv4", "op-open-confirm-done", 0, 0, "struct compound_state *",
665           "conninfo_t *" },
666         { "nfsv4", "op-open-confirm-done", 1, 0, "struct compound_state *",
667           "nfsv4opinfo_t *" },
668         { "nfsv4", "op-open-confirm-done", 2, 1, "OPEN_CONFIRM4res *" },
669         { "nfsv4", "op-open-downgrade-start", 0, 0, "struct compound_state *",
670           "conninfo_t *" },
671         { "nfsv4", "op-open-downgrade-start", 1, 0, "struct compound_state *",
672           "nfsv4opinfo_t *" },
673         { "nfsv4", "op-open-downgrade-start", 2, 1, "OPEN_DOWNGRADE4args *" },
674         { "nfsv4", "op-open-downgrade-done", 0, 0, "struct compound_state *",
675           "conninfo_t *" },
676         { "nfsv4", "op-open-downgrade-done", 1, 0, "struct compound_state *",
677           "nfsv4opinfo_t *" },
678         { "nfsv4", "op-open-downgrade-done", 2, 1, "OPEN_DOWNGRADE4res *" },
679         { "nfsv4", "op-openattr-start", 0, 0, "struct compound_state *",
680           "conninfo_t *" },
681         { "nfsv4", "op-openattr-start", 1, 0, "struct compound_state *",
682           "nfsv4opinfo_t *" },
683         { "nfsv4", "op-openattr-start", 2, 1, "OPENATTR4args *" },
684         { "nfsv4", "op-openattr-done", 0, 0, "struct compound_state *",
685           "conninfo_t *" },
```



```
686         { "nfsv4", "op-openattr-done", 1, 0, "struct compound_state *",
687           "nfsv4opinfo_t *" },
688         { "nfsv4", "op-openattr-done", 2, 1, "OPENATTR4res *" },
689         { "nfsv4", "op-putfh-start", 0, 0, "struct compound_state *",
690           "conninfo_t *" },
691         { "nfsv4", "op-putfh-start", 1, 0, "struct compound_state *",
692           "nfsv4opinfo_t *" },
693         { "nfsv4", "op-putfh-start", 2, 1, "PUTFH4args *" },
694         { "nfsv4", "op-putfh-done", 0, 0, "struct compound_state *",
695           "conninfo_t *" },
696         { "nfsv4", "op-putfh-done", 1, 0, "struct compound_state *",
697           "nfsv4opinfo_t *" },
698         { "nfsv4", "op-putfh-done", 2, 1, "PUTFH4res *" },
699         { "nfsv4", "op-putpubfh-start", 0, 0, "struct compound_state *",
700           "conninfo_t *" },
701         { "nfsv4", "op-putpubfh-start", 1, 0, "struct compound_state *",
702           "nfsv4opinfo_t *" },
703         { "nfsv4", "op-putpubfh-done", 0, 0, "struct compound_state *",
704           "conninfo_t *" },
705         { "nfsv4", "op-putpubfh-done", 1, 0, "struct compound_state *",
706           "nfsv4opinfo_t *" },
707         { "nfsv4", "op-putpubfh-done", 2, 1, "PUTPUBFH4res *" },
708         { "nfsv4", "op-putrootfh-start", 0, 0, "struct compound_state *",
709           "conninfo_t *" },
710         { "nfsv4", "op-putrootfh-start", 1, 0, "struct compound_state *",
711           "nfsv4opinfo_t *" },
712         { "nfsv4", "op-putrootfh-done", 0, 0, "struct compound_state *",
713           "conninfo_t *" },
714         { "nfsv4", "op-putrootfh-done", 1, 0, "struct compound_state *",
715           "nfsv4opinfo_t *" },
716         { "nfsv4", "op-putrootfh-done", 2, 1, "PUTROOTFH4res *" },
717         { "nfsv4", "op-read-start", 0, 0, "struct compound_state *",
718           "conninfo_t *" },
719         { "nfsv4", "op-read-start", 1, 0, "struct compound_state *",
720           "nfsv4opinfo_t *" },
721         { "nfsv4", "op-read-start", 2, 1, "READ4args *" },
722         { "nfsv4", "op-read-done", 0, 0, "struct compound_state *",
723           "conninfo_t *" },
724         { "nfsv4", "op-read-done", 1, 0, "struct compound_state *",
725           "nfsv4opinfo_t *" },
726         { "nfsv4", "op-read-done", 2, 1, "READ4res *" },
727         { "nfsv4", "op-readdir-start", 0, 0, "struct compound_state *",
728           "conninfo_t *" },
729         { "nfsv4", "op-readdir-start", 1, 0, "struct compound_state *",
730           "nfsv4opinfo_t *" },
731         { "nfsv4", "op-readdir-start", 2, 1, "READDIR4args *" },
732         { "nfsv4", "op-readdir-done", 0, 0, "struct compound_state *",
733           "conninfo_t *" },
734         { "nfsv4", "op-readdir-done", 1, 0, "struct compound_state *",
735           "nfsv4opinfo_t *" },
736         { "nfsv4", "op-readdir-done", 2, 1, "READDIR4res *" },
737         { "nfsv4", "op-readlink-start", 0, 0, "struct compound_state *",
738           "conninfo_t *" },
739         { "nfsv4", "op-readlink-start", 1, 0, "struct compound_state *",
740           "nfsv4opinfo_t *" },
741         { "nfsv4", "op-readlink-done", 0, 0, "struct compound_state *",
742           "conninfo_t *" },
743         { "nfsv4", "op-readlink-done", 1, 0, "struct compound_state *",
744           "nfsv4opinfo_t *" },
745         { "nfsv4", "op-readlink-done", 2, 1, "READLINK4res *" },
746         { "nfsv4", "op-release-lockowner-start", 0, 0,
747           "struct compound_state *", "conninfo_t *" },
748         { "nfsv4", "op-release-lockowner-start", 1, 0,
749           "struct compound_state *", "nfsv4opinfo_t *" },
750         { "nfsv4", "op-release-lockowner-start", 2, 1,
751           "RELEASE_LOCKOWNER4args *" },
```

```
752         { "nfsv4", "op-release-lockowner-done", 0, 0,
753           "struct compound_state *", "conninfo_t *" },
754         { "nfsv4", "op-release-lockowner-done", 1, 0,
755           "struct compound_state *", "nfsv4opinfo_t *" },
756         { "nfsv4", "op-release-lockowner-done", 2, 1,
757           "RELEASE_LOCKOWNER4res *" },
758         { "nfsv4", "op-remove-start", 0, 0, "struct compound_state *",
759           "conninfo_t *" },
760         { "nfsv4", "op-remove-start", 1, 0, "struct compound_state *",
761           "nfsv4opinfo_t *" },
762         { "nfsv4", "op-remove-start", 2, 1, "REMOVE4args *" },
763         { "nfsv4", "op-remove-done", 0, 0, "struct compound_state *",
764           "conninfo_t *" },
765         { "nfsv4", "op-remove-done", 1, 0, "struct compound_state *",
766           "nfsv4opinfo_t *" },
767         { "nfsv4", "op-remove-done", 2, 1, "REMOVE4res *" },
768         { "nfsv4", "op-rename-start", 0, 0, "struct compound_state *",
769           "conninfo_t *" },
770         { "nfsv4", "op-rename-start", 1, 0, "struct compound_state *",
771           "nfsv4opinfo_t *" },
772         { "nfsv4", "op-rename-start", 2, 1, "RENAME4args *" },
773         { "nfsv4", "op-rename-done", 0, 0, "struct compound_state *",
774           "conninfo_t *" },
775         { "nfsv4", "op-rename-done", 1, 0, "struct compound_state *",
776           "nfsv4opinfo_t *" },
777         { "nfsv4", "op-rename-done", 2, 1, "RENAME4res *" },
778         { "nfsv4", "op-renew-start", 0, 0, "struct compound_state *",
779           "conninfo_t *" },
780         { "nfsv4", "op-renew-start", 1, 0, "struct compound_state *",
781           "nfsv4opinfo_t *" },
782         { "nfsv4", "op-renew-start", 2, 1, "RENEW4args *" },
783         { "nfsv4", "op-renew-done", 0, 0, "struct compound_state *",
784           "conninfo_t *" },
785         { "nfsv4", "op-renew-done", 1, 0, "struct compound_state *",
786           "nfsv4opinfo_t *" },
787         { "nfsv4", "op-renew-done", 2, 1, "RENEW4res *" },
788         { "nfsv4", "op-restorefh-start", 0, 0, "struct compound_state *",
789           "conninfo_t *" },
790         { "nfsv4", "op-restorefh-start", 1, 0, "struct compound_state *",
791           "nfsv4opinfo_t *" },
792         { "nfsv4", "op-restorefh-done", 0, 0, "struct compound_state *",
793           "conninfo_t *" },
794         { "nfsv4", "op-restorefh-done", 1, 0, "struct compound_state *",
795           "nfsv4opinfo_t *" },
796         { "nfsv4", "op-restorefh-done", 2, 1, "RESTOREFH4res *" },
797         { "nfsv4", "op-savefh-start", 0, 0, "struct compound_state *",
798           "conninfo_t *" },
799         { "nfsv4", "op-savefh-start", 1, 0, "struct compound_state *",
800           "nfsv4opinfo_t *" },
801         { "nfsv4", "op-savefh-done", 0, 0, "struct compound_state *",
802           "conninfo_t *" },
803         { "nfsv4", "op-savefh-done", 1, 0, "struct compound_state *",
804           "nfsv4opinfo_t *" },
805         { "nfsv4", "op-savefh-done", 2, 1, "SAVEFH4res *" },
806         { "nfsv4", "op-secinfo-start", 0, 0, "struct compound_state *",
807           "conninfo_t *" },
808         { "nfsv4", "op-secinfo-start", 1, 0, "struct compound_state *",
809           "nfsv4opinfo_t *" },
810         { "nfsv4", "op-secinfo-start", 2, 1, "SECINFO4args *" },
811         { "nfsv4", "op-secinfo-done", 0, 0, "struct compound_state *",
812           "conninfo_t *" },
813         { "nfsv4", "op-secinfo-done", 1, 0, "struct compound_state *",
814           "nfsv4opinfo_t *" },
815         { "nfsv4", "op-secinfo-done", 2, 1, "SECINFO4res *" },
816         { "nfsv4", "op-setattr-start", 0, 0, "struct compound_state *",
817           "conninfo_t *" },
```



```
818         { "nfsv4", "op-setattr-start", 1, 0, "struct compound_state *",
819           "nfsv4opinfo_t *" },
820         { "nfsv4", "op-setattr-start", 2, 1, "SETATTR4args *" },
821         { "nfsv4", "op-setattr-done", 0, 0, "struct compound_state *",
822           "conninfo_t *" },
823         { "nfsv4", "op-setattr-done", 1, 0, "struct compound_state *",
824           "nfsv4opinfo_t *" },
825         { "nfsv4", "op-setattr-done", 2, 1, "SETATTR4res *" },
826         { "nfsv4", "op-setclientid-start", 0, 0, "struct compound_state *",
827           "conninfo_t *" },
828         { "nfsv4", "op-setclientid-start", 1, 0, "struct compound_state *",
829           "nfsv4opinfo_t *" },
830         { "nfsv4", "op-setclientid-start", 2, 1, "SETCLIENTID4args *" },
831         { "nfsv4", "op-setclientid-done", 0, 0, "struct compound_state *",
832           "conninfo_t *" },
833         { "nfsv4", "op-setclientid-done", 1, 0, "struct compound_state *",
834           "nfsv4opinfo_t *" },
835         { "nfsv4", "op-setclientid-done", 2, 1, "SETCLIENTID4res *" },
836         { "nfsv4", "op-setclientid-confirm-start", 0, 0,
837           "struct compound_state *", "conninfo_t *" },
838         { "nfsv4", "op-setclientid-confirm-start", 1, 0,
839           "struct compound_state *", "nfsv4opinfo_t *" },
840         { "nfsv4", "op-setclientid-confirm-start", 2, 1,
841           "SETCLIENTID_CONFIRM4args *" },
842         { "nfsv4", "op-setclientid-confirm-done", 0, 0,
843           "struct compound_state *", "conninfo_t *" },
844         { "nfsv4", "op-setclientid-confirm-done", 1, 0,
845           "struct compound_state *", "nfsv4opinfo_t *" },
846         { "nfsv4", "op-setclientid-confirm-done", 2, 1,
847           "SETCLIENTID_CONFIRM4res *" },
848         { "nfsv4", "op-verify-start", 0, 0, "struct compound_state *",
849           "conninfo_t *" },
850         { "nfsv4", "op-verify-start", 1, 0, "struct compound_state *",
851           "nfsv4opinfo_t *" },
852         { "nfsv4", "op-verify-start", 2, 1, "VERIFY4args *" },
853         { "nfsv4", "op-verify-done", 0, 0, "struct compound_state *",
854           "conninfo_t *" },
855         { "nfsv4", "op-verify-done", 1, 0, "struct compound_state *",
856           "nfsv4opinfo_t *" },
857         { "nfsv4", "op-verify-done", 2, 1, "VERIFY4res *" },
858         { "nfsv4", "op-write-start", 0, 0, "struct compound_state *",
859           "conninfo_t *" },
860         { "nfsv4", "op-write-start", 1, 0, "struct compound_state *",
861           "nfsv4opinfo_t *" },
862         { "nfsv4", "op-write-start", 2, 1, "WRITE4args *" },
863         { "nfsv4", "op-write-done", 0, 0, "struct compound_state *",
864           "conninfo_t *" },
865         { "nfsv4", "op-write-done", 1, 0, "struct compound_state *",
866           "nfsv4opinfo_t *" },
867         { "nfsv4", "op-write-done", 2, 1, "WRITE4res *" },
868         { "nfsv4", "cb-recall-start", 0, 0, "rfs4_client_t *",
869           "conninfo_t *" },
870         { "nfsv4", "cb-recall-start", 1, 1, "rfs4_deleg_state_t *",
871           "nfsv4cbinfo_t *" },
872         { "nfsv4", "cb-recall-start", 2, 2, "CB_RECALL4args *" },
873         { "nfsv4", "cb-recall-done", 0, 0, "rfs4_client_t *",
874           "conninfo_t *" },
875         { "nfsv4", "cb-recall-done", 1, 1, "rfs4_deleg_state_t *",
876           "nfsv4cbinfo_t *" },
877         { "nfsv4", "cb-recall-done", 2, 2, "CB_RECALL4res *" },

879         { "ip", "send", 0, 0, "mblk_t *", "pktinfo_t *" },
880         { "ip", "send", 1, 1, "conn_t *", "csinfo_t *" },
881         { "ip", "send", 2, 2, "void_ip_t *", "ipinfo_t *" },
882         { "ip", "send", 3, 3, "__dtrace_ipsr_ill_t *", "ifinfo_t *" },
883         { "ip", "send", 4, 4, "ipha_t *", "ipv4info_t *" },
```

```
884         { "ip", "send", 5, 5, "ip6_t *", "ipv6info_t *" },
885         { "ip", "send", 6, 6, "int" }, /* used by __dtrace_ipsr_ill_t */
886         { "ip", "receive", 0, 0, "mblk_t *", "pktinfo_t *" },
887         { "ip", "receive", 1, 1, "conn_t *", "csinfo_t *" },
888         { "ip", "receive", 2, 2, "void_ip_t *", "ipinfo_t *" },
889         { "ip", "receive", 3, 3, "__dtrace_ipsr_ill_t *", "ifinfo_t *" },
890         { "ip", "receive", 4, 4, "ipha_t *", "ipv4info_t *" },
891         { "ip", "receive", 5, 5, "ip6_t *", "ipv6info_t *" },
892         { "ip", "receive", 6, 6, "int" }, /* used by __dtrace_ipsr_ill_t */

894         { "tcp", "connect-established", 0, 0, "mblk_t *", "pktinfo_t *" },
895         { "tcp", "connect-established", 1, 1, "ip_xmit_attr_t *",
896           "csinfo_t *" },
897         { "tcp", "connect-established", 2, 2, "void_ip_t *", "ipinfo_t *" },
898         { "tcp", "connect-established", 3, 3, "tcp_t *", "tcpsinfo_t *" },
899         { "tcp", "connect-established", 4, 4, "tcph_t *", "tcpinfo_t *" },
900         { "tcp", "connect-refused", 0, 0, "mblk_t *", "pktinfo_t *" },
901         { "tcp", "connect-refused", 1, 1, "ip_xmit_attr_t *", "csinfo_t *" },
902         { "tcp", "connect-refused", 2, 2, "void_ip_t *", "ipinfo_t *" },
903         { "tcp", "connect-refused", 3, 3, "tcp_t *", "tcpsinfo_t *" },
904         { "tcp", "connect-refused", 4, 4, "tcph_t *", "tcpinfo_t *" },
905         { "tcp", "connect-request", 0, 0, "mblk_t *", "pktinfo_t *" },
906         { "tcp", "connect-request", 1, 1, "ip_xmit_attr_t *", "csinfo_t *" },
907         { "tcp", "connect-request", 2, 2, "void_ip_t *", "ipinfo_t *" },
908         { "tcp", "connect-request", 3, 3, "tcp_t *", "tcpsinfo_t *" },
909         { "tcp", "connect-request", 4, 4, "tcph_t *", "tcpinfo_t *" },
910         { "tcp", "accept-established", 0, 0, "mblk_t *", "pktinfo_t *" },
911         { "tcp", "accept-established", 1, 1, "ip_xmit_attr_t *", "csinfo_t *" },
912         { "tcp", "accept-established", 2, 2, "void_ip_t *", "ipinfo_t *" },
913         { "tcp", "accept-established", 3, 3, "tcp_t *", "tcpsinfo_t *" },
914         { "tcp", "accept-established", 4, 4, "tcph_t *", "tcpinfo_t *" },
915         { "tcp", "accept-refused", 0, 0, "mblk_t *", "pktinfo_t *" },
916         { "tcp", "accept-refused", 1, 1, "ip_xmit_attr_t *", "csinfo_t *" },
917         { "tcp", "accept-refused", 2, 2, "void_ip_t *", "ipinfo_t *" },
918         { "tcp", "accept-refused", 3, 3, "tcp_t *", "tcpsinfo_t *" },
919         { "tcp", "accept-refused", 4, 4, "tcph_t *", "tcpinfo_t *" },
920         { "tcp", "state-change", 0, 0, "void", "void" },
921         { "tcp", "state-change", 1, 1, "ip_xmit_attr_t *", "csinfo_t *" },
922         { "tcp", "state-change", 2, 2, "void", "void" },
923         { "tcp", "state-change", 3, 3, "tcp_t *", "tcpsinfo_t *" },
924         { "tcp", "state-change", 4, 4, "void", "void" },
925         { "tcp", "state-change", 5, 5, "int32_t", "tcplsinfo_t *" },
926         { "tcp", "send", 0, 0, "mblk_t *", "pktinfo_t *" },
927         { "tcp", "send", 1, 1, "ip_xmit_attr_t *", "csinfo_t *" },
928         { "tcp", "send", 2, 2, "__dtrace_tcp_void_ip_t *", "ipinfo_t *" },
929         { "tcp", "send", 3, 3, "tcp_t *", "tcpsinfo_t *" },
930         { "tcp", "send", 4, 4, "__dtrace_tcp_tcph_t *", "tcpinfo_t *" },
931         { "tcp", "receive", 0, 0, "mblk_t *", "pktinfo_t *" },
932         { "tcp", "receive", 1, 1, "ip_xmit_attr_t *", "csinfo_t *" },
933         { "tcp", "receive", 2, 2, "__dtrace_tcp_void_ip_t *", "ipinfo_t *" },
934         { "tcp", "receive", 3, 3, "tcp_t *", "tcpsinfo_t *" },
935         { "tcp", "receive", 4, 4, "__dtrace_tcp_tcph_t *", "tcpinfo_t *" },

