```
**********************************************************
    2236 Fri Apr 17 12:29:23 2015
new/usr/src/cmd/mdb/common/libstandctf/mapfile
1730 DTrace should ignore type information from modules with cth_parlabel mismat
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
**********************************************************
#
# Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 2012, Joyent, Inc.
#

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

$mapfile_version 2

SYMBOL_SCOPE {
        global:
                ctf_add_array;
                ctf_add_float;
                ctf_add_integer;
                ctf_add_member;
                ctf_add_pointer;
                ctf_add_struct;
                ctf_add_type;
                ctf_add_typedef;
                ctf_add_union;
                ctf_array_info;
                ctf_bufopen;
                ctf_close;
                ctf_create;
                ctf_delete_type;
                ctf_discard;
```



```
                ctf_enum_iter;
                ctf_enum_name;
                ctf_enum_value;
                ctf_errmsg;
                ctf_errno;
                ctf_fdopen;
                ctf_func_args;
                ctf_func_info;
                ctf_getmodel;
                ctf_getspecific;
                ctf_import;
                ctf_label_info;
                ctf_label_iter;
                ctf_label_topmost;
                ctf_lookup_by_name;
                ctf_lookup_by_symbol;
                ctf_member_info;
                ctf_member_iter;
                ctf_open;
                ctf_parent_name;
                ctf_parent_label;
#endif /* ! codereview */
                ctf_setmodel;
                ctf_setspecific;
                ctf_type_align;
                ctf_type_cmp;
                ctf_type_encoding;
                ctf_type_iter;
                ctf_type_lname;
                ctf_type_kind;
                ctf_type_name;
                ctf_type_pointer;
                ctf_type_reference;
                ctf_type_resolve;
                ctf_type_size;
                ctf_type_visit;
                ctf_update;
                ctf_version;
        local:
                *;
};
```

```
**********************************************************
   29279 Fri Apr 17 12:29:24 2015
new/usr/src/common/ctf/ctf_open.c
1730 DTrace should ignore type information from modules with cth_parlabel mismat
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
**********************************************************
______unchanged_portion_omitted_

 982 /*
 983  * Return the label of the parent CTF container, if one exists.  Otherwise
 984  * return NULL.
 985  */
 986 const char *
 987 ctf_parent_label(ctf_file_t *fp)
 988 {
 989         return (fp->ctf_parlabel);
 990 }

 992 /*
 993 #endif /* ! codereview */
 994  * Import the types from the specified parent container by storing a pointer
 995  * to it in ctf_parent and incrementing its reference count.  Only one parent
 996  * is allowed: if a parent already exists, it is replaced by the new parent.
 997  */
 998 int
 999 ctf_import(ctf_file_t *fp, ctf_file_t *pfp)
1000 {
1001         if (fp == NULL || fp == pfp || (pfp != NULL && pfp->ctf_refcnt == 0))
1002                 return (ctf_set_errno(fp, EINVAL));

1004         if (pfp != NULL && pfp->ctf_dmodel != fp->ctf_dmodel)
1005                 return (ctf_set_errno(fp, ECTF_DMODEL));

1007         if (fp->ctf_parent != NULL)
1008                 ctf_close(fp->ctf_parent);

1010         if (pfp != NULL) {
1011                 fp->ctf_flags |= LCTF_CHILD;
1012                 pfp->ctf_refcnt++;
1013         }

1015         fp->ctf_parent = pfp;
1016         return (0);
1017 }

1019 /*
1020  * Set the data model constant for the CTF container.
1021  */
1022 int
1023 ctf_setmodel(ctf_file_t *fp, int model)
1024 {
1025         const ctf_dmodel_t *dp;

1027         for (dp = _libctf_models; dp->ctd_name != NULL; dp++) {
1028                 if (dp->ctd_code == model) {
1029                         fp->ctf_dmodel = dp;
1030                         return (0);
1031                 }
1032         }

1034         return (ctf_set_errno(fp, EINVAL));
1035 }

1037 /*
1038  * Return the data model constant for the CTF container.