937         { "udp", "send", 0, 0, "mblk_t *", "pktinfo_t *" },
938         { "udp", "send", 1, 1, "ip_xmit_attr_t *", "csinfo_t *" },
939         { "udp", "send", 2, 2, "void_ip_t *", "ipinfo_t *" },
940         { "udp", "send", 3, 3, "udp_t *", "udpsinfo_t *" },
941         { "udp", "send", 4, 4, "udpha_t *", "udpinfo_t *" },
942         { "udp", "receive", 0, 0, "mblk_t *", "pktinfo_t *" },
943         { "udp", "receive", 1, 1, "ip_xmit_attr_t *", "csinfo_t *" },
944         { "udp", "receive", 2, 2, "void_ip_t *", "ipinfo_t *" },
945         { "udp", "receive", 3, 3, "udp_t *", "udpsinfo_t *" },
946         { "udp", "receive", 4, 4, "udpha_t *", "udpinfo_t *" },

948         { "sctp", "send", 0, 0, "mblk_t *", "pktinfo_t *" },
949         { "sctp", "send", 1, 1, "ip_xmit_attr_t *", "csinfo_t *" },
```



```
950         { "sctp", "send", 2, 2, "void_ip_t *", "ipinfo_t *" },
951         { "sctp", "send", 3, 3, "sctp_t *", "sctpsinfo_t *" },
952         { "sctp", "send", 4, 4, "sctp_hdr_t *", "sctpinfo_t *" },

954 #endif /* ! codereview */
955         { "sysevent", "post", 0, 0, "evch_bind_t *", "syseventchaninfo_t *" },
956         { "sysevent", "post", 1, 1, "sysevent_impl_t *", "syseventinfo_t *" },

958         { "xpv", "add-to-physmap-end", 0, 0, "int" },
959         { "xpv", "add-to-physmap-start", 0, 0, "domid_t" },
960         { "xpv", "add-to-physmap-start", 1, 1, "uint_t" },
961         { "xpv", "add-to-physmap-start", 2, 2, "ulong_t" },
962         { "xpv", "add-to-physmap-start", 3, 3, "ulong_t" },
963         { "xpv", "decrease-reservation-end", 0, 0, "int" },
964         { "xpv", "decrease-reservation-start", 0, 0, "domid_t" },
965         { "xpv", "decrease-reservation-start", 1, 1, "ulong_t" },
966         { "xpv", "decrease-reservation-start", 2, 2, "uint_t" },
967         { "xpv", "decrease-reservation-start", 3, 3, "ulong_t *" },
968         { "xpv", "dom-create-start", 0, 0, "xen_domctl_t *" },
969         { "xpv", "dom-destroy-start", 0, 0, "domid_t" },
970         { "xpv", "dom-pause-start", 0, 0, "domid_t" },
971         { "xpv", "dom-unpause-start", 0, 0, "domid_t" },
972         { "xpv", "dom-create-end", 0, 0, "int" },
973         { "xpv", "dom-destroy-end", 0, 0, "int" },
974         { "xpv", "dom-pause-end", 0, 0, "int" },
975         { "xpv", "dom-unpause-end", 0, 0, "int" },
976         { "xpv", "evtchn-op-end", 0, 0, "int" },
977         { "xpv", "evtchn-op-start", 0, 0, "int" },
978         { "xpv", "evtchn-op-start", 1, 1, "void *" },
979         { "xpv", "increase-reservation-end", 0, 0, "int" },
980         { "xpv", "increase-reservation-start", 0, 0, "domid_t" },
981         { "xpv", "increase-reservation-start", 1, 1, "ulong_t" },
982         { "xpv", "increase-reservation-start", 2, 2, "uint_t" },
983         { "xpv", "increase-reservation-start", 3, 3, "ulong_t *" },
984         { "xpv", "mmap-end", 0, 0, "int" },
985         { "xpv", "mmap-entry", 0, 0, "ulong_t" },
986         { "xpv", "mmap-entry", 1, 1, "ulong_t" },
987         { "xpv", "mmap-entry", 2, 2, "ulong_t" },
988         { "xpv", "mmap-start", 0, 0, "domid_t" },
989         { "xpv", "mmap-start", 1, 1, "int" },
990         { "xpv", "mmap-start", 2, 2, "privcmd_mmap_entry_t *" },
991         { "xpv", "mmapbatch-end", 0, 0, "int" },
992         { "xpv", "mmapbatch-end", 1, 1, "struct seg *" },
993         { "xpv", "mmapbatch-end", 2, 2, "caddr_t" },
994         { "xpv", "mmapbatch-start", 0, 0, "domid_t" },
995         { "xpv", "mmapbatch-start", 1, 1, "int" },
996         { "xpv", "mmapbatch-start", 2, 2, "caddr_t" },
997         { "xpv", "mmu-ext-op-end", 0, 0, "int" },
998         { "xpv", "mmu-ext-op-start", 0, 0, "int" },
999         { "xpv", "mmu-ext-op-start", 1, 1, "struct mmuext_op *" },
1000        { "xpv", "mmu-update-start", 0, 0, "int" },
1001        { "xpv", "mmu-update-start", 1, 1, "int" },
1002        { "xpv", "mmu-update-start", 2, 2, "mmu_update_t *" },
1003        { "xpv", "mmu-update-end", 0, 0, "int" },
1004        { "xpv", "populate-physmap-end", 0, 0, "int" },
1005        { "xpv", "populate-physmap-start", 0, 0, "domid_t" },
1006        { "xpv", "populate-physmap-start", 1, 1, "ulong_t" },
1007        { "xpv", "populate-physmap-start", 2, 2, "ulong_t *" },
1008        { "xpv", "set-memory-map-end", 0, 0, "int" },
1009        { "xpv", "set-memory-map-start", 0, 0, "domid_t" },
1010        { "xpv", "set-memory-map-start", 1, 1, "int" },
1011        { "xpv", "set-memory-map-start", 2, 2, "struct xen_memory_map *" },
1012        { "xpv", "setvcpucontext-end", 0, 0, "int" },
1013        { "xpv", "setvcpucontext-start", 0, 0, "domid_t" },
1014        { "xpv", "setvcpucontext-start", 1, 1, "vcpu_guest_context_t *" },
```

```
1016         { "srp", "service-up", 0, 0, "srpt_session_t *", "conninfo_t *" },
1017         { "srp", "service-up", 1, 0, "srpt_session_t *", "srp_portinfo_t *" },
1018         { "srp", "service-down", 0, 0, "srpt_session_t *", "conninfo_t *" },
1019         { "srp", "service-down", 1, 0, "srpt_session_t *",
1020           "srp_portinfo_t *" },
1021         { "srp", "login-command", 0, 0, "srpt_session_t *", "conninfo_t *" },
1022         { "srp", "login-command", 1, 0, "srpt_session_t *",
1023           "srp_portinfo_t *" },
1024         { "srp", "login-command", 2, 1, "srp_login_req_t *",
1025           "srp_logininfo_t *" },
1026         { "srp", "login-response", 0, 0, "srpt_session_t *", "conninfo_t *" },
1027         { "srp", "login-response", 1, 0, "srpt_session_t *",
1028           "srp_portinfo_t *" },
1029         { "srp", "login-response", 2, 1, "srp_login_rsp_t *",
1030           "srp_logininfo_t *" },
1031         { "srp", "login-response", 3, 2, "srp_login_rej_t *" },
1032         { "srp", "logout-command", 0, 0, "srpt_channel_t *", "conninfo_t *" },
1033         { "srp", "logout-command", 1, 0, "srpt_channel_t *",
1034           "srp_portinfo_t *" },
1035         { "srp", "task-command", 0, 0, "srpt_channel_t *", "conninfo_t *" },
1036         { "srp", "task-command", 1, 0, "srpt_channel_t *",
1037           "srp_portinfo_t *" },
1038         { "srp", "task-command", 2, 1, "srp_cmd_req_t *", "srp_taskinfo_t *" },
1039         { "srp", "task-response", 0, 0, "srpt_channel_t *", "conninfo_t *" },
1040         { "srp", "task-response", 1, 0, "srpt_channel_t *",
1041           "srp_portinfo_t *" },
1042         { "srp", "task-response", 2, 1, "srp_rsp_t *", "srp_taskinfo_t *" },
1043         { "srp", "task-response", 3, 2, "scsi_task_t *" },
1044         { "srp", "task-response", 4, 3, "int8_t" },
1045         { "srp", "scsi-command", 0, 0, "srpt_channel_t *", "conninfo_t *" },
1046         { "srp", "scsi-command", 1, 0, "srpt_channel_t *",
1047           "srp_portinfo_t *" },
1048         { "srp", "scsi-command", 2, 1, "scsi_task_t *", "scsicmd_t *" },
1049         { "srp", "scsi-command", 3, 2, "srp_cmd_req_t *", "srp_taskinfo_t *" },
1050         { "srp", "scsi-response", 0, 0, "srpt_channel_t *", "conninfo_t *" },
1051         { "srp", "scsi-response", 1, 0, "srpt_channel_t *",
1052           "srp_portinfo_t *" },
1053         { "srp", "scsi-response", 2, 1, "srp_rsp_t *", "srp_taskinfo_t *" },
1054         { "srp", "scsi-response", 3, 2, "scsi_task_t *" },
1055         { "srp", "scsi-response", 4, 3, "int8_t" },
1056         { "srp", "xfer-start", 0, 0, "srpt_channel_t *", "conninfo_t *" },
1057         { "srp", "xfer-start", 1, 0, "srpt_channel_t *",
1058           "srp_portinfo_t *" },
1059         { "srp", "xfer-start", 2, 1, "ibt_wr_ds_t *", "xferinfo_t *" },
1060         { "srp", "xfer-start", 3, 2, "srpt_iu_t *", "srp_taskinfo_t *" },
1061         { "srp", "xfer-start", 4, 3, "ibt_send_wr_t *"},
1062         { "srp", "xfer-start", 5, 4, "uint32_t" },
1063         { "srp", "xfer-start", 6, 5, "uint32_t" },
1064         { "srp", "xfer-start", 7, 6, "uint32_t" },
1065         { "srp", "xfer-start", 8, 7, "uint32_t" },
1066         { "srp", "xfer-done", 0, 0, "srpt_channel_t *", "conninfo_t *" },
1067         { "srp", "xfer-done", 1, 0, "srpt_channel_t *",
1068           "srp_portinfo_t *" },
1069         { "srp", "xfer-done", 2, 1, "ibt_wr_ds_t *", "xferinfo_t *" },
1070         { "srp", "xfer-done", 3, 2, "srpt_iu_t *", "srp_taskinfo_t *" },
1071         { "srp", "xfer-done", 4, 3, "ibt_send_wr_t *"},
1072         { "srp", "xfer-done", 5, 4, "uint32_t" },
1073         { "srp", "xfer-done", 6, 5, "uint32_t" },
1074         { "srp", "xfer-done", 7, 6, "uint32_t" },
1075         { "srp", "xfer-done", 8, 7, "uint32_t" },

1077         { "fc", "link-up",   0, 0, "fct_i_local_port_t *", "conninfo_t *" },
1078         { "fc", "link-down", 0, 0, "fct_i_local_port_t *", "conninfo_t *" },
1079         { "fc", "fabric-login-start", 0, 0, "fct_i_local_port_t *",
1080           "conninfo_t *" },
1081         { "fc", "fabric-login-start", 1, 0, "fct_i_local_port_t *",
```



```
1082           "fc_port_info_t *" },
1083         { "fc", "fabric-login-end", 0, 0, "fct_i_local_port_t *",
1084           "conninfo_t *" },
1085         { "fc", "fabric-login-end", 1, 0, "fct_i_local_port_t *",
1086           "fc_port_info_t *" },
1087         { "fc", "rport-login-start", 0, 0, "fct_cmd_t *",
1088           "conninfo_t *" },
1089         { "fc", "rport-login-start", 1, 1, "fct_local_port_t *",
1090           "fc_port_info_t *" },
1091         { "fc", "rport-login-start", 2, 2, "fct_i_remote_port_t *",
1092           "fc_port_info_t *" },
1093         { "fc", "rport-login-start", 3, 3, "int", "int" },
1094         { "fc", "rport-login-end", 0, 0, "fct_cmd_t *",
1095           "conninfo_t *" },
1096         { "fc", "rport-login-end", 1, 1, "fct_local_port_t *",
1097           "fc_port_info_t *" },
1098         { "fc", "rport-login-end", 2, 2, "fct_i_remote_port_t *",
1099           "fc_port_info_t *" },
1100         { "fc", "rport-login-end", 3, 3, "int", "int" },
1101         { "fc", "rport-login-end", 4, 4, "int", "int" },
1102         { "fc", "rport-logout-start", 0, 0, "fct_cmd_t *",
1103           "conninfo_t *" },
1104         { "fc", "rport-logout-start", 1, 1, "fct_local_port_t *",
1105           "fc_port_info_t *" },
1106         { "fc", "rport-logout-start", 2, 2, "fct_i_remote_port_t *",
1107           "fc_port_info_t *" },
1108         { "fc", "rport-logout-start", 3, 3, "int", "int" },
1109         { "fc", "rport-logout-end", 0, 0, "fct_cmd_t *",
1110           "conninfo_t *" },
1111         { "fc", "rport-logout-end", 1, 1, "fct_local_port_t *",
1112           "fc_port_info_t *" },
1113         { "fc", "rport-logout-end", 2, 2, "fct_i_remote_port_t *",
1114           "fc_port_info_t *" },
1115         { "fc", "rport-logout-end", 3, 3, "int", "int" },
1116         { "fc", "scsi-command", 0, 0, "fct_cmd_t *",
1117           "conninfo_t *" },
1118         { "fc", "scsi-command", 1, 1, "fct_i_local_port_t *",
1119           "fc_port_info_t *" },
1120         { "fc", "scsi-command", 2, 2, "scsi_task_t *",
1121           "scsicmd_t *" },
1122         { "fc", "scsi-command", 3, 3, "fct_i_remote_port_t *",
1123           "fc_port_info_t *" },
1124         { "fc", "scsi-response", 0, 0, "fct_cmd_t *",
1125           "conninfo_t *" },
1126         { "fc", "scsi-response", 1, 1, "fct_i_local_port_t *",
1127           "fc_port_info_t *" },
1128         { "fc", "scsi-response", 2, 2, "scsi_task_t *",
1129           "scsicmd_t *" },
1130         { "fc", "scsi-response", 3, 3, "fct_i_remote_port_t *",
1131           "fc_port_info_t *" },
1132         { "fc", "xfer-start", 0, 0, "fct_cmd_t *",
1133           "conninfo_t *" },
1134         { "fc", "xfer-start", 1, 1, "fct_i_local_port_t *",
1135           "fc_port_info_t *" },
1136         { "fc", "xfer-start", 2, 2, "scsi_task_t *",
1137           "scsicmd_t *" },
1138         { "fc", "xfer-start", 3, 3, "fct_i_remote_port_t *",
1139           "fc_port_info_t *" },
1140         { "fc", "xfer-start", 4, 4, "stmf_data_buf_t *",
1141           "fc_xferinfo_t *" },
1142         { "fc", "xfer-done", 0, 0, "fct_cmd_t *",
1143           "conninfo_t *" },
1144         { "fc", "xfer-done", 1, 1, "fct_i_local_port_t *",
1145           "fc_port_info_t *" },
1146         { "fc", "xfer-done", 2, 2, "scsi_task_t *",
1147           "scsicmd_t *" },
```

```
1148         { "fc", "xfer-done", 3, 3, "fct_i_remote_port_t *",
1149           "fc_port_info_t *" },
1150         { "fc", "xfer-done", 4, 4, "stmf_data_buf_t *",
1151           "fc_xferinfo_t *" },
1152         { "fc", "rscn-receive", 0, 0, "fct_i_local_port_t *",
1153           "conninfo_t *" },
1154         { "fc", "rscn-receive", 1, 1, "int", "int"},
1155         { "fc", "abts-receive", 0, 0, "fct_cmd_t *",
1156           "conninfo_t *" },
1157         { "fc", "abts-receive", 1, 1, "fct_i_local_port_t *",
1158           "fc_port_info_t *" },
1159         { "fc", "abts-receive", 2, 2, "fct_i_remote_port_t *",
1160           "fc_port_info_t *" },


1163         { NULL }
1164 };

1166 /*ARGSUSED*/
1167 void
1168 sdt_getargdesc(void *arg, dtrace_id_t id, void *parg, dtrace_argdesc_t *desc)
1169 {
1170         sdt_probe_t *sdp = parg;
1171         int i;

1173         desc->dtargd_native[0] = '\0';
1174         desc->dtargd_xlate[0] = '\0';

1176         for (i = 0; sdt_args[i].sda_provider != NULL; i++) {
1177                 sdt_argdesc_t *a = &sdt_args[i];

1179                 if (strcmp(sdp->sdp_provider->sdtp_name, a->sda_provider) != 0)
1180                         continue;

1182                 if (a->sda_name != NULL &&
1183                     strcmp(sdp->sdp_name, a->sda_name) != 0)
1184                         continue;

1186                 if (desc->dtargd_ndx != a->sda_ndx)
1187                         continue;

1189                 if (a->sda_native != NULL)
1190                         (void) strcpy(desc->dtargd_native, a->sda_native);

1192                 if (a->sda_xlate != NULL)
1193                         (void) strcpy(desc->dtargd_xlate, a->sda_xlate);

1195                 desc->dtargd_mapping = a->sda_mapping;
1196                 return;
1197         }

1199         desc->dtargd_ndx = DTRACE_ARGNONE;
1200 }
```

```
**********************************************************
   22475 Sat Apr 12 11:18:55 2014
new/usr/src/uts/common/inet/sctp/sctp_bind.c
3903 DTrace SCTP Provider
**********************************************************
______unchanged_portion_omitted_

 124 int
 125 sctp_listen(sctp_t *sctp)
 126 {
 127         sctp_tf_t       *tf;
 128         sctp_stack_t    *sctps = sctp->sctp_sctps;
 129         conn_t          *connp = sctp->sctp_connp;

 131         RUN_SCTP(sctp);
 132         /*
 133          * TCP handles listen() increasing the backlog, need to check
 134          * if it should be handled here too
 135          */
 136         if (sctp->sctp_state > SCTPS_BOUND ||
 137             (sctp->sctp_connp->conn_state_flags & CONN_CLOSING)) {
 138                 WAKE_SCTP(sctp);
 139                 return (EINVAL);
 140         }

 142         /* Do an anonymous bind for unbound socket doing listen(). */
 143         if (sctp->sctp_nsaddrs == 0) {
 144                 struct sockaddr_storage ss;
 145                 int ret;

 147                 bzero(&ss, sizeof (ss));
 148                 ss.ss_family = connp->conn_family;

 150                 WAKE_SCTP(sctp);
 151                 if ((ret = sctp_bind(sctp, (struct sockaddr *)&ss,
 152                     sizeof (ss))) != 0)
 153                         return (ret);
 154                 RUN_SCTP(sctp)
 155         }

 157         /* Cache things in the ixa without any refhold */
 158         ASSERT(!(connp->conn_ixa->ixa_free_flags & IXA_FREE_CRED));
 159         connp->conn_ixa->ixa_cred = connp->conn_cred;
 160         connp->conn_ixa->ixa_cpid = connp->conn_cpid;
 161         if (is_system_labeled())
 162                 connp->conn_ixa->ixa_tsl = crgetlabel(connp->conn_cred);

 164         sctp->sctp_state = SCTPS_LISTEN;
 165         DTRACE_SCTP6(state__change, void, NULL, ip_xmit_attr_t *,
 166             connp->conn_ixa, void, NULL, sctp_t *, sctp, void, NULL,
 167             int32_t, SCTPS_BOUND);
 168 #endif /* ! codereview */
 169         (void) random_get_pseudo_bytes(sctp->sctp_secret, SCTP_SECRET_LEN);
 170         sctp->sctp_last_secret_update = ddi_get_lbolt64();
 171         bzero(sctp->sctp_old_secret, SCTP_SECRET_LEN);

 173         /*
 174          * If there is an association limit, allocate and initialize
 175          * the counter struct.  Note that since listen can be called
 176          * multiple times, the struct may have been allready allocated.
 177          */
 178         if (!list_is_empty(&sctps->sctps_listener_conf) &&
 179             sctp->sctp_listen_cnt == NULL) {
 180                 sctp_listen_cnt_t *slc;
 181                 uint32_t ratio;
```



```
 183                 ratio = sctp_find_listener_conf(sctps,
 184                     ntohs(connp->conn_lport));
 185                 if (ratio != 0) {
 186                         uint32_t mem_ratio, tot_buf;

 188                         slc = kmem_alloc(sizeof (sctp_listen_cnt_t), KM_SLEEP);
 189                         /*
 190                          * Calculate the connection limit based on
 191                          * the configured ratio and maxusers.  Maxusers
 192                          * are calculated based on memory size,
 193                          * ~ 1 user per MB.  Note that the conn_rcvbuf
 194                          * and conn_sndbuf may change after a
 195                          * connection is accepted.  So what we have
 196                          * is only an approximation.
 197                          */
 198                         if ((tot_buf = connp->conn_rcvbuf +
 199                             connp->conn_sndbuf) < MB) {
 200                                 mem_ratio = MB / tot_buf;
 201                                 slc->slc_max = maxusers / ratio * mem_ratio;
 202                         } else {
 203                                 mem_ratio = tot_buf / MB;
 204                                 slc->slc_max = maxusers / ratio / mem_ratio;
 205                         }
 206                         /* At least we should allow some associations! */
 207                         if (slc->slc_max < sctp_min_assoc_listener)
 208                                 slc->slc_max = sctp_min_assoc_listener;
 209                         slc->slc_cnt = 1;
 210                         slc->slc_drop = 0;
 211                         sctp->sctp_listen_cnt = slc;
 212                 }
 213         }


 216         tf = &sctps->sctps_listen_fanout[SCTP_LISTEN_HASH(
 217             ntohs(connp->conn_lport))];
 218         sctp_listen_hash_insert(tf, sctp);

 220         WAKE_SCTP(sctp);
 221         return (0);
 222 }

 224 /*
 225  * Bind the sctp_t to a sockaddr, which includes an address and other
 226  * information, such as port or flowinfo.
 227  */
 228 int
 229 sctp_bind(sctp_t *sctp, struct sockaddr *sa, socklen_t len)
 230 {
 231         int             user_specified;
 232         boolean_t       bind_to_req_port_only;
 233         in_port_t       requested_port;
 234         in_port_t       allocated_port;
 235         int             err = 0;
 236         conn_t          *connp = sctp->sctp_connp;
 237         uint_t          scope_id;
 238         sin_t           *sin;
 239         sin6_t          *sin6;

 241         ASSERT(sctp != NULL);

 243         RUN_SCTP(sctp);

 245         if ((sctp->sctp_state >= SCTPS_BOUND) ||
 246             (sctp->sctp_connp->conn_state_flags & CONN_CLOSING) ||
 247             (sa == NULL || len == 0)) {
 248                 /*
```

```
249                  * Multiple binds not allowed for any SCTP socket
250                  * Also binding with null address is not supported.
251                  */
252                 err = EINVAL;
253                 goto done;
254         }

256         switch (sa->sa_family) {
257         case AF_INET:
258                 sin = (sin_t *)sa;
259                 if (len < sizeof (struct sockaddr_in) ||
260                     connp->conn_family == AF_INET6) {
261                         err = EINVAL;
262                         goto done;
263                 }
264                 requested_port = ntohs(sin->sin_port);
265                 break;
266         case AF_INET6:
267                 sin6 = (sin6_t *)sa;
268                 if (len < sizeof (struct sockaddr_in6) ||
269                     connp->conn_family == AF_INET) {
270                         err = EINVAL;
271                         goto done;
272                 }
273                 requested_port = ntohs(sin6->sin6_port);
274                 /* Set the flowinfo. */
275                 connp->conn_flowinfo =
276                     sin6->sin6_flowinfo & ~IPV6_VERS_AND_FLOW_MASK;

278                 scope_id = sin6->sin6_scope_id;
279                 if (scope_id != 0 && IN6_IS_ADDR_LINKSCOPE(&sin6->sin6_addr)) {
280                         connp->conn_ixa->ixa_flags |= IXAF_SCOPEID_SET;
281                         connp->conn_ixa->ixa_scopeid = scope_id;
282                         connp->conn_incoming_ifindex = scope_id;
283                 } else {
284                         connp->conn_ixa->ixa_flags &= ~IXAF_SCOPEID_SET;
285                         connp->conn_incoming_ifindex = connp->conn_bound_if;
286                 }
287                 break;
288         default:
289                 err = EAFNOSUPPORT;
290                 goto done;
291         }
292         bind_to_req_port_only = requested_port == 0 ? B_FALSE : B_TRUE;

294         err = sctp_select_port(sctp, &requested_port, &user_specified);
295         if (err != 0)
296                 goto done;

298         if ((err = sctp_bind_add(sctp, sa, 1, B_TRUE,
299             user_specified == 1 ? htons(requested_port) : 0)) != 0) {
300                 goto done;
301         }
302         err = sctp_bindi(sctp, requested_port, bind_to_req_port_only,
303             user_specified, &allocated_port);
304         if (err != 0) {
305                 sctp_free_saddrs(sctp);
306         } else {
307                 ASSERT(sctp->sctp_state == SCTPS_BOUND);
308         }
309 done:
310         WAKE_SCTP(sctp);
311         return (err);
312 }

314 /*
```



```
315  * Perform bind/unbind operation of a list of addresses on a sctp_t
316  */
317 int
318 sctp_bindx(sctp_t *sctp, const void *addrs, int addrcnt, int bindop)
319 {
320         ASSERT(sctp != NULL);
321         ASSERT(addrs != NULL);
322         ASSERT(addrcnt > 0);

324         switch (bindop) {
325         case SCTP_BINDX_ADD_ADDR:
326                 return (sctp_bind_add(sctp, addrs, addrcnt, B_FALSE,
327                     sctp->sctp_connp->conn_lport));
328         case SCTP_BINDX_REM_ADDR:
329                 return (sctp_bind_del(sctp, addrs, addrcnt, B_FALSE));
330         default:
331                 return (EINVAL);
332         }
333 }

335 /*
336  * Add a list of addresses to a sctp_t.
337  */
338 int
339 sctp_bind_add(sctp_t *sctp, const void *addrs, uint32_t addrcnt,
340     boolean_t caller_hold_lock, in_port_t port)
341 {
342         int             err = 0;
343         boolean_t       do_asconf = B_FALSE;
344         sctp_stack_t    *sctps = sctp->sctp_sctps;
345         conn_t          *connp = sctp->sctp_connp;

347         if (!caller_hold_lock)
348                 RUN_SCTP(sctp);

350         if (sctp->sctp_state > SCTPS_ESTABLISHED ||
351             (sctp->sctp_connp->conn_state_flags & CONN_CLOSING)) {
352                 if (!caller_hold_lock)
353                         WAKE_SCTP(sctp);
354                 return (EINVAL);
355         }

357         if (sctp->sctp_state > SCTPS_LISTEN) {
358                 /*
359                  * Let's do some checking here rather than undoing the
360                  * add later (for these reasons).
361                  */
362                 if (!sctps->sctps_addip_enabled ||
363                     !sctp->sctp_understands_asconf ||
364                     !sctp->sctp_understands_addip) {
365                         if (!caller_hold_lock)
366                                 WAKE_SCTP(sctp);
367                         return (EINVAL);
368                 }
369                 do_asconf = B_TRUE;
370         }
371         /*
372          * On a clustered node, for an inaddr_any bind, we will pass the list
373          * of all the addresses in the global list, minus any address on the
374          * loopback interface, and expect the clustering susbsystem to give us
375          * the correct list for the 'port'. For explicit binds we give the
376          * list of addresses  and the clustering module validates it for the
377          * 'port'.
378          *
379          * On a non-clustered node, cl_sctp_check_addrs will be NULL and
380          * we proceed as usual.
```

```
381          */
382         if (cl_sctp_check_addrs != NULL) {
383                 uchar_t         *addrlist = NULL;
384                 size_t          size = 0;
385                 int             unspec = 0;
386                 boolean_t       do_listen;
387                 uchar_t         *llist = NULL;
388                 size_t          lsize = 0;

390                 /*
391                  * If we are adding addresses after listening, but before
392                  * an association is established, we need to update the
393                  * clustering module with this info.
394                  */
395                 do_listen = !do_asconf && sctp->sctp_state > SCTPS_BOUND &&
396                     cl_sctp_listen != NULL;

398                 err = sctp_get_addrlist(sctp, addrs, &addrcnt, &addrlist,
399                     &unspec, &size);
400                 if (err != 0) {
401                         ASSERT(addrlist == NULL);
402                         ASSERT(addrcnt == 0);
403                         ASSERT(size == 0);
404                         if (!caller_hold_lock)
405                                 WAKE_SCTP(sctp);
406                         SCTP_KSTAT(sctps, sctp_cl_check_addrs);
407                         return (err);
408                 }
409                 ASSERT(addrlist != NULL);
410                 (*cl_sctp_check_addrs)(connp->conn_family, port, &addrlist,
411                     size, &addrcnt, unspec == 1);
412                 if (addrcnt == 0) {
413                         /* We free the list */
414                         kmem_free(addrlist, size);
415                         if (!caller_hold_lock)
416                                 WAKE_SCTP(sctp);
417                         return (EINVAL);
418                 }
419                 if (do_listen) {
420                         lsize = sizeof (in6_addr_t) * addrcnt;
421                         llist = kmem_alloc(lsize, KM_SLEEP);
422                 }
423                 err = sctp_valid_addr_list(sctp, addrlist, addrcnt, llist,
424                     lsize);
425                 if (err == 0 && do_listen) {
426                         (*cl_sctp_listen)(connp->conn_family, llist,
427                             addrcnt, connp->conn_lport);
428                         /* list will be freed by the clustering module */
429                 } else if (err != 0 && llist != NULL) {
430                         kmem_free(llist, lsize);
431                 }
432                 /* free the list we allocated */
433                 kmem_free(addrlist, size);
434         } else {
435                 err = sctp_valid_addr_list(sctp, addrs, addrcnt, NULL, 0);
436         }
437         if (err != 0) {
438                 if (!caller_hold_lock)
439                         WAKE_SCTP(sctp);
440                 return (err);
441         }
442         /* Need to send  ASCONF messages */
443         if (do_asconf) {
444                 err = sctp_add_ip(sctp, addrs, addrcnt);
445                 if (err != 0) {
446                         sctp_del_saddr_list(sctp, addrs, addrcnt, B_FALSE);
```



```
447                         if (!caller_hold_lock)
448                                 WAKE_SCTP(sctp);
449                         return (err);
450                 }
451         }
452         if (!caller_hold_lock)
453                 WAKE_SCTP(sctp);
454         return (0);
455 }

457 /*
458  * Remove one or more addresses bound to the sctp_t.
459  */
460 int
461 sctp_bind_del(sctp_t *sctp, const void *addrs, uint32_t addrcnt,
462     boolean_t caller_hold_lock)
463 {
464         int             error = 0;
465         boolean_t       do_asconf = B_FALSE;
466         uchar_t         *ulist = NULL;
467         size_t          usize = 0;
468         sctp_stack_t    *sctps = sctp->sctp_sctps;
469         conn_t          *connp = sctp->sctp_connp;

471         if (!caller_hold_lock)
472                 RUN_SCTP(sctp);

474         if (sctp->sctp_state > SCTPS_ESTABLISHED ||
475             (sctp->sctp_connp->conn_state_flags & CONN_CLOSING)) {
476                 if (!caller_hold_lock)
477                         WAKE_SCTP(sctp);
478                 return (EINVAL);
479         }
480         /*
481          * Fail the remove if we are beyond listen, but can't send this
482          * to the peer.
483          */
484         if (sctp->sctp_state > SCTPS_LISTEN) {
485                 if (!sctps->sctps_addip_enabled ||
486                     !sctp->sctp_understands_asconf ||
487                     !sctp->sctp_understands_addip) {
488                         if (!caller_hold_lock)
489                                 WAKE_SCTP(sctp);
490                         return (EINVAL);
491                 }
492                 do_asconf = B_TRUE;
493         }

495         /* Can't delete the last address nor all of the addresses */
496         if (sctp->sctp_nsaddrs == 1 || addrcnt >= sctp->sctp_nsaddrs) {
497                 if (!caller_hold_lock)
498                         WAKE_SCTP(sctp);
499                 return (EINVAL);
500         }

502         if (cl_sctp_unlisten != NULL && !do_asconf &&
503             sctp->sctp_state > SCTPS_BOUND) {
504                 usize = sizeof (in6_addr_t) * addrcnt;
505                 ulist = kmem_alloc(usize, KM_SLEEP);
506         }

508         error = sctp_del_ip(sctp, addrs, addrcnt, ulist, usize);
509         if (error != 0) {
510                 if (ulist != NULL)
511                         kmem_free(ulist, usize);
512                 if (!caller_hold_lock)
```

```
513                         WAKE_SCTP(sctp);
514                 return (error);
515         }
516         /* ulist will be non-NULL only if cl_sctp_unlisten is non-NULL */
517         if (ulist != NULL) {
518                 ASSERT(cl_sctp_unlisten != NULL);
519                 (*cl_sctp_unlisten)(connp->conn_family, ulist, addrcnt,
520                     connp->conn_lport);
521                 /* ulist will be freed by the clustering module */
522         }
523         if (!caller_hold_lock)
524                 WAKE_SCTP(sctp);
525         return (error);
526 }

528 /*
529  * Returns 0 for success, errno value otherwise.
530  *
531  * If the "bind_to_req_port_only" parameter is set and the requested port
532  * number is available, then set allocated_port to it.  If not available,
533  * return an error.
534  *
535  * If the "bind_to_req_port_only" parameter is not set and the requested port
536  * number is available, then set allocated_port to it.  If not available,
537  * find the first anonymous port we can and set allocated_port to that.  If no
538  * anonymous ports are available, return an error.
539  *
540  * In either case, when succeeding, update the sctp_t to record the port number
541  * and insert it in the bind hash table.
542  */
543 int
544 sctp_bindi(sctp_t *sctp, in_port_t port, boolean_t bind_to_req_port_only,
545     int user_specified, in_port_t *allocated_port)
546 {
547         /* number of times we have run around the loop */
548         int count = 0;
549         /* maximum number of times to run around the loop */
550         int loopmax;
551         sctp_stack_t    *sctps = sctp->sctp_sctps;
552         conn_t          *connp = sctp->sctp_connp;
553         zone_t *zone = crgetzone(connp->conn_cred);
554         zoneid_t zoneid = connp->conn_zoneid;