```



```
1039  */
1040 int
1041 ctf_getmodel(ctf_file_t *fp)
1042 {
1043         return (fp->ctf_dmodel->ctd_code);
1044 }

1046 void
1047 ctf_setspecific(ctf_file_t *fp, void *data)
1048 {
1049         fp->ctf_specific = data;
1050 }

1052 void *
1053 ctf_getspecific(ctf_file_t *fp)
1054 {
1055         return (fp->ctf_specific);
1056 }
```

```
**********************************************************
    2653 Fri Apr 17 12:29:25 2015
new/usr/src/lib/libctf/common/mapfile-vers
1730 DTrace should ignore type information from modules with cth_parlabel mismat
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
#

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

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

$mapfile_version 2

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

SYMBOL_VERSION SUNWprivate_1.2 {
    global:
        ctf_add_array;
        ctf_add_const;
        ctf_add_enum;
        ctf_add_enumerator;
        ctf_add_float;
        ctf_add_forward;
        ctf_add_function;
        ctf_add_integer;
        ctf_add_member;
        ctf_add_pointer;
```



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

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

```
**********************************************************
   62921 Fri Apr 17 12:29:25 2015
new/usr/src/lib/libdtrace/common/dt_cg.c
libdtrace: attempt to resolve FORWARD types to concrete types
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

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

#include <strings.h>
#include <stdlib.h>
#include <setjmp.h>
#include <assert.h>
#include <errno.h>

#include <dt_impl.h>
#include <dt_grammar.h>
#include <dt_module.h>
#endif /* ! codereview */
#include <dt_parser.h>
#include <dt_provider.h>

static void dt_cg_node(dt_node_t *, dt_irlist_t *, dt_regset_t *);

static dt_irnode_t *
dt_cg_node_alloc(uint_t label, dif_instr_t instr)
{
        dt_irnode_t *dip = malloc(sizeof (dt_irnode_t));

        if (dip == NULL)
                longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

        dip->di_label = label;
        dip->di_instr = instr;
        dip->di_extern = NULL;
```



```
        dip->di_next = NULL;

        return (dip);
}

/*
 * Code generator wrapper function for ctf_member_info.  If we are given a
 * reference to a forward declaration tag, search the entire type space for
 * the actual definition and then call ctf_member_info on the result.
 */
static ctf_file_t *
dt_cg_membinfo(ctf_file_t *fp, ctf_id_t type, const char *s, ctf_membinfo_t *mp)
{
        dt_resolve_forward_decl(&fp, &type);
        while (ctf_type_kind(fp, type) == CTF_K_FORWARD) {
                char n[DT_TYPE_NAMELEN];
                dtrace_typeinfo_t dtt;

                if (ctf_type_name(fp, type, n, sizeof (n)) == NULL ||
                    dt_type_lookup(n, &dtt) == -1 || (
                    dtt.dtt_ctfp == fp && dtt.dtt_type == type))
                        break; /* unable to improve our position */

                fp = dtt.dtt_ctfp;
                type = ctf_type_resolve(fp, dtt.dtt_type);
        }

        if (ctf_member_info(fp, type, s, mp) == CTF_ERR)
                return (NULL); /* ctf_errno is set for us */

        return (fp);
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    8053 Fri Apr 17 12:29:26 2015
new/usr/src/lib/libdtrace/common/dt_error.c
1730 DTrace should ignore type information from modules with cth_parlabel mismat
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
**********************************************************
______unchanged_portion_omitted_
  39         { EDT_VERSION,  "Client requested version newer than library" },
  40         { EDT_VERSINVAL, "Version is not properly formatted or is too large" },
  41         { EDT_VERSUNDEF, "Requested version is not supported by compiler" },
  42         { EDT_VERSREDUCED, "Requested version conflicts with earlier setting" },
  43         { EDT_CTF,      "Unexpected libctf error" },
  44         { EDT_COMPILER, "Error in D program compilation" },
  45         { EDT_NOTUPREG, "Insufficient tuple registers to generate code" },
  46         { EDT_NOMEM,    "Memory allocation failure" },
  47         { EDT_INT2BIG,  "Integer constant table limit exceeded" },
  48         { EDT_STR2BIG,  "String constant table limit exceeded" },
  49         { EDT_NOMOD,    "Unknown module name" },
  50         { EDT_NOPROV,   "Unknown provider name" },
  51         { EDT_NOPROBE,  "No probe matches description" },
  52         { EDT_NOSYM,    "Unknown symbol name" },
  53         { EDT_NOSYMADDR, "No symbol corresponds to address" },
  54         { EDT_NOTYPE,   "Unknown type name" },
  55         { EDT_NOVAR,    "Unknown variable name" },
  56         { EDT_NOAGG,    "Unknown aggregation name" },
  57         { EDT_BADSCOPE, "Improper use of scoping operator in type name" },
  58         { EDT_BADSPEC,  "Overspecified probe description" },
  59         { EDT_BADSPCV,  "Undefined macro variable in probe description" },