556         /*
557          * Lookup for free addresses is done in a loop and "loopmax"
558          * influences how long we spin in the loop
559          */
560         if (bind_to_req_port_only) {
561                 /*
562                  * If the requested port is busy, don't bother to look
563                  * for a new one. Setting loop maximum count to 1 has
564                  * that effect.
565                  */
566                 loopmax = 1;
567         } else {
568                 /*
569                  * If the requested port is busy, look for a free one
570                  * in the anonymous port range.
571                  * Set loopmax appropriately so that one does not look
572                  * forever in the case all of the anonymous ports are in use.
573                  */
574                 loopmax = (sctps->sctps_largest_anon_port -
575                     sctps->sctps_smallest_anon_port + 1);
576         }
577         do {
578                 uint16_t        lport;
```



```
579                 sctp_tf_t       *tbf;
580                 sctp_t          *lsctp;
581                 int             addrcmp;

583                 lport = htons(port);

585                 /*
586                  * Ensure that the sctp_t is not currently in the bind hash.
587                  * Hold the lock on the hash bucket to ensure that
588                  * the duplicate check plus the insertion is an atomic
589                  * operation.
590                  *
591                  * This function does an inline lookup on the bind hash list
592                  * Make sure that we access only members of sctp_t
593                  * and that we don't look at sctp_sctp, since we are not
594                  * doing a SCTPB_REFHOLD. For more details please see the notes
595                  * in sctp_compress()
596                  */
597                 sctp_bind_hash_remove(sctp);
598                 tbf = &sctps->sctps_bind_fanout[SCTP_BIND_HASH(port)];
599                 mutex_enter(&tbf->tf_lock);
600                 for (lsctp = tbf->tf_sctp; lsctp != NULL;
601                     lsctp = lsctp->sctp_bind_hash) {
602                         conn_t *lconnp = lsctp->sctp_connp;

604                         if (lport != lconnp->conn_lport ||
605                             lsctp->sctp_state < SCTPS_BOUND)
606                                 continue;

608                         /*
609                          * On a labeled system, we must treat bindings to ports
610                          * on shared IP addresses by sockets with MAC exemption
611                          * privilege as being in all zones, as there's
612                          * otherwise no way to identify the right receiver.
613                          */
614                         if (lconnp->conn_zoneid != zoneid &&
615                             lconnp->conn_mac_mode == CONN_MAC_DEFAULT &&
616                             connp->conn_mac_mode == CONN_MAC_DEFAULT)
617                                 continue;

619                         addrcmp = sctp_compare_saddrs(sctp, lsctp);
620                         if (addrcmp != SCTP_ADDR_DISJOINT) {
621                                 if (!connp->conn_reuseaddr) {
622                                         /* in use */
623                                         break;
624                                 } else if (lsctp->sctp_state == SCTPS_BOUND ||
625                                     lsctp->sctp_state == SCTPS_LISTEN) {
626                                         /*
627                                          * socket option SO_REUSEADDR is set
628                                          * on the binding sctp_t.
629                                          *
630                                          * We have found a match of IP source
631                                          * address and source port, which is
632                                          * refused regardless of the
633                                          * SO_REUSEADDR setting, so we break.
634                                          */
635                                         break;
636                                 }
637                         }
638                 }
639                 if (lsctp != NULL) {
640                         /* The port number is busy */
641                         mutex_exit(&tbf->tf_lock);
642                 } else {
643                         if (is_system_labeled()) {
644                                 mlp_type_t addrtype, mlptype;
```

```
645                             uint_t ipversion;
647                             /*
648                              * On a labeled system we must check the type
649                              * of the binding requested by the user (either
650                              * MLP or SLP on shared and private addresses),
651                              * and that the user's requested binding
652                              * is permitted.
653                              */
654                             if (connp->conn_family == AF_INET)
655                                     ipversion = IPV4_VERSION;
656                             else
657                                     ipversion = IPV6_VERSION;
659                             addrtype = tsol_mlp_addr_type(
660                                 connp->conn_allzones ? ALL_ZONES :
661                                 zone->zone_id,
662                                 ipversion,
663                                 connp->conn_family == AF_INET ?
664                                 (void *)&sctp->sctp_ipha->ipha_src :
665                                 (void *)&sctp->sctp_ip6h->ip6_src,
666                                 sctps->sctps_netstack->netstack_ip);
668                             /*
669                              * tsol_mlp_addr_type returns the possibilities
670                              * for the selected address.  Since all local
671                              * addresses are either private or shared, the
672                              * return value mlptSingle means "local address
673                              * not valid (interface not present)."
674                              */
675                             if (addrtype == mlptSingle) {
676                                     mutex_exit(&tbf->tf_lock);
677                                     return (EADDRNOTAVAIL);
678                             }
679                             mlptype = tsol_mlp_port_type(zone, IPPROTO_SCTP,
680                                 port, addrtype);
681                             if (mlptype != mlptSingle) {
682                                     if (secpolicy_net_bindmlp(connp->
683                                         conn_cred) != 0) {
684                                             mutex_exit(&tbf->tf_lock);
685                                             return (EACCES);
686                                     }
687                                     /*
688                                      * If we're binding a shared MLP, then
689                                      * make sure that this zone is the one
690                                      * that owns that MLP.  Shared MLPs can
691                                      * be owned by at most one zone.
692                                      *
693                                      * No need to handle exclusive-stack
694                                      * zones since ALL_ZONES only applies
695                                      * to the shared stack.
696                                      */
698                                     if (mlptype == mlptShared &&
699                                         addrtype == mlptShared &&
700                                         connp->conn_zoneid !=
701                                         tsol_mlp_findzone(IPPROTO_SCTP,
702                                         lport)) {
703                                             mutex_exit(&tbf->tf_lock);
704                                             return (EACCES);
705                                     }
706                                     connp->conn_mlp_type = mlptype;
707                             }
708                     }
709                     /*
710                      * This port is ours. Insert in fanout and mark as
```



```
711                      * bound to prevent others from getting the port
712                      * number.
713                      */
714                     sctp->sctp_state = SCTPS_BOUND;
715                     DTRACE_SCTP6(state__change, void, NULL,
716                         ip_xmit_attr_t *, connp->conn_ixa, void, NULL,
717                         scpt_t *, sctp, void, NULL,
718                         int32_t, SCTPS_IDLE);
719 #endif /* ! codereview */
720                     connp->conn_lport = lport;
722                     ASSERT(&sctps->sctps_bind_fanout[
723                         SCTP_BIND_HASH(port)] == tbf);
724                     sctp_bind_hash_insert(tbf, sctp, 1);
726                     mutex_exit(&tbf->tf_lock);
728                     /*
729                      * We don't want sctp_next_port_to_try to "inherit"
730                      * a port number supplied by the user in a bind.
731                      *
732                      * This is the only place where sctp_next_port_to_try
733                      * is updated. After the update, it may or may not
734                      * be in the valid range.
735                      */
736                     if (user_specified == 0)
737                             sctps->sctps_next_port_to_try = port + 1;
739                     *allocated_port = port;
741                     return (0);
742             }
744             if ((count == 0) && (user_specified)) {
745                     /*
746                      * We may have to return an anonymous port. So
747                      * get one to start with.
748                      */
749                     port = sctp_update_next_port(
750                         sctps->sctps_next_port_to_try,
751                         zone, sctps);
752                     user_specified = 0;
753             } else {
754                     port = sctp_update_next_port(port + 1, zone, sctps);
755             }
756             if (port == 0)
757                     break;
759             /*
760              * Don't let this loop run forever in the case where
761              * all of the anonymous ports are in use.
762              */
763         } while (++count < loopmax);
765         return (bind_to_req_port_only ? EADDRINUSE : EADDRNOTAVAIL);
766 }
768 /*
769  * Don't let port fall into the privileged range.
770  * Since the extra privileged ports can be arbitrary we also
771  * ensure that we exclude those from consideration.
772  * sctp_g_epriv_ports is not sorted thus we loop over it until
773  * there are no changes.
774  *
775  * Note: No locks are held when inspecting sctp_g_*epriv_ports
776  * but instead the code relies on:
```

```
 * - the fact that the address of the array and its size never changes
 * - the atomic assignment of the elements of the array
 */
in_port_t
sctp_update_next_port(in_port_t port, zone_t *zone, sctp_stack_t *sctps)
{
        int i;
        boolean_t restart = B_FALSE;

retry:
        if (port < sctps->sctps_smallest_anon_port)
                port = sctps->sctps_smallest_anon_port;

        if (port > sctps->sctps_largest_anon_port) {
                if (restart)
                        return (0);
                restart = B_TRUE;
                port = sctps->sctps_smallest_anon_port;
        }

        if (port < sctps->sctps_smallest_nonpriv_port)
                port = sctps->sctps_smallest_nonpriv_port;

        for (i = 0; i < sctps->sctps_g_num_epriv_ports; i++) {
                if (port == sctps->sctps_g_epriv_ports[i]) {
                        port++;
                        /*
                         * Make sure whether the port is in the
                         * valid range.
                         *
                         * XXX Note that if sctp_g_epriv_ports contains
                         * all the anonymous ports this will be an
                         * infinite loop.
                         */
                        goto retry;
                }
        }

        if (is_system_labeled() &&
            (i = tsol_next_port(zone, port, IPPROTO_SCTP, B_TRUE)) != 0) {
                port = i;
                goto retry;
        }

        return (port);
}
```

```
**********************************************************
   66543 Sat Apr 12 11:18:56 2014
new/usr/src/uts/common/inet/sctp/sctp_output.c
3903 DTrace SCTP Provider
**********************************************************
______unchanged_portion_omitted_

 954 void
 955 sctp_fast_rexmit(sctp_t *sctp)
 956 {
 957         mblk_t          *mp, *head;
 958         int             pktlen = 0;
 959         sctp_faddr_t    *fp = NULL;
 960         sctp_stack_t    *sctps = sctp->sctp_sctps;

 962         ASSERT(sctp->sctp_xmit_head != NULL);
 963         mp = sctp_find_fast_rexmit_mblks(sctp, &pktlen, &fp);
 964         if (mp == NULL) {
 965                 SCTP_KSTAT(sctps, sctp_fr_not_found);
 966                 return;
 967         }
 968         if ((head = sctp_add_proto_hdr(sctp, fp, mp, 0, NULL)) == NULL) {
 969                 freemsg(mp);
 970                 SCTP_KSTAT(sctps, sctp_fr_add_hdr);
 971                 return;
 972         }
 973         if ((pktlen > fp->sf_pmss) && fp->sf_isv4) {
 974                 ipha_t *iph = (ipha_t *)head->b_rptr;

 976                 iph->ipha_fragment_offset_and_flags = 0;
 977         }

 979         sctp_set_iplen(sctp, head, fp->sf_ixa);

 981         DTRACE_SCTP5(send, mblk_t *, NULL, ip_xmit_attr_t *, fp->sf_ixa,
 982             void_ip_t *, mp->b_rptr, sctp_t *, sctp, sctp_hdr_t *,
 983             &mp->b_rptr[fp->sf_ixa->ixa_ip_hdr_length]);

 985 #endif /* ! codereview */
 986         (void) conn_ip_output(head, fp->sf_ixa);
 987         BUMP_LOCAL(sctp->sctp_opkts);
 988         sctp->sctp_active = fp->sf_lastactive = ddi_get_lbolt64();
 989 }

 991 void
 992 sctp_output(sctp_t *sctp, uint_t num_pkt)
 993 {
 994         mblk_t                  *mp = NULL;
 995         mblk_t                  *nmp;
 996         mblk_t                  *head;
 997         mblk_t                  *meta = sctp->sctp_xmit_tail;
 998         mblk_t                  *fill = NULL;
 999         uint16_t                chunklen;
1000         uint32_t                cansend;
1001         int32_t                 seglen;
1002         int32_t                 xtralen;
1003         int32_t                 sacklen;
1004         int32_t                 pad = 0;
1005         int32_t                 pathmax;
1006         int                     extra;
1007         int64_t                 now = LBOLT_FASTPATH64;
1008         sctp_faddr_t            *fp;
1009         sctp_faddr_t            *lfp;
1010         sctp_data_hdr_t         *sdc;
1011         int                     error;
1012         boolean_t               notsent = B_TRUE;
```



```
1013         sctp_stack_t            *sctps = sctp->sctp_sctps;
1014         uint32_t                tsn;

1016         if (sctp->sctp_ftsn == sctp->sctp_lastacked + 1) {
1017                 sacklen = 0;
1018         } else {
1019                 /* send a SACK chunk */
1020                 sacklen = sizeof (sctp_chunk_hdr_t) +
1021                     sizeof (sctp_sack_chunk_t) +
1022                     (sizeof (sctp_sack_frag_t) * sctp->sctp_sack_gaps);
1023                 lfp = sctp->sctp_lastdata;
1024                 ASSERT(lfp != NULL);
1025                 if (lfp->sf_state != SCTP_FADDRS_ALIVE)
1026                         lfp = sctp->sctp_current;
1027         }

1029         cansend = sctp->sctp_frwnd;
1030         if (sctp->sctp_unsent < cansend)
1031                 cansend = sctp->sctp_unsent;

1033         /*
1034          * Start persist timer if unable to send or when
1035          * trying to send into a zero window. This timer
1036          * ensures the blocked send attempt is retried.
1037          */
1038         if ((cansend < sctp->sctp_current->sf_pmss / 2) &&
1039             (sctp->sctp_unacked != 0) &&
1040             (sctp->sctp_unacked < sctp->sctp_current->sf_pmss) &&
1041             !sctp->sctp_ndelay ||
1042             (cansend == 0 && sctp->sctp_unacked == 0 &&
1043             sctp->sctp_unsent != 0)) {
1044                 head = NULL;
1045                 fp = sctp->sctp_current;
1046                 goto unsent_data;
1047         }
1048         if (meta != NULL)
1049                 mp = meta->b_cont;
1050         while (cansend > 0 && num_pkt-- != 0) {
1051                 pad = 0;

1053                 /*
1054                  * Find first segment eligible for transmit.
1055                  */
1056                 while (mp != NULL) {
1057                         if (SCTP_CHUNK_CANSEND(mp))
1058                                 break;
1059                         mp = mp->b_next;
1060                 }
1061                 if (mp == NULL) {
1062                         meta = sctp_get_msg_to_send(sctp, &mp,
1063                             meta == NULL ? NULL : meta->b_next, &error, sacklen,
1064                             cansend, NULL);
1065                         if (error != 0 || meta == NULL) {
1066                                 head = NULL;
1067                                 fp = sctp->sctp_current;
1068                                 goto unsent_data;
1069                         }
1070                         sctp->sctp_xmit_tail =  meta;
1071                 }

1073                 sdc = (sctp_data_hdr_t *)mp->b_rptr;
1074                 seglen = ntohs(sdc->sdh_len);
1075                 xtralen = sizeof (*sdc);
1076                 chunklen = seglen - xtralen;

1078                 /*
```

```
1079                 * Check rwnd.
1080                 */
1081                if (chunklen > cansend) {
1082                        head = NULL;
1083                        fp = SCTP_CHUNK_DEST(meta);
1084                        if (fp == NULL || fp->sf_state != SCTP_FADDRS_ALIVE)
1085                                fp = sctp->sctp_current;
1086                        goto unsent_data;
1087                }
1088                if ((extra = seglen & (SCTP_ALIGN - 1)) != 0)
1089                        extra = SCTP_ALIGN - extra;

1091                /*
1092                 * Pick destination address, and check cwnd.
1093                 */
1094                if (sacklen > 0 && (seglen + extra <= lfp->sf_cwnd -
1095                    lfp->sf_suna) &&
1096                    (seglen + sacklen + extra <= lfp->sf_pmss)) {
1097                        /*
1098                         * Only include SACK chunk if it can be bundled
1099                         * with a data chunk, and sent to sctp_lastdata.
1100                         */
1101                        pathmax = lfp->sf_cwnd - lfp->sf_suna;

1103                        fp = lfp;
1104                        if ((nmp = dupmsg(mp)) == NULL) {
1105                                head = NULL;
1106                                goto unsent_data;
1107                        }
1108                        SCTP_CHUNK_CLEAR_FLAGS(nmp);
1109                        head = sctp_add_proto_hdr(sctp, fp, nmp, sacklen,
1110                            &error);
1111                        if (head == NULL) {
1112                                /*
1113                                 * If none of the source addresses are
1114                                 * available (i.e error == EHOSTUNREACH),
1115                                 * pretend we have sent the data. We will
1116                                 * eventually time out trying to retramsmit
1117                                 * the data if the interface never comes up.
1118                                 * If we have already sent some stuff (i.e.,
1119                                 * notsent is B_FALSE) then we are fine, else
1120                                 * just mark this packet as sent.
1121                                 */
1122                                if (notsent && error == EHOSTUNREACH) {
1123                                        SCTP_CHUNK_SENT(sctp, mp, sdc,
1124                                            fp, chunklen, meta);
1125                                }
1126                                freemsg(nmp);
1127                                SCTP_KSTAT(sctps, sctp_output_failed);
1128                                goto unsent_data;
1129                        }
1130                        seglen += sacklen;
1131                        xtralen += sacklen;
1132                        sacklen = 0;
1133                } else {
1134                        fp = SCTP_CHUNK_DEST(meta);
1135                        if (fp == NULL || fp->sf_state != SCTP_FADDRS_ALIVE)
1136                                fp = sctp->sctp_current;
1137                        /*
1138                         * If we haven't sent data to this destination for
1139                         * a while, do slow start again.
1140                         */
1141                        if (now - fp->sf_lastactive > fp->sf_rto) {
1142                                SET_CWND(fp, fp->sf_pmss,
1143                                    sctps->sctps_slow_start_after_idle);
1144                        }
```



```
1146                        pathmax = fp->sf_cwnd - fp->sf_suna;
1147                        if (seglen + extra > pathmax) {
1148                                head = NULL;
1149                                goto unsent_data;
1150                        }
1151                        if ((nmp = dupmsg(mp)) == NULL) {
1152                                head = NULL;
1153                                goto unsent_data;
1154                        }
1155                        SCTP_CHUNK_CLEAR_FLAGS(nmp);
1156                        head = sctp_add_proto_hdr(sctp, fp, nmp, 0, &error);
1157                        if (head == NULL) {
1158                                /*
1159                                 * If none of the source addresses are
1160                                 * available (i.e error == EHOSTUNREACH),
1161                                 * pretend we have sent the data. We will
1162                                 * eventually time out trying to retramsmit
1163                                 * the data if the interface never comes up.
1164                                 * If we have already sent some stuff (i.e.,
1165                                 * notsent is B_FALSE) then we are fine, else
1166                                 * just mark this packet as sent.
1167                                 */
1168                                if (notsent && error == EHOSTUNREACH) {
1169                                        SCTP_CHUNK_SENT(sctp, mp, sdc,
1170                                            fp, chunklen, meta);
1171                                }
1172                                freemsg(nmp);
1173                                SCTP_KSTAT(sctps, sctp_output_failed);
1174                                goto unsent_data;
1175                        }
1176                }
1177                fp->sf_lastactive = now;
1178                if (pathmax > fp->sf_pmss)
1179                        pathmax = fp->sf_pmss;
1180                SCTP_CHUNK_SENT(sctp, mp, sdc, fp, chunklen, meta);
1181                mp = mp->b_next;

1183                /*
1184                 * Use this chunk to measure RTT?
1185                 * Must not be a retransmision of an earlier chunk,
1186                 * ensure the tsn is current.
1187                 */
1188                tsn = ntohl(sdc->sdh_tsn);
1189                if (sctp->sctp_out_time == 0 && tsn == (sctp->sctp_ltsn - 1)) {
1190                        sctp->sctp_out_time = now;
1191                        sctp->sctp_rtt_tsn = tsn;
1192                }
1193                if (extra > 0) {
1194                        fill = sctp_get_padding(sctp, extra);
1195                        if (fill != NULL) {
1196                                linkb(head, fill);
1197                                pad = extra;
1198                                seglen += extra;
1199                        } else {
1200                                goto unsent_data;
1201                        }
1202                }
1203                /*
1204                 * Bundle chunks. We linkb() the chunks together to send
1205                 * downstream in a single packet.
1206                 * Partial chunks MUST NOT be bundled with full chunks, so we
1207                 * rely on sctp_get_msg_to_send() to only return messages that
1208                 * will fit entirely in the current packet.
1209                 */
1210                while (seglen < pathmax) {
```

```
1211                        int32_t         new_len;
1212                        int32_t         new_xtralen;

1214                        while (mp != NULL) {
1215                                if (SCTP_CHUNK_CANSEND(mp))
1216                                        break;
1217                                mp = mp->b_next;
1218                        }
1219                        if (mp == NULL) {
1220                                meta = sctp_get_msg_to_send(sctp, &mp,
1221                                    meta->b_next, &error, seglen,
1222                                    (seglen - xtralen) >= cansend ? 0 :
1223                                    cansend - seglen, fp);
1224                                if (error != 0)
1225                                        break;
1226                                /* If no more eligible chunks, cease bundling */
1227                                if (meta == NULL)
1228                                        break;
1229                                sctp->sctp_xmit_tail =  meta;
1230                        }
1231                        ASSERT(mp != NULL);
1232                        if (!SCTP_CHUNK_ISSENT(mp) && SCTP_CHUNK_DEST(meta) &&
1233                            fp != SCTP_CHUNK_DEST(meta)) {
1234                                break;
1235                        }
1236                        sdc = (sctp_data_hdr_t *)mp->b_rptr;
1237                        chunklen = ntohs(sdc->sdh_len);
1238                        if ((extra = chunklen  & (SCTP_ALIGN - 1)) != 0)
1239                                extra = SCTP_ALIGN - extra;

1241                        new_len = seglen + chunklen;
1242                        new_xtralen = xtralen + sizeof (*sdc);
1243                        chunklen -= sizeof (*sdc);

1245                        if (new_len - new_xtralen > cansend ||
1246                            new_len + extra > pathmax) {
1247                                break;
1248                        }
1249                        if ((nmp = dupmsg(mp)) == NULL)
1250                                break;
1251                        if (extra > 0) {
1252                                fill = sctp_get_padding(sctp, extra);
1253                                if (fill != NULL) {
1254                                        pad += extra;
1255                                        new_len += extra;
1256                                        linkb(nmp, fill);
1257                                } else {
1258                                        freemsg(nmp);
1259                                        break;
1260                                }
1261                        }
1262                        seglen = new_len;
1263                        xtralen = new_xtralen;
1264                        SCTP_CHUNK_CLEAR_FLAGS(nmp);
1265                        SCTP_CHUNK_SENT(sctp, mp, sdc, fp, chunklen, meta);
1266                        linkb(head, nmp);
1267                        mp = mp->b_next;
1268                }
1269                if ((seglen > fp->sf_pmss) && fp->sf_isv4) {
1270                        ipha_t *iph = (ipha_t *)head->b_rptr;

1272                        /*
1273                         * Path MTU is different from what we thought it would
1274                         * be when we created chunks, or IP headers have grown.
1275                         * Need to clear the DF bit.
1276                         */
```



```
1277                                iph->ipha_fragment_offset_and_flags = 0;
1278                        }
1279                        /* xmit segment */
1280                        ASSERT(cansend >= seglen - pad - xtralen);
1281                        cansend -= (seglen - pad - xtralen);
1282                        dprint(2, ("sctp_output: Sending packet %d bytes, tsn %x "
1283                            "ssn %d to %p (rwnd %d, cansend %d, lastack_rxd %x)\n",
1284                            seglen - xtralen, ntohl(sdc->sdh_tsn),
1285                            ntohs(sdc->sdh_ssn), (void *)fp, sctp->sctp_frwnd,
1286                            cansend, sctp->sctp_lastack_rxd));
1287                        sctp_set_iplen(sctp, head, fp->sf_ixa);

1289                        DTRACE_SCTP5(send, mblk_t *, NULL, ip_xmit_attr_t *, fp->sf_ixa,
1290                            void_ip_t *, head->b_rptr, sctp_t *, sctp, sctp_hdr_t *,
1291                            &head->b_rptr[fp->sf_ixa->ixa_ip_hdr_length]);

1293 #endif /* ! codereview */
1294                        (void) conn_ip_output(head, fp->sf_ixa);
1295                        BUMP_LOCAL(sctp->sctp_opkts);
1296                        /* arm rto timer (if not set) */
1297                        if (!fp->sf_timer_running)
1298                                SCTP_FADDR_TIMER_RESTART(sctp, fp, fp->sf_rto);
1299                        notsent = B_FALSE;
1300                }
1301                sctp->sctp_active = now;
1302                return;
1303        unsent_data:
1304                /* arm persist timer (if rto timer not set) */
1305                if (!fp->sf_timer_running)
1306                        SCTP_FADDR_TIMER_RESTART(sctp, fp, fp->sf_rto);
1307                if (head != NULL)
1308                        freemsg(head);
1309        }

1311        /*
1312         * The following two functions initialize and destroy the cache
1313         * associated with the sets used for PR-SCTP.
1314         */
1315        void
1316        sctp_ftsn_sets_init(void)
1317        {
1318                sctp_kmem_ftsn_set_cache = kmem_cache_create("sctp_ftsn_set_cache",
1319                    sizeof (sctp_ftsn_set_t), 0, NULL, NULL, NULL, NULL,
1320                    NULL, 0);
1321        }

1323        void
1324        sctp_ftsn_sets_fini(void)
1325        {
1326                kmem_cache_destroy(sctp_kmem_ftsn_set_cache);
1327        }


1330        /* Free PR-SCTP sets */
1331        void
1332        sctp_free_ftsn_set(sctp_ftsn_set_t *s)
1333        {
1334                sctp_ftsn_set_t *p;

1336                while (s != NULL) {
1337                        p = s->next;
1338                        s->next = NULL;
1339                        kmem_cache_free(sctp_kmem_ftsn_set_cache, s);
1340                        s = p;
1341                }
1342        }
```

```
1344 /*
1345  * Given a message meta block, meta, this routine creates or modifies
1346  * the set that will be used to generate a Forward TSN chunk. If the
1347  * entry for stream id, sid, for this message already exists, the
1348  * sequence number, ssn, is updated if it is greater than the existing
1349  * one. If an entry for this sid does not exist, one is created if
1350  * the size does not exceed fp->sf_pmss. We return false in case
1351  * or an error.
1352  */
1353 boolean_t
1354 sctp_add_ftsn_set(sctp_ftsn_set_t **s, sctp_faddr_t *fp, mblk_t *meta,
1355     uint_t *nsets, uint32_t *slen)
1356 {
1357         sctp_ftsn_set_t         *p;
1358         sctp_msg_hdr_t          *msg_hdr = (sctp_msg_hdr_t *)meta->b_rptr;
1359         uint16_t                sid = htons(msg_hdr->smh_sid);
1360         /* msg_hdr->smh_ssn is already in NBO */
1361         uint16_t                ssn = msg_hdr->smh_ssn;

1363         ASSERT(s != NULL && nsets != NULL);
1364         ASSERT((*nsets == 0 && *s == NULL) || (*nsets > 0 && *s != NULL));

1366         if (*s == NULL) {
1367                 ASSERT((*slen + sizeof (uint32_t)) <= fp->sf_pmss);
1368                 *s = kmem_cache_alloc(sctp_kmem_ftsn_set_cache, KM_NOSLEEP);
1369                 if (*s == NULL)
1370                         return (B_FALSE);
1371                 (*s)->ftsn_entries.ftsn_sid = sid;
1372                 (*s)->ftsn_entries.ftsn_ssn = ssn;
1373                 (*s)->next = NULL;
1374                 *nsets = 1;
1375                 *slen += sizeof (uint32_t);
1376                 return (B_TRUE);
1377         }
1378         for (p = *s; p->next != NULL; p = p->next) {
1379                 if (p->ftsn_entries.ftsn_sid == sid) {
1380                         if (SSN_GT(ssn, p->ftsn_entries.ftsn_ssn))
1381                                 p->ftsn_entries.ftsn_ssn = ssn;
1382                         return (B_TRUE);
1383                 }
1384         }
1385         /* the last one */
1386         if (p->ftsn_entries.ftsn_sid == sid) {
1387                 if (SSN_GT(ssn, p->ftsn_entries.ftsn_ssn))
1388                         p->ftsn_entries.ftsn_ssn = ssn;
1389         } else {
1390                 if ((*slen + sizeof (uint32_t)) > fp->sf_pmss)
1391                         return (B_FALSE);
1392                 p->next = kmem_cache_alloc(sctp_kmem_ftsn_set_cache,
1393                     KM_NOSLEEP);
1394                 if (p->next == NULL)
1395                         return (B_FALSE);
1396                 p = p->next;
1397                 p->ftsn_entries.ftsn_sid = sid;
1398                 p->ftsn_entries.ftsn_ssn = ssn;
1399                 p->next = NULL;
1400                 (*nsets)++;
1401                 *slen += sizeof (uint32_t);
1402         }
1403         return (B_TRUE);
1404 }

1406 /*
1407  * Given a set of stream id - sequence number pairs, this routing creates
1408  * a Forward TSN chunk. The cumulative TSN (advanced peer ack point)
```



```
1409  * for the chunk is obtained from sctp->sctp_adv_pap. The caller
1410  * will add the IP/SCTP header.
1411  */
1412 mblk_t *
1413 sctp_make_ftsn_chunk(sctp_t *sctp, sctp_faddr_t *fp, sctp_ftsn_set_t *sets,
1414     uint_t nsets, uint32_t seglen)
1415 {
1416         mblk_t                  *ftsn_mp;
1417         sctp_chunk_hdr_t        *ch_hdr;
1418         uint32_t                *advtsn;
1419         uint16_t                schlen;
1420         size_t                  xtralen;
1421         ftsn_entry_t            *ftsn_entry;
1422         sctp_stack_t    *sctps = sctp->sctp_sctps;

1424         seglen += sizeof (sctp_chunk_hdr_t);
1425         if (fp->sf_isv4)
1426                 xtralen = sctp->sctp_hdr_len + sctps->sctps_wroff_xtra;
1427         else
1428                 xtralen = sctp->sctp_hdr6_len + sctps->sctps_wroff_xtra;
1429         ftsn_mp = allocb(xtralen + seglen, BPRI_MED);
1430         if (ftsn_mp == NULL)
1431                 return (NULL);
1432         ftsn_mp->b_rptr += xtralen;
1433         ftsn_mp->b_wptr = ftsn_mp->b_rptr + seglen;

1435         ch_hdr = (sctp_chunk_hdr_t *)ftsn_mp->b_rptr;
1436         ch_hdr->sch_id = CHUNK_FORWARD_TSN;
1437         ch_hdr->sch_flags = 0;
1438         /*
1439          * The cast here should not be an issue since seglen is
1440          * the length of the Forward TSN chunk.
1441          */
1442         schlen = (uint16_t)seglen;
1443         U16_TO_ABE16(schlen, &(ch_hdr->sch_len));

1445         advtsn = (uint32_t *)(ch_hdr + 1);
1446         U32_TO_ABE32(sctp->sctp_adv_pap, advtsn);
1447         ftsn_entry = (ftsn_entry_t *)(advtsn + 1);
1448         while (nsets > 0) {
1449                 ASSERT((uchar_t *)&ftsn_entry[1] <= ftsn_mp->b_wptr);
1450                 ftsn_entry->ftsn_sid = sets->ftsn_entries.ftsn_sid;
1451                 ftsn_entry->ftsn_ssn = sets->ftsn_entries.ftsn_ssn;
1452                 ftsn_entry++;
1453                 sets = sets->next;
1454                 nsets--;
1455         }
1456         return (ftsn_mp);
1457 }

1459 /*
1460  * Given a starting message, the routine steps through all the
1461  * messages whose TSN is less than sctp->sctp_adv_pap and creates
1462  * ftsn sets. The ftsn sets is then used to create an Forward TSN
1463  * chunk. All the messages, that have chunks that are included in the
1464  * ftsn sets, are flagged abandonded. If a message is partially sent
1465  * and is deemed abandoned, all remaining unsent chunks are marked
1466  * abandoned and are deducted from sctp_unsent.
1467  */
1468 void
1469 sctp_make_ftsns(sctp_t *sctp, mblk_t *meta, mblk_t *mp, mblk_t **nmp,
1470     sctp_faddr_t *fp, uint32_t *seglen)
1471 {
1472         mblk_t          *mp1 = mp;
1473         mblk_t          *mp_head = mp;
1474         mblk_t          *meta_head = meta;
```

```
1475         mblk_t          *head;
1476         sctp_ftsn_set_t *sets = NULL;
1477         uint_t          nsets = 0;
1478         uint16_t        clen;
1479         sctp_data_hdr_t *sdc;
1480         uint32_t        sacklen;
1481         uint32_t        adv_pap = sctp->sctp_adv_pap;
1482         uint32_t        unsent = 0;
1483         boolean_t       ubit;
1484         sctp_stack_t    *sctps = sctp->sctp_sctps;

1486         *seglen = sizeof (uint32_t);

1488         sdc  = (sctp_data_hdr_t *)mp1->b_rptr;
1489         while (meta != NULL &&
1490             SEQ_GEQ(sctp->sctp_adv_pap, ntohl(sdc->sdh_tsn))) {
1491                 /*
1492                  * Skip adding FTSN sets for un-ordered messages as they do
1493                  * not have SSNs.
1494                  */
1495                 ubit = SCTP_DATA_GET_UBIT(sdc);
1496                 if (!ubit &&
1497                     !sctp_add_ftsn_set(&sets, fp, meta, &nsets, seglen)) {
1498                         meta = NULL;
1499                         sctp->sctp_adv_pap = adv_pap;
1500                         goto ftsn_done;
1501                 }
1502                 while (mp1 != NULL && SCTP_CHUNK_ISSENT(mp1)) {
1503                         sdc = (sctp_data_hdr_t *)mp1->b_rptr;
1504                         adv_pap = ntohl(sdc->sdh_tsn);
1505                         mp1 = mp1->b_next;
1506                 }
1507                 meta = meta->b_next;
1508                 if (meta != NULL) {
1509                         mp1 = meta->b_cont;
1510                         if (!SCTP_CHUNK_ISSENT(mp1))
1511                                 break;
1512                         sdc  = (sctp_data_hdr_t *)mp1->b_rptr;
1513                 }
1514         }
1515 ftsn_done:
1516         /*
1517          * Can't compare with sets == NULL, since we don't add any
1518          * sets for un-ordered messages.
1519          */
1520         if (meta == meta_head)
1521                 return;
1522         *nmp = sctp_make_ftsn_chunk(sctp, fp, sets, nsets, *seglen);
1523         sctp_free_ftsn_set(sets);
1524         if (*nmp == NULL)
1525                 return;
1526         if (sctp->sctp_ftsn == sctp->sctp_lastacked + 1) {
1527                 sacklen = 0;
1528         } else {
1529                 sacklen = sizeof (sctp_chunk_hdr_t) +
1530                     sizeof (sctp_sack_chunk_t) +
1531                     (sizeof (sctp_sack_frag_t) * sctp->sctp_sack_gaps);
1532                 if (*seglen + sacklen > sctp->sctp_lastdata->sf_pmss) {
1533                         /* piggybacked SACK doesn't fit */
1534                         sacklen = 0;
1535                 } else {
1536                         fp = sctp->sctp_lastdata;
1537                 }
1538         }
1539         head = sctp_add_proto_hdr(sctp, fp, *nmp, sacklen, NULL);
1540         if (head == NULL) {
```



```
1541                 freemsg(*nmp);
1542                 *nmp = NULL;
1543                 SCTP_KSTAT(sctps, sctp_send_ftsn_failed);
1544                 return;
1545         }
1546         *seglen += sacklen;
1547         *nmp = head;

1549         /*
1550          * XXXNeed to optimise this, the reason it is done here is so
1551          * that we don't have to undo in case of failure.
1552          */
1553         mp1 = mp_head;
1554         sdc  = (sctp_data_hdr_t *)mp1->b_rptr;
1555         while (meta_head != NULL &&
1556             SEQ_GEQ(sctp->sctp_adv_pap, ntohl(sdc->sdh_tsn))) {
1557                 if (!SCTP_IS_MSG_ABANDONED(meta_head))
1558                         SCTP_MSG_SET_ABANDONED(meta_head);
1559                 while (mp1 != NULL && SCTP_CHUNK_ISSENT(mp1)) {
1560                         sdc = (sctp_data_hdr_t *)mp1->b_rptr;
1561                         if (!SCTP_CHUNK_ISACKED(mp1)) {
1562                                 clen = ntohs(sdc->sdh_len) - sizeof (*sdc);
1563                                 SCTP_CHUNK_SENT(sctp, mp1, sdc, fp, clen,
1564                                     meta_head);
1565                         }
1566                         mp1 = mp1->b_next;
1567                 }
1568                 while (mp1 != NULL) {
1569                         sdc = (sctp_data_hdr_t *)mp1->b_rptr;
1570                         if (!SCTP_CHUNK_ABANDONED(mp1)) {
1571                                 ASSERT(!SCTP_CHUNK_ISSENT(mp1));
1572                                 unsent += ntohs(sdc->sdh_len) - sizeof (*sdc);
1573                                 SCTP_ABANDON_CHUNK(mp1);
1574                         }
1575                         mp1 = mp1->b_next;
1576                 }
1577                 meta_head = meta_head->b_next;
1578                 if (meta_head != NULL) {
1579                         mp1 = meta_head->b_cont;
1580                         if (!SCTP_CHUNK_ISSENT(mp1))
1581                                 break;
1582                         sdc  = (sctp_data_hdr_t *)mp1->b_rptr;
1583                 }
1584         }
1585         if (unsent > 0) {
1586                 ASSERT(sctp->sctp_unsent >= unsent);
1587                 sctp->sctp_unsent -= unsent;
1588                 /*
1589                  * Update ULP the amount of queued data, which is
1590                  * sent-unack'ed + unsent.
1591                  */
1592                 if (!SCTP_IS_DETACHED(sctp))
1593                         SCTP_TXQ_UPDATE(sctp);
1594         }
1595 }

1597 /*
1598  * This function steps through messages starting at meta and checks if
1599  * the message is abandoned. It stops when it hits an unsent chunk or
1600  * a message that has all its chunk acked. This is the only place
1601  * where the sctp_adv_pap is moved forward to indicated abandoned
1602  * messages.
1603  */
1604 void
1605 sctp_check_adv_ack_pt(sctp_t *sctp, mblk_t *meta, mblk_t *mp)
1606 {
```

```
1607         uint32_t        tsn = sctp->sctp_adv_pap;
1608         sctp_data_hdr_t *sdc;
1609         sctp_msg_hdr_t  *msg_hdr;

1611         ASSERT(mp != NULL);
1612         sdc = (sctp_data_hdr_t *)mp->b_rptr;
1613         ASSERT(SEQ_GT(ntohl(sdc->sdh_tsn), sctp->sctp_lastack_rxd));
1614         msg_hdr = (sctp_msg_hdr_t *)meta->b_rptr;
1615         if (!SCTP_IS_MSG_ABANDONED(meta) &&
1616             !SCTP_MSG_TO_BE_ABANDONED(meta, msg_hdr, sctp)) {
1617                 return;
1618         }
1619         while (meta != NULL) {
1620                 while (mp != NULL && SCTP_CHUNK_ISSENT(mp)) {
1621                         sdc = (sctp_data_hdr_t *)mp->b_rptr;
1622                         tsn = ntohl(sdc->sdh_tsn);
1623                         mp = mp->b_next;
1624                 }
1625                 if (mp != NULL)
1626                         break;
1627                 /*
1628                  * We continue checking for successive messages only if there
1629                  * is a chunk marked for retransmission. Else, we might
1630                  * end up sending FTSN prematurely for chunks that have been
1631                  * sent, but not yet acked.
1632                  */
1633                 if ((meta = meta->b_next) != NULL) {
1634                         msg_hdr = (sctp_msg_hdr_t *)meta->b_rptr;
1635                         if (!SCTP_IS_MSG_ABANDONED(meta) &&
1636                             !SCTP_MSG_TO_BE_ABANDONED(meta, msg_hdr, sctp)) {
1637                                 break;
1638                         }
1639                         for (mp = meta->b_cont; mp != NULL; mp = mp->b_next) {
1640                                 if (!SCTP_CHUNK_ISSENT(mp)) {
1641                                         sctp->sctp_adv_pap = tsn;
1642                                         return;
1643                                 }
1644                                 if (SCTP_CHUNK_WANT_REXMIT(mp))
1645                                         break;
1646                         }
1647                         if (mp == NULL)
1648                                 break;
1649                 }
1650         }
1651         sctp->sctp_adv_pap = tsn;
1652 }


1655 /*
1656  * Determine if we should bundle a data chunk with the chunk being
1657  * retransmitted.  We bundle if
1658  *
1659  * - the chunk is sent to the same destination and unack'ed.
1660  *
1661  * OR
1662  *
1663  * - the chunk is unsent, i.e. new data.
1664  */
1665 #define SCTP_CHUNK_RX_CANBUNDLE(mp, fp)                                 \
1666         (!SCTP_CHUNK_ABANDONED((mp)) &&                                 \
1667         ((SCTP_CHUNK_ISSENT((mp)) && (SCTP_CHUNK_DEST(mp) == (fp) &&    \
1668         !SCTP_CHUNK_ISACKED(mp))) ||                                    \
1669         (((mp)->b_flag & (SCTP_CHUNK_FLAG_REXMIT|SCTP_CHUNK_FLAG_SENT)) != \
1670         SCTP_CHUNK_FLAG_SENT)))

1672 /*
```



```
1673  * Retransmit first segment which hasn't been acked with cumtsn or send
1674  * a Forward TSN chunk, if appropriate.
1675  */
1676 void
1677 sctp_rexmit(sctp_t *sctp, sctp_faddr_t *oldfp)
1678 {
1679         mblk_t          *mp;
1680         mblk_t          *nmp = NULL;
1681         mblk_t          *head;
1682         mblk_t          *meta = sctp->sctp_xmit_head;
1683         mblk_t          *fill;
1684         uint32_t        seglen = 0;
1685         uint32_t        sacklen;
1686         uint16_t        chunklen;
1687         int             extra;
1688         sctp_data_hdr_t *sdc;
1689         sctp_faddr_t    *fp;
1690         uint32_t        adv_pap = sctp->sctp_adv_pap;
1691         boolean_t       do_ftsn = B_FALSE;
1692         boolean_t       ftsn_check = B_TRUE;
1693         uint32_t        first_ua_tsn;
1694         sctp_msg_hdr_t  *mhdr;
1695         sctp_stack_t    *sctps = sctp->sctp_sctps;
1696         int             error;

1698         while (meta != NULL) {
1699                 for (mp = meta->b_cont; mp != NULL; mp = mp->b_next) {
1700                         uint32_t        tsn;

1702                         if (!SCTP_CHUNK_ISSENT(mp))
1703                                 goto window_probe;
1704                         /*
1705                          * We break in the following cases -
1706                          *
1707                          *      if the advanced peer ack point includes the next
1708                          *      chunk to be retransmited - possibly the Forward
1709                          *      TSN was lost.
1710                          *
1711                          *      if we are PRSCTP aware and the next chunk to be
1712                          *      retransmitted is now abandoned
1713                          *
1714                          *      if the next chunk to be retransmitted is for
1715                          *      the dest on which the timer went off. (this
1716                          *      message is not abandoned).
1717                          *
1718                          * We check for Forward TSN only for the first
1719                          * eligible chunk to be retransmitted. The reason
1720                          * being if the first eligible chunk is skipped (say
1721                          * it was sent to a destination other than oldfp)
1722                          * then we cannot advance the cum TSN via Forward
1723                          * TSN chunk.
1724                          *
1725                          * Also, ftsn_check is B_TRUE only for the first
1726                          * eligible chunk, it  will be B_FALSE for all
1727                          * subsequent candidate messages for retransmission.
1728                          */
1729                         sdc = (sctp_data_hdr_t *)mp->b_rptr;
1730                         tsn = ntohl(sdc->sdh_tsn);
1731                         if (SEQ_GT(tsn, sctp->sctp_lastack_rxd)) {
1732                                 if (sctp->sctp_prsctp_aware && ftsn_check) {
1733                                         if (SEQ_GEQ(sctp->sctp_adv_pap, tsn)) {
1734                                                 ASSERT(sctp->sctp_prsctp_aware);
1735                                                 do_ftsn = B_TRUE;
1736                                                 goto out;
1737                                         } else {
1738                                                 sctp_check_adv_ack_pt(sctp,
```

```
1739                                                meta, mp);
1740                                        if (SEQ_GT(sctp->sctp_adv_pap,
1741                                            adv_pap)) {
1742                                                do_ftsn = B_TRUE;
1743                                                goto out;
1744                                        }
1745                                }
1746                                ftsn_check = B_FALSE;
1747                        }
1748                        if (SCTP_CHUNK_DEST(mp) == oldfp)
1749                                goto out;
1750                }
1751        }
1752        meta = meta->b_next;
1753        if (meta != NULL && sctp->sctp_prsctp_aware) {
1754                mhdr = (sctp_msg_hdr_t *)meta->b_rptr;

1756                while (meta != NULL && (SCTP_IS_MSG_ABANDONED(meta) ||
1757                    SCTP_MSG_TO_BE_ABANDONED(meta, mhdr, sctp))) {
1758                        meta = meta->b_next;
1759                }
1760        }
1761 }
1762 window_probe:
1763         /*
1764          * Retransmit fired for a destination which didn't have
1765          * any unacked data pending.
1766          */
1767         if (sctp->sctp_unacked == 0 && sctp->sctp_unsent != 0) {
1768                 /*
1769                  * Send a window probe. Inflate frwnd to allow
1770                  * sending one segment.
1771                  */
1772                 if (sctp->sctp_frwnd < (oldfp->sf_pmss - sizeof (*sdc)))
1773                         sctp->sctp_frwnd = oldfp->sf_pmss - sizeof (*sdc);

1775                 /* next TSN to send */
1776                 sctp->sctp_rxt_nxttsn = sctp->sctp_ltsn;

1778                 /*
1779                  * The above sctp_frwnd adjustment is coarse.  The "changed"
1780                  * sctp_frwnd may allow us to send more than 1 packet.  So
1781                  * tell sctp_output() to send only 1 packet.
1782                  */
1783                 sctp_output(sctp, 1);

1785                 /* Last sent TSN */
1786                 sctp->sctp_rxt_maxtsn = sctp->sctp_ltsn - 1;
1787                 ASSERT(sctp->sctp_rxt_maxtsn >= sctp->sctp_rxt_nxttsn);
1788                 sctp->sctp_zero_win_probe = B_TRUE;
1789                 SCTPS_BUMP_MIB(sctps, sctpOutWinProbe);
1790         }
1791         return;
1792 out:
1793         /*
1794          * After a time out, assume that everything has left the network.  So
1795          * we can clear rxt_unacked for the original peer address.
1796          */
1797         oldfp->sf_rxt_unacked = 0;

1799         /*
1800          * If we were probing for zero window, don't adjust retransmission
1801          * variables, but the timer is still backed off.
1802          */
1803         if (sctp->sctp_zero_win_probe) {
1804                 mblk_t  *pkt;
```



```
1805                 uint_t  pkt_len;

1807                 /*
1808                  * Get the Zero Win Probe for retrasmission, sctp_rxt_nxttsn
1809                  * and sctp_rxt_maxtsn will specify the ZWP packet.
1810                  */
1811                 fp = oldfp;
1812                 if (oldfp->sf_state != SCTP_FADDRS_ALIVE)
1813                         fp = sctp_rotate_faddr(sctp, oldfp);
1814                 pkt = sctp_rexmit_packet(sctp, &meta, &mp, fp, &pkt_len);
1815                 if (pkt != NULL) {
1816                         ASSERT(pkt_len <= fp->sf_pmss);
1817                         sctp_set_iplen(sctp, pkt, fp->sf_ixa);

1819                         DTRACE_SCTP5(send, mblk_t *, NULL,
1820                             ip_xmit_attr_t *, fp->sf_ixa,
1821                             void_ip_t *, mp->b_rptr, sctp_t *, sctp, sctp_hdr_t
1822                             &mp->b_rptr[fp->sf_ixa->ixa_ip_hdr_length]);

1824 #endif /* ! codereview */
1825                         (void) conn_ip_output(pkt, fp->sf_ixa);
1826                         BUMP_LOCAL(sctp->sctp_opkts);
1827                 } else {
1828                         SCTP_KSTAT(sctps, sctp_ss_rexmit_failed);
1829                 }

1831                 /*
1832                  * The strikes will be clear by sctp_faddr_alive() when the
1833                  * other side sends us an ack.
1834                  */
1835                 oldfp->sf_strikes++;
1836                 sctp->sctp_strikes++;

1838                 SCTP_CALC_RXT(sctp, oldfp, sctp->sctp_rto_max);
1839                 if (oldfp != fp && oldfp->sf_suna != 0)
1840                         SCTP_FADDR_TIMER_RESTART(sctp, oldfp, fp->sf_rto);
1841                 SCTP_FADDR_TIMER_RESTART(sctp, fp, fp->sf_rto);
1842                 SCTPS_BUMP_MIB(sctps, sctpOutWinProbe);
1843                 return;
1844         }

1846         /*
1847          * Enter slowstart for this destination
1848          */
1849         oldfp->sf_ssthresh = oldfp->sf_cwnd / 2;
1850         if (oldfp->sf_ssthresh < 2 * oldfp->sf_pmss)
1851                 oldfp->sf_ssthresh = 2 * oldfp->sf_pmss;
1852         oldfp->sf_cwnd = oldfp->sf_pmss;
1853         oldfp->sf_pba = 0;
1854         fp = sctp_rotate_faddr(sctp, oldfp);
1855         ASSERT(fp != NULL);
1856         sdc = (sctp_data_hdr_t *)mp->b_rptr;

1858         first_ua_tsn = ntohl(sdc->sdh_tsn);
1859         if (do_ftsn) {
1860                 sctp_make_ftsns(sctp, meta, mp, &nmp, fp, &seglen);
1861                 if (nmp == NULL) {
1862                         sctp->sctp_adv_pap = adv_pap;
1863                         goto restart_timer;
1864                 }
1865                 head = nmp;
1866                 /*
1867                  * Move to the next unabandoned chunk. XXXCheck if meta will
1868                  * always be marked abandoned.
1869                  */
1870                 while (meta != NULL && SCTP_IS_MSG_ABANDONED(meta))
```

```
1871                        meta = meta->b_next;
1872                if (meta != NULL)
1873                        mp = mp->b_cont;
1874                else
1875                        mp = NULL;
1876                goto try_bundle;
1877        }
1878        seglen = ntohs(sdc->sdh_len);
1879        chunklen = seglen - sizeof (*sdc);
1880        if ((extra = seglen & (SCTP_ALIGN - 1)) != 0)
1881                extra = SCTP_ALIGN - extra;

1883        /* Find out if we need to piggyback SACK. */
1884        if (sctp->sctp_ftsn == sctp->sctp_lastacked + 1) {
1885                sacklen = 0;
1886        } else {
1887                sacklen = sizeof (sctp_chunk_hdr_t) +
1888                    sizeof (sctp_sack_chunk_t) +
1889                    (sizeof (sctp_sack_frag_t) * sctp->sctp_sack_gaps);
1890                if (seglen + sacklen > sctp->sctp_lastdata->sf_pmss) {
1891                        /* piggybacked SACK doesn't fit */
1892                        sacklen = 0;
1893                } else {
1894                        /*
1895                         * OK, we have room to send SACK back.  But we
1896                         * should send it back to the last fp where we
1897                         * receive data from, unless sctp_lastdata equals
1898                         * oldfp, then we should probably not send it
1899                         * back to that fp.  Also we should check that
1900                         * the fp is alive.
1901                         */
1902                        if (sctp->sctp_lastdata != oldfp &&
1903                            sctp->sctp_lastdata->sf_state ==
1904                            SCTP_FADDRS_ALIVE) {
1905                                fp = sctp->sctp_lastdata;
1906                        }
1907                }
1908        }

1910        /*
1911         * Cancel RTT measurement if the retransmitted TSN is before the
1912         * TSN used for timimg.
1913         */
1914        if (sctp->sctp_out_time != 0 &&
1915            SEQ_GEQ(sctp->sctp_rtt_tsn, sdc->sdh_tsn)) {
1916                sctp->sctp_out_time = 0;
1917        }
1918        /* Clear the counter as the RTT calculation may be off. */
1919        fp->sf_rtt_updates = 0;
1920        oldfp->sf_rtt_updates = 0;

1922        /*
1923         * After a timeout, we should change the current faddr so that
1924         * new chunks will be sent to the alternate address.
1925         */
1926        sctp_set_faddr_current(sctp, fp);

1928        nmp = dupmsg(mp);
1929        if (nmp == NULL)
1930                goto restart_timer;
1931        if (extra > 0) {
1932                fill = sctp_get_padding(sctp, extra);
1933                if (fill != NULL) {
1934                        linkb(nmp, fill);
1935                        seglen += extra;
1936                } else {
```



```
1937                        freemsg(nmp);
1938                        goto restart_timer;
1939                }
1940        }
1941        SCTP_CHUNK_CLEAR_FLAGS(nmp);
1942        head = sctp_add_proto_hdr(sctp, fp, nmp, sacklen, NULL);
1943        if (head == NULL) {
1944                freemsg(nmp);
1945                SCTP_KSTAT(sctps, sctp_rexmit_failed);
1946                goto restart_timer;
1947        }
1948        seglen += sacklen;

1950        SCTP_CHUNK_SENT(sctp, mp, sdc, fp, chunklen, meta);

1952        mp = mp->b_next;

1954 try_bundle:
1955        /* We can at least and at most send 1 packet at timeout. */
1956        while (seglen < fp->sf_pmss) {
1957                int32_t new_len;

1959                /* Go through the list to find more chunks to be bundled. */
1960                while (mp != NULL) {
1961                        /* Check if the chunk can be bundled. */
1962                        if (SCTP_CHUNK_RX_CANBUNDLE(mp, oldfp))
1963                                break;
1964                        mp = mp->b_next;
1965                }
1966                /* Go to the next message. */
1967                if (mp == NULL) {
1968                        for (meta = meta->b_next; meta != NULL;
1969                            meta = meta->b_next) {
1970                                mhdr = (sctp_msg_hdr_t *)meta->b_rptr;

1972                                if (SCTP_IS_MSG_ABANDONED(meta) ||
1973                                    SCTP_MSG_TO_BE_ABANDONED(meta, mhdr,
1974                                    sctp)) {
1975                                        continue;
1976                                }

1978                                mp = meta->b_cont;
1979                                goto try_bundle;
1980                        }
1981                        /*
1982                         * Check if there is a new message which potentially
1983                         * could be bundled with this retransmission.
1984                         */
1985                        meta = sctp_get_msg_to_send(sctp, &mp, NULL, &error,
1986                            seglen, fp->sf_pmss - seglen, NULL);
1987                        if (error != 0 || meta == NULL) {
1988                                /* No more chunk to be bundled. */
1989                                break;
1990                        } else {
1991                                goto try_bundle;
1992                        }
1993                }

1995                sdc = (sctp_data_hdr_t *)mp->b_rptr;
1996                new_len = ntohs(sdc->sdh_len);
1997                chunklen = new_len - sizeof (*sdc);

1999                if ((extra = new_len & (SCTP_ALIGN - 1)) != 0)
2000                        extra = SCTP_ALIGN - extra;
2001                if ((new_len = seglen + new_len + extra) > fp->sf_pmss)
2002                        break;
```

```
2003                 if ((nmp = dupmsg(mp)) == NULL)
2004                         break;
2005
2006                 if (extra > 0) {
2007                         fill = sctp_get_padding(sctp, extra);
2008                         if (fill != NULL) {
2009                                 linkb(nmp, fill);
2010                         } else {
2011                                 freemsg(nmp);
2012                                 break;
2013                         }
2014                 }
2015                 linkb(head, nmp);
2016
2017                 SCTP_CHUNK_CLEAR_FLAGS(nmp);
2018                 SCTP_CHUNK_SENT(sctp, mp, sdc, fp, chunklen, meta);
2019
2020                 seglen = new_len;
2021                 mp = mp->b_next;
2022         }
2023 done_bundle:
2024         if ((seglen > fp->sf_pmss) && fp->sf_isv4) {
2025                 ipha_t *iph = (ipha_t *)head->b_rptr;
2026
2027                 /*
2028                  * Path MTU is different from path we thought it would
2029                  * be when we created chunks, or IP headers have grown.
2030                  * Need to clear the DF bit.
2031                  */
2032                 iph->ipha_fragment_offset_and_flags = 0;
2033         }
2034         fp->sf_rxt_unacked += seglen;
2035
2036         dprint(2, ("sctp_rexmit: Sending packet %d bytes, tsn %x "
2037             "ssn %d to %p (rwnd %d, lastack_rxd %x)\n",
2038             seglen, ntohl(sdc->sdh_tsn), ntohs(sdc->sdh_ssn),
2039             (void *)fp, sctp->sctp_frwnd, sctp->sctp_lastack_rxd));
2040
2041         sctp->sctp_rexmitting = B_TRUE;
2042         sctp->sctp_rxt_nxttsn = first_ua_tsn;
2043         sctp->sctp_rxt_maxtsn = sctp->sctp_ltsn - 1;
2044         sctp_set_iplen(sctp, head, fp->sf_ixa);
2045
2046         DTRACE_SCTP5(send, mblk_t *, NULL, ip_xmit_attr_t *, fp->sf_ixa,
2047             void_ip_t *, mp->b_rptr, sctp_t *, sctp, sctp_hdr_t *,
2048             &mp->b_rptr[fp->sf_ixa->ixa_ip_hdr_length]);
2049
2050 #endif /* ! codereview */
2051         (void) conn_ip_output(head, fp->sf_ixa);
2052         BUMP_LOCAL(sctp->sctp_opkts);
2053
2054         /*
2055          * Restart the oldfp timer with exponential backoff and
2056          * the new fp timer for the retransmitted chunks.
2057          */
2058 restart_timer:
2059         oldfp->sf_strikes++;
2060         sctp->sctp_strikes++;
2061         SCTP_CALC_RXT(sctp, oldfp, sctp->sctp_rto_max);
2062         /*
2063          * If there is still some data in the oldfp, restart the
2064          * retransmission timer.  If there is no data, the heartbeat will
2065          * continue to run so it will do its job in checking the reachability
2066          * of the oldfp.
2067          */
2068         if (oldfp != fp && oldfp->sf_suna != 0)
```



```
2069                 SCTP_FADDR_TIMER_RESTART(sctp, oldfp, oldfp->sf_rto);
2070
2071         /*
2072          * Should we restart the timer of the new fp?  If there is
2073          * outstanding data to the new fp, the timer should be
2074          * running already.  So restarting it means that the timer
2075          * will fire later for those outstanding data.  But if
2076          * we don't restart it, the timer will fire too early for the
2077          * just retransmitted chunks to the new fp.  The reason is that we
2078          * don't keep a timestamp on when a chunk is retransmitted.
2079          * So when the timer fires, it will just search for the
2080          * chunk with the earliest TSN sent to new fp.  This probably
2081          * is the chunk we just retransmitted.  So for now, let's
2082          * be conservative and restart the timer of the new fp.
2083          */
2084         SCTP_FADDR_TIMER_RESTART(sctp, fp, fp->sf_rto);
2085
2086         sctp->sctp_active = ddi_get_lbolt64();
2087 }
2088
2089 /*
2090  * This function is called by sctp_ss_rexmit() to create a packet
2091  * to be retransmitted to the given fp.  The given meta and mp
2092  * parameters are respectively the sctp_msg_hdr_t and the mblk of the
2093  * first chunk to be retransmitted.  This is also called when we want
2094  * to retransmit a zero window probe from sctp_rexmit() or when we
2095  * want to retransmit the zero window probe after the window has
2096  * opened from sctp_got_sack().
2097  */
2098 mblk_t *
2099 sctp_rexmit_packet(sctp_t *sctp, mblk_t **meta, mblk_t **mp, sctp_faddr_t *fp,
2100     uint_t *packet_len)
2101 {
2102         uint32_t        seglen = 0;
2103         uint16_t        chunklen;
2104         int             extra;
2105         mblk_t          *nmp;
2106         mblk_t          *head;
2107         mblk_t          *fill;
2108         sctp_data_hdr_t *sdc;
2109         sctp_msg_hdr_t  *mhdr;
2110
2111         sdc = (sctp_data_hdr_t *)(*mp)->b_rptr;
2112         seglen = ntohs(sdc->sdh_len);
2113         chunklen = seglen - sizeof (*sdc);
2114         if ((extra = seglen & (SCTP_ALIGN - 1)) != 0)
2115                 extra = SCTP_ALIGN - extra;
2116
2117         nmp = dupmsg(*mp);
2118         if (nmp == NULL)
2119                 return (NULL);
2120         if (extra > 0) {
2121                 fill = sctp_get_padding(sctp, extra);
2122                 if (fill != NULL) {
2123                         linkb(nmp, fill);
2124                         seglen += extra;
2125                 } else {
2126                         freemsg(nmp);
2127                         return (NULL);
2128                 }
2129         }
2130         SCTP_CHUNK_CLEAR_FLAGS(nmp);
2131         head = sctp_add_proto_hdr(sctp, fp, nmp, 0, NULL);
2132         if (head == NULL) {
2133                 freemsg(nmp);
2134                 return (NULL);
```

```
2135         }
2136         SCTP_CHUNK_SENT(sctp, *mp, sdc, fp, chunklen, *meta);
2137         /*
2138          * Don't update the TSN if we are doing a Zero Win Probe.
2139          */
2140         if (!sctp->sctp_zero_win_probe)
2141                 sctp->sctp_rxt_nxttsn = ntohl(sdc->sdh_tsn);
2142         *mp = (*mp)->b_next;

2144 try_bundle:
2145         while (seglen < fp->sf_pmss) {
2146                 int32_t new_len;

2148                 /*
2149                  * Go through the list to find more chunks to be bundled.
2150                  * We should only retransmit sent by unack'ed chunks.  Since
2151                  * they were sent before, the peer's receive window should
2152                  * be able to receive them.
2153                  */
2154                 while (*mp != NULL) {
2155                         /* Check if the chunk can be bundled. */
2156                         if (SCTP_CHUNK_ISSENT(*mp) && !SCTP_CHUNK_ISACKED(*mp))
2157                                 break;
2158                         *mp = (*mp)->b_next;
2159                 }
2160                 /* Go to the next message. */
2161                 if (*mp == NULL) {
2162                         for (*meta = (*meta)->b_next; *meta != NULL;
2163                             *meta = (*meta)->b_next) {
2164                                 mhdr = (sctp_msg_hdr_t *)(*meta)->b_rptr;

2166                                 if (SCTP_IS_MSG_ABANDONED(*meta) ||
2167                                     SCTP_MSG_TO_BE_ABANDONED(*meta, mhdr,
2168                                     sctp)) {
2169                                         continue;
2170                                 }

2172                                 *mp = (*meta)->b_cont;
2173                                 goto try_bundle;
2174                         }
2175                         /* No more chunk to be bundled. */
2176                         break;
2177                 }

2179                 sdc = (sctp_data_hdr_t *)(*mp)->b_rptr;
2180                 /* Don't bundle chunks beyond sctp_rxt_maxtsn. */
2181                 if (SEQ_GT(ntohl(sdc->sdh_tsn), sctp->sctp_rxt_maxtsn))
2182                         break;
2183                 new_len = ntohs(sdc->sdh_len);
2184                 chunklen = new_len - sizeof (*sdc);

2186                 if ((extra = new_len & (SCTP_ALIGN - 1)) != 0)
2187                         extra = SCTP_ALIGN - extra;
2188                 if ((new_len = seglen + new_len + extra) > fp->sf_pmss)
2189                         break;
2190                 if ((nmp = dupmsg(*mp)) == NULL)
2191                         break;

2193                 if (extra > 0) {
2194                         fill = sctp_get_padding(sctp, extra);
2195                         if (fill != NULL) {
2196                                 linkb(nmp, fill);
2197                         } else {
2198                                 freemsg(nmp);
2199                                 break;
2200                         }
```



```
2201                 }
2202                 linkb(head, nmp);

2204                 SCTP_CHUNK_CLEAR_FLAGS(nmp);
2205                 SCTP_CHUNK_SENT(sctp, *mp, sdc, fp, chunklen, *meta);
2206                 /*
2207                  * Don't update the TSN if we are doing a Zero Win Probe.
2208                  */
2209                 if (!sctp->sctp_zero_win_probe)
2210                         sctp->sctp_rxt_nxttsn = ntohl(sdc->sdh_tsn);

2212                 seglen = new_len;
2213                 *mp = (*mp)->b_next;
2214         }
2215         *packet_len = seglen;
2216         fp->sf_rxt_unacked += seglen;
2217         return (head);
2218 }

2220 /*
2221  * sctp_ss_rexmit() is called when we get a SACK after a timeout which
2222  * advances the cum_tsn but the cum_tsn is still less than what we have sent
2223  * (sctp_rxt_maxtsn) at the time of the timeout.  This SACK is a "partial"
2224  * SACK.  We retransmit unacked chunks without having to wait for another
2225  * timeout.  The rationale is that the SACK should not be "partial" if all the
2226  * lost chunks have been retransmitted.  Since the SACK is "partial,"
2227  * the chunks between the cum_tsn and the sctp_rxt_maxtsn should still
2228  * be missing.  It is better for us to retransmit them now instead
2229  * of waiting for a timeout.
2230  */
2231 void
2232 sctp_ss_rexmit(sctp_t *sctp)
2233 {
2234         mblk_t          *meta;
2235         mblk_t          *mp;
2236         mblk_t          *pkt;
2237         sctp_faddr_t    *fp;
2238         uint_t          pkt_len;
2239         uint32_t        tot_wnd;
2240         sctp_data_hdr_t *sdc;
2241         int             burst;
2242         sctp_stack_t    *sctps = sctp->sctp_sctps;

2244         ASSERT(!sctp->sctp_zero_win_probe);

2246         /*
2247          * If the last cum ack is smaller than what we have just
2248          * retransmitted, simply return.
2249          */
2250         if (SEQ_GEQ(sctp->sctp_lastack_rxd, sctp->sctp_rxt_nxttsn))
2251                 sctp->sctp_rxt_nxttsn = sctp->sctp_lastack_rxd + 1;
2252         else
2253                 return;
2254         ASSERT(SEQ_LEQ(sctp->sctp_rxt_nxttsn, sctp->sctp_rxt_maxtsn));

2256         /*
2257          * After a timer fires, sctp_current should be set to the new
2258          * fp where the retransmitted chunks are sent.
2259          */
2260         fp = sctp->sctp_current;

2262         /*
2263          * Since we are retransmitting, we only need to use cwnd to determine
2264          * how much we can send as we were allowed (by peer's receive window)
2265          * to send those retransmitted chunks previously when they are first
2266          * sent.  If we record how much we have retransmitted but
```

```
2267          * unacknowledged using rxt_unacked, then the amount we can now send
2268          * is equal to cwnd minus rxt_unacked.
2269          *
2270          * The field rxt_unacked is incremented when we retransmit a packet
2271          * and decremented when we got a SACK acknowledging something.  And
2272          * it is reset when the retransmission timer fires as we assume that
2273          * all packets have left the network after a timeout.  If this
2274          * assumption is not true, it means that after a timeout, we can
2275          * get a SACK acknowledging more than rxt_unacked (its value only
2276          * contains what is retransmitted when the timer fires).  So
2277          * rxt_unacked will become very big (it is an unsiged int so going
2278          * negative means that the value is huge).  This is the reason we
2279          * always send at least 1 MSS bytes.
2280          *
2281          * The reason why we do not have an accurate count is that we
2282          * only know how many packets are outstanding (using the TSN numbers).
2283          * But we do not know how many bytes those packets contain.  To
2284          * have an accurate count, we need to walk through the send list.
2285          * As it is not really important to have an accurate count during
2286          * retransmission, we skip this walk to save some time.  This should
2287          * not make the retransmission too aggressive to cause congestion.
2288          */
2289         if (fp->sf_cwnd <= fp->sf_rxt_unacked)
2290                 tot_wnd = fp->sf_pmss;
2291         else
2292                 tot_wnd = fp->sf_cwnd - fp->sf_rxt_unacked;

2294         /* Find the first unack'ed chunk */
2295         for (meta = sctp->sctp_xmit_head; meta != NULL; meta = meta->b_next) {
2296                 sctp_msg_hdr_t  *mhdr = (sctp_msg_hdr_t *)meta->b_rptr;

2298                 if (SCTP_IS_MSG_ABANDONED(meta) ||
2299                     SCTP_MSG_TO_BE_ABANDONED(meta, mhdr, sctp)) {
2300                         continue;
2301                 }

2303                 for (mp = meta->b_cont; mp != NULL; mp = mp->b_next) {
2304                         /* Again, this may not be possible */
2305                         if (!SCTP_CHUNK_ISSENT(mp))
2306                                 return;
2307                         sdc = (sctp_data_hdr_t *)mp->b_rptr;
2308                         if (ntohl(sdc->sdh_tsn) == sctp->sctp_rxt_nxttsn)
2309                                 goto found_msg;
2310                 }
2311         }

2313         /* Everything is abandoned... */
2314         return;

2316 found_msg:
2317         if (!fp->sf_timer_running)
2318                 SCTP_FADDR_TIMER_RESTART(sctp, fp, fp->sf_rto);
2319         pkt = sctp_rexmit_packet(sctp, &meta, &mp, fp, &pkt_len);
2320         if (pkt == NULL) {
2321                 SCTP_KSTAT(sctps, sctp_ss_rexmit_failed);
2322                 return;
2323         }
2324         if ((pkt_len > fp->sf_pmss) && fp->sf_isv4) {
2325                 ipha_t  *iph = (ipha_t *)pkt->b_rptr;

2327                 /*
2328                  * Path MTU is different from path we thought it would
2329                  * be when we created chunks, or IP headers have grown.
2330                  *  Need to clear the DF bit.
2331                  */
2332                 iph->ipha_fragment_offset_and_flags = 0;
```



```
2333         }
2334         sctp_set_iplen(sctp, pkt, fp->sf_ixa);

2336         DTRACE_SCTP5(send, mblk_t *, NULL, ip_xmit_attr_t *, fp->sf_ixa,
2337             void_ip_t *, mp->b_rptr, sctp_t *, sctp, sctp_hdr_t *,
2338             &mp->b_rptr[fp->sf_ixa->ixa_ip_hdr_length]);

2340 #endif /* ! codereview */
2341         (void) conn_ip_output(pkt, fp->sf_ixa);
2342         BUMP_LOCAL(sctp->sctp_opkts);

2344         /* Check and see if there is more chunk to be retransmitted. */
2345         if (tot_wnd <= pkt_len || tot_wnd - pkt_len < fp->sf_pmss ||
2346             meta == NULL)
2347                 return;
2348         if (mp == NULL)
2349                 meta = meta->b_next;
2350         if (meta == NULL)
2351                 return;

2353         /* Retransmit another packet if the window allows. */
2354         for (tot_wnd -= pkt_len, burst = sctps->sctps_maxburst - 1;
2355             meta != NULL && burst > 0; meta = meta->b_next, burst--) {
2356                 if (mp == NULL)
2357                         mp = meta->b_cont;
2358                 for (; mp != NULL; mp = mp->b_next) {
2359                         /* Again, this may not be possible */
2360                         if (!SCTP_CHUNK_ISSENT(mp))
2361                                 return;
2362                         if (!SCTP_CHUNK_ISACKED(mp))
2363                                 goto found_msg;
2364                 }
2365         }
2366 }
```

```
**********************************************************
   17598 Sat Apr 12 11:18:56 2014
new/usr/src/uts/common/sys/sdt.h
3903 DTrace SCTP Provider
**********************************************************
______unchanged_portion_omitted_

 144 #define DTRACE_SCHED(name)                                              \
 145         DTRACE_PROBE(__sched_##name);

 147 #define DTRACE_SCHED1(name, type1, arg1)                                \
 148         DTRACE_PROBE1(__sched_##name, type1, arg1);

 150 #define DTRACE_SCHED2(name, type1, arg1, type2, arg2)                   \
 151         DTRACE_PROBE2(__sched_##name, type1, arg1, type2, arg2);

 153 #define DTRACE_SCHED3(name, type1, arg1, type2, arg2, type3, arg3)      \
 154         DTRACE_PROBE3(__sched_##name, type1, arg1, type2, arg2, type3, arg3);

 156 #define DTRACE_SCHED4(name, type1, arg1, type2, arg2,                   \
 157     type3, arg3, type4, arg4)                                           \
 158         DTRACE_PROBE4(__sched_##name, type1, arg1, type2, arg2,         \
 159             type3, arg3, type4, arg4);

 161 #define DTRACE_PROC(name)                                               \
 162         DTRACE_PROBE(__proc_##name);

 164 #define DTRACE_PROC1(name, type1, arg1)                                 \
 165         DTRACE_PROBE1(__proc_##name, type1, arg1);

 167 #define DTRACE_PROC2(name, type1, arg1, type2, arg2)                    \
 168         DTRACE_PROBE2(__proc_##name, type1, arg1, type2, arg2);

 170 #define DTRACE_PROC3(name, type1, arg1, type2, arg2, type3, arg3)       \
 171         DTRACE_PROBE3(__proc_##name, type1, arg1, type2, arg2, type3, arg3);

 173 #define DTRACE_PROC4(name, type1, arg1, type2, arg2,                    \
 174     type3, arg3, type4, arg4)                                           \
 175         DTRACE_PROBE4(__proc_##name, type1, arg1, type2, arg2,          \
 176             type3, arg3, type4, arg4);

 178 #define DTRACE_IO(name)                                                 \
 179         DTRACE_PROBE(__io_##name);

 181 #define DTRACE_IO1(name, type1, arg1)                                   \
 182         DTRACE_PROBE1(__io_##name, type1, arg1);

 184 #define DTRACE_IO2(name, type1, arg1, type2, arg2)                      \
 185         DTRACE_PROBE2(__io_##name, type1, arg1, type2, arg2);

 187 #define DTRACE_IO3(name, type1, arg1, type2, arg2, type3, arg3) \
 188         DTRACE_PROBE3(__io_##name, type1, arg1, type2, arg2, type3, arg3);

 190 #define DTRACE_IO4(name, type1, arg1, type2, arg2,                      \
 191     type3, arg3, type4, arg4)                                           \
 192         DTRACE_PROBE4(__io_##name, type1, arg1, type2, arg2,            \
 193             type3, arg3, type4, arg4);

 195 #define DTRACE_ISCSI_2(name, type1, arg1, type2, arg2)                  \
 196         DTRACE_PROBE2(__iscsi_##name, type1, arg1, type2, arg2);

 198 #define DTRACE_ISCSI_3(name, type1, arg1, type2, arg2, type3, arg3)     \
 199         DTRACE_PROBE3(__iscsi_##name, type1, arg1, type2, arg2, type3, arg3);

 201 #define DTRACE_ISCSI_4(name, type1, arg1, type2, arg2,                  \
 202     type3, arg3, type4, arg4)                                           \
```



```
 203         DTRACE_PROBE4(__iscsi_##name, type1, arg1, type2, arg2,         \
 204             type3, arg3, type4, arg4);

 206 #define DTRACE_ISCSI_5(name, type1, arg1, type2, arg2,                  \
 207     type3, arg3, type4, arg4, type5, arg5)                              \
 208         DTRACE_PROBE5(__iscsi_##name, type1, arg1, type2, arg2,         \
 209             type3, arg3, type4, arg4, type5, arg5);

 211 #define DTRACE_ISCSI_6(name, type1, arg1, type2, arg2,                  \
 212     type3, arg3, type4, arg4, type5, arg5, type6, arg6)                 \
 213         DTRACE_PROBE6(__iscsi_##name, type1, arg1, type2, arg2,         \
 214             type3, arg3, type4, arg4, type5, arg5, type6, arg6);

 216 #define DTRACE_ISCSI_7(name, type1, arg1, type2, arg2,                  \
 217     type3, arg3, type4, arg4, type5, arg5, type6, arg6, type7, arg7)    \
 218         DTRACE_PROBE7(__iscsi_##name, type1, arg1, type2, arg2,         \
 219             type3, arg3, type4, arg4, type5, arg5, type6, arg6,         \
 220             type7, arg7);

 222 #define DTRACE_ISCSI_8(name, type1, arg1, type2, arg2,                  \
 223     type3, arg3, type4, arg4, type5, arg5, type6, arg6,                 \
 224     type7, arg7, type8, arg8)                                           \
 225         DTRACE_PROBE8(__iscsi_##name, type1, arg1, type2, arg2,         \
 226             type3, arg3, type4, arg4, type5, arg5, type6, arg6,         \
 227             type7, arg7, type8, arg8);

 229 #define DTRACE_NFSV3_3(name, type1, arg1, type2, arg2,                  \
 230     type3, arg3)                                                        \
 231         DTRACE_PROBE3(__nfsv3_##name, type1, arg1, type2, arg2,         \
 232             type3, arg3);
 233 #define DTRACE_NFSV3_4(name, type1, arg1, type2, arg2,                  \
 234     type3, arg3, type4, arg4)                                           \
 235         DTRACE_PROBE4(__nfsv3_##name, type1, arg1, type2, arg2,         \
 236             type3, arg3, type4, arg4);

 238 #define DTRACE_NFSV4_1(name, type1, arg1) \
 239         DTRACE_PROBE1(__nfsv4_##name, type1, arg1);

 241 #define DTRACE_NFSV4_2(name, type1, arg1, type2, arg2) \
 242         DTRACE_PROBE2(__nfsv4_##name, type1, arg1, type2, arg2);

 244 #define DTRACE_NFSV4_3(name, type1, arg1, type2, arg2, type3, arg3) \
 245         DTRACE_PROBE3(__nfsv4_##name, type1, arg1, type2, arg2, type3, arg3);

 247 #define DTRACE_SMB_1(name, type1, arg1) \
 248         DTRACE_PROBE1(__smb_##name, type1, arg1);

 250 #define DTRACE_SMB_2(name, type1, arg1, type2, arg2) \
 251         DTRACE_PROBE2(__smb_##name, type1, arg1, type2, arg2);

 253 #define DTRACE_IP(name)                                                 \
 254         DTRACE_PROBE(__ip_##name);

 256 #define DTRACE_IP1(name, type1, arg1)                                   \
 257         DTRACE_PROBE1(__ip_##name, type1, arg1);

 259 #define DTRACE_IP2(name, type1, arg1, type2, arg2)                      \
 260         DTRACE_PROBE2(__ip_##name, type1, arg1, type2, arg2);

 262 #define DTRACE_IP3(name, type1, arg1, type2, arg2, type3, arg3) \
 263         DTRACE_PROBE3(__ip_##name, type1, arg1, type2, arg2, type3, arg3);

 265 #define DTRACE_IP4(name, type1, arg1, type2, arg2,                      \
 266     type3, arg3, type4, arg4)                                           \
 267         DTRACE_PROBE4(__ip_##name, type1, arg1, type2, arg2,            \
 268             type3, arg3, type4, arg4);
```

```
270 #define DTRACE_IP5(name, type1, arg1, type2, arg2,                      \
271     type3, arg3, type4, arg4, type5, arg5)                              \
272         DTRACE_PROBE5(__ip_##name, type1, arg1, type2, arg2,            \
273             type3, arg3, type4, arg4, type5, arg5);

275 #define DTRACE_IP6(name, type1, arg1, type2, arg2,                      \
276     type3, arg3, type4, arg4, type5, arg5, type6, arg6)                 \
277         DTRACE_PROBE6(__ip_##name, type1, arg1, type2, arg2,            \
278             type3, arg3, type4, arg4, type5, arg5, type6, arg6);

280 #define DTRACE_IP7(name, type1, arg1, type2, arg2, type3, arg3,         \
281     type4, arg4, type5, arg5, type6, arg6, type7, arg7)                 \
282         DTRACE_PROBE7(__ip_##name, type1, arg1, type2, arg2,            \
283             type3, arg3, type4, arg4, type5, arg5, type6, arg6,         \
284             type7, arg7);

286 #define DTRACE_TCP(name)                                                \
287         DTRACE_PROBE(__tcp_##name);

289 #define DTRACE_TCP1(name, type1, arg1)                                  \
290         DTRACE_PROBE1(__tcp_##name, type1, arg1);

292 #define DTRACE_TCP2(name, type1, arg1, type2, arg2)                     \
293         DTRACE_PROBE2(__tcp_##name, type1, arg1, type2, arg2);

295 #define DTRACE_TCP3(name, type1, arg1, type2, arg2, type3, arg3)        \
296         DTRACE_PROBE3(__tcp_##name, type1, arg1, type2, arg2, type3, arg3);

298 #define DTRACE_TCP4(name, type1, arg1, type2, arg2,                     \
299     type3, arg3, type4, arg4)                                           \
300         DTRACE_PROBE4(__tcp_##name, type1, arg1, type2, arg2,           \
301             type3, arg3, type4, arg4);

303 #define DTRACE_TCP5(name, type1, arg1, type2, arg2,                     \
304     type3, arg3, type4, arg4, type5, arg5)                              \
305         DTRACE_PROBE5(__tcp_##name, type1, arg1, type2, arg2,           \
306             type3, arg3, type4, arg4, type5, arg5);

308 #define DTRACE_TCP6(name, type1, arg1, type2, arg2,                     \
309     type3, arg3, type4, arg4, type5, arg5, type6, arg6)                 \
310         DTRACE_PROBE6(__tcp_##name, type1, arg1, type2, arg2,           \
311             type3, arg3, type4, arg4, type5, arg5, type6, arg6);

313 #define DTRACE_UDP(name)                                                \
314         DTRACE_PROBE(__udp_##name);

316 #define DTRACE_UDP1(name, type1, arg1)                                  \
317         DTRACE_PROBE1(__udp_##name, type1, arg1);

319 #define DTRACE_UDP2(name, type1, arg1, type2, arg2)                     \
320         DTRACE_PROBE2(__udp_##name, type1, arg1, type2, arg2);

322 #define DTRACE_UDP3(name, type1, arg1, type2, arg2, type3, arg3)        \
323         DTRACE_PROBE3(__udp_##name, type1, arg1, type2, arg2, type3, arg3);

325 #define DTRACE_UDP4(name, type1, arg1, type2, arg2,                     \
326     type3, arg3, type4, arg4)                                           \
327         DTRACE_PROBE4(__udp_##name, type1, arg1, type2, arg2,           \
328             type3, arg3, type4, arg4);

330 #define DTRACE_UDP5(name, type1, arg1, type2, arg2,                     \
331     type3, arg3, type4, arg4, type5, arg5)                              \
332         DTRACE_PROBE5(__udp_##name, type1, arg1, type2, arg2,           \
333             type3, arg3, type4, arg4, type5, arg5);
```



```
335 #define DTRACE_SCTP(name)                                               \
336         DTRACE_PROBE(__sctp_##name);

338 #define DTRACE_SCTP1(name, type1, arg1)                                 \
339         DTRACE_PROBE1(__sctp_##name, type1, arg1);

341 #define DTRACE_SCTP2(name, type1, arg1, type2, arg2)                    \
342         DTRACE_PROBE2(__sctp_##name, type1, arg1, type2, arg2);

344 #define DTRACE_SCTP3(name, type1, arg1, type2, arg2, type3, arg3)       \
345         DTRACE_PROBE3(__sctp_##name, type1, arg1, type2, arg2, type3, arg3);

347 #define DTRACE_SCTP4(name, type1, arg1, type2, arg2,                    \
348     type3, arg3, type4, arg4)                                           \
349         DTRACE_PROBE4(__sctp_##name, type1, arg1, type2, arg2,          \
350             type3, arg3, type4, arg4);

352 #define DTRACE_SCTP5(name, type1, arg1, type2, arg2,                    \
353     type3, arg3, type4, arg4, type5, arg5)                              \
354         DTRACE_PROBE5(__sctp_##name, type1, arg1, type2, arg2,          \
355             type3, arg3, type4, arg4, type5, arg5);

357 #define DTRACE_SCTP6(name, type1, arg1, type2, arg2,                    \
358     type3, arg3, type4, arg4, type5, arg5, type6, arg6)                 \
359         DTRACE_PROBE6(__sctp_##name, type1, arg1, type2, arg2,          \
360             type3, arg3, type4, arg4, type5, arg5, type6, arg6);
361 #endif /* ! codereview */

363 #define DTRACE_SYSEVENT2(name, type1, arg1, type2, arg2)                \
364         DTRACE_PROBE2(__sysevent_##name, type1, arg1, type2, arg2);

366 #define DTRACE_XPV(name)                                                \
367         DTRACE_PROBE(__xpv_##name);

369 #define DTRACE_XPV1(name, type1, arg1)                                  \
370         DTRACE_PROBE1(__xpv_##name, type1, arg1);

372 #define DTRACE_XPV2(name, type1, arg1, type2, arg2)                     \
373         DTRACE_PROBE2(__xpv_##name, type1, arg1, type2, arg2);

375 #define DTRACE_XPV3(name, type1, arg1, type2, arg2, type3, arg3)        \
376         DTRACE_PROBE3(__xpv_##name, type1, arg1, type2, arg2, type3, arg3);

378 #define DTRACE_XPV4(name, type1, arg1, type2, arg2, type3, arg3,        \
379             type4, arg4)                                                \
380         DTRACE_PROBE4(__xpv_##name, type1, arg1, type2, arg2,           \
381             type3, arg3, type4, arg4);

383 #define DTRACE_FC_1(name, type1, arg1) \
384         DTRACE_PROBE1(__fc_##name, type1, arg1);

386 #define DTRACE_FC_2(name, type1, arg1, type2, arg2) \
387         DTRACE_PROBE2(__fc_##name, type1, arg1, type2, arg2);

389 #define DTRACE_FC_3(name, type1, arg1, type2, arg2, type3, arg3) \
390         DTRACE_PROBE3(__fc_##name, type1, arg1, type2, arg2, type3, arg3);

392 #define DTRACE_FC_4(name, type1, arg1, type2, arg2, type3, arg3, type4, arg4) \
393         DTRACE_PROBE4(__fc_##name, type1, arg1, type2, arg2, type3, arg3, \
394             type4, arg4);

396 #define DTRACE_FC_5(name, type1, arg1, type2, arg2, type3, arg3,        \
397             type4, arg4, type5, arg5)                                   \
398         DTRACE_PROBE5(__fc_##name, type1, arg1, type2, arg2, type3, arg3, \
399             type4, arg4, type5, arg5);
```

```
401 #define DTRACE_SRP_1(name, type1, arg1)                                 \
402         DTRACE_PROBE1(__srp_##name, type1, arg1);

404 #define DTRACE_SRP_2(name, type1, arg1, type2, arg2)                    \
405         DTRACE_PROBE2(__srp_##name, type1, arg1, type2, arg2);

407 #define DTRACE_SRP_3(name, type1, arg1, type2, arg2, type3, arg3)       \
408         DTRACE_PROBE3(__srp_##name, type1, arg1, type2, arg2, type3, arg3);

410 #define DTRACE_SRP_4(name, type1, arg1, type2, arg2, type3, arg3,       \
411             type4, arg4)                                                \
412         DTRACE_PROBE4(__srp_##name, type1, arg1, type2, arg2,           \
413             type3, arg3, type4, arg4);

415 #define DTRACE_SRP_5(name, type1, arg1, type2, arg2, type3, arg3,       \
416             type4, arg4, type5, arg5)                                   \
417         DTRACE_PROBE5(__srp_##name, type1, arg1, type2, arg2,           \
418             type3, arg3, type4, arg4, type5, arg5);

420 #define DTRACE_SRP_6(name, type1, arg1, type2, arg2, type3, arg3,       \
421             type4, arg4, type5, arg5, type6, arg6)                      \
422         DTRACE_PROBE6(__srp_##name, type1, arg1, type2, arg2,           \
423             type3, arg3, type4, arg4, type5, arg5, type6, arg6);

425 #define DTRACE_SRP_7(name, type1, arg1, type2, arg2, type3, arg3,       \
426             type4, arg4, type5, arg5, type6, arg6, type7, arg7)         \
427         DTRACE_PROBE7(__srp_##name, type1, arg1, type2, arg2,           \
428             type3, arg3, type4, arg4, type5, arg5, type6, arg6, type7, arg7);

430 #define DTRACE_SRP_8(name, type1, arg1, type2, arg2, type3, arg3,       \
431             type4, arg4, type5, arg5, type6, arg6, type7, arg7, type8, arg8) \
432         DTRACE_PROBE8(__srp_##name, type1, arg1, type2, arg2,           \
433             type3, arg3, type4, arg4, type5, arg5, type6, arg6,         \
434             type7, arg7, type8, arg8);

436 /*
437  * the set-error SDT probe is extra static, in that we declare its fake
438  * function literally, rather than with the DTRACE_PROBE1() macro.  This is
439  * necessary so that SET_ERROR() can evaluate to a value, which wouldn't
440  * be possible if it required multiple statements (to declare the function
441  * and then call it).
442  *
443  * SET_ERROR() uses the comma operator so that it can be used without much
444  * additional code.  For example, "return (EINVAL);" becomes
445  * "return (SET_ERROR(EINVAL));".  Note that the argument will be evaluated
446  * twice, so it should not have side effects (e.g. something like:
447  * "return (SET_ERROR(log_error(EINVAL, info)));" would log the error twice).
448  */
449 extern void __dtrace_probe_set__error(uintptr_t);
450 #define SET_ERROR(err) (__dtrace_probe_set__error(err), err)

452 #endif /* _KERNEL */

454 extern const char *sdt_prefix;

456 typedef struct sdt_probedesc {
457         char                    *sdpd_name;     /* name of this probe */
458         unsigned long           sdpd_offset;    /* offset of call in text */
459         struct sdt_probedesc    *sdpd_next;     /* next static probe */
460 } sdt_probedesc_t;

462 #ifdef  __cplusplus
463 }
464 #endif

466 #endif  /* _SYS_SDT_H */
```