  60         { EDT_BADID,    "Unknown probe identifier" },
  61         { EDT_NOTLOADED, "Module is no longer loaded" },
  62         { EDT_NOCTF,    "Module does not contain any CTF data" },
  63         { EDT_DATAMODEL, "Module and program data models do not match" },
  64         { EDT_DIFVERS,  "Library uses newer DIF version than kernel" },
  65         { EDT_BADAGG,   "Unknown aggregating action" },
  66         { EDT_FIO,      "Error occurred while reading from input stream" },
  67         { EDT_DIFINVAL, "DIF program content is invalid" },
  68         { EDT_DIFSIZE,  "DIF program exceeds maximum program size" },
  69         { EDT_DIFFAULT, "DIF program contains invalid pointer" },
  70         { EDT_BADPROBE, "Invalid probe specification" },
  71         { EDT_BADPGLOB, "Probe description has too many globbing characters" },
  72         { EDT_NOSCOPE,  "Declaration scope stack underflow" },
  73         { EDT_NODECL,   "Declaration stack underflow" },
  74         { EDT_DMISMATCH, "Data record list does not match statement" },
  75         { EDT_DOFFSET,  "Data record offset exceeds buffer boundary" },
  76         { EDT_DALIGN,   "Data record has inappropriate alignment" },
  77         { EDT_BADOPTNAME, "Invalid option name" },
  78         { EDT_BADOPTVAL, "Invalid value for specified option" },
  79         { EDT_BADOPTCTX, "Option cannot be used from within a D program" },
  80         { EDT_CPPFORK,  "Failed to fork preprocessor" },
  81         { EDT_CPPEXEC,  "Failed to exec preprocessor" },
  82         { EDT_CPPENT,   "Preprocessor not found" },
  83         { EDT_CPPERR,   "Preprocessor failed to process input program" },
  84         { EDT_SYMOFLOW, "Symbol table identifier space exhausted" },
  85         { EDT_ACTIVE,   "Operation illegal when tracing is active" },
  86         { EDT_DESTRUCTIVE, "Destructive actions not allowed" },
  87         { EDT_NOANON,   "No anonymous tracing state" },
  88         { EDT_ISANON,   "Can't claim anonymous state and enable probes" },
  89         { EDT_ENDTOOBIG, "END enablings exceed size of principal buffer" },
  90         { EDT_NOCONV,   "Failed to load type for printf conversion" },
  91         { EDT_BADCONV,  "Incomplete printf conversion" },
  92         { EDT_BADERROR, "Invalid library ERROR action" },
  93         { EDT_ERRABORT, "Abort due to error" },
  94         { EDT_DROPABORT, "Abort due to drop" },
  95         { EDT_DIRABORT, "Abort explicitly directed" },
  96         { EDT_BADRVAL,  "Invalid return value from callback" },
```



```
  97         { EDT_BADNORMAL, "Invalid normalization" },
  98         { EDT_BUFTOOSMALL, "Enabling exceeds size of buffer" },
  99         { EDT_BADTRUNC, "Invalid truncation" },
 100         { EDT_BUSY, "DTrace cannot be used when kernel debugger is active" },
 101         { EDT_ACCESS, "DTrace requires additional privileges" },
 102         { EDT_NOENT, "DTrace device not available on system" },
 103         { EDT_BRICKED, "Abort due to systemic unresponsiveness" },
 104         { EDT_HARDWIRE, "Failed to load language definitions" },
 105         { EDT_ELFVERSION, "libelf is out-of-date with respect to libdtrace" },
 106         { EDT_NOBUFFERED, "Attempt to buffer output without handler" },
 107         { EDT_UNSTABLE, "Description matched an unstable set of probes" },
 108         { EDT_BADSETOPT, "Invalid setopt() library action" },
 109         { EDT_BADSTACKPC, "Invalid stack program counter size" },
 110         { EDT_BADAGGVAR, "Invalid aggregation variable identifier" },
 111         { EDT_OVERSION, "Client requested deprecated version of library" },
 112         { EDT_ENABLING_ERR, "Failed to enable probe" },
 113         { EDT_NOPROBES, "No probe sites found for declared provider" },
 114         { EDT_CANTLOAD, "Failed to load module" },
 115         { EDT_BADCTF, "Module contains invalid or incomplete CTF information" }
 116 #endif /* ! codereview */
 117 };

 119 static const int _dt_nerr = sizeof (_dt_errlist) / sizeof (_dt_errlist[0]);

 121 const char *
 122 dtrace_errmsg(dtrace_hdl_t *dtp, int error)
 123 {
 124         const char *str;
 125         int i;

 127         if (error == EDT_COMPILER && dtp != NULL && dtp->dt_errmsg[0] != '\0')
 128                 str = dtp->dt_errmsg;
 129         else if (error == EDT_CTF && dtp != NULL && dtp->dt_ctferr != 0)
 130                 str = ctf_errmsg(dtp->dt_ctferr);
 131         else if (error >= EDT_BASE && (error - EDT_BASE) < _dt_nerr) {
 132                 for (i = 0; i < _dt_nerr; i++) {
 133                         if (_dt_errlist[i].err == error)
 134                                 return (_dt_errlist[i].msg);
 135                 }
 136                 str = NULL;
 137         } else
 138                 str = strerror(error);

 140         return (str ? str : "Unknown error");
 141 }

 143 int
 144 dtrace_errno(dtrace_hdl_t *dtp)
 145 {
 146         return (dtp->dt_errno);
 147 }

 149 int
 150 dt_set_errno(dtrace_hdl_t *dtp, int err)
 151 {
 152         dtp->dt_errno = err;
 153         return (-1);
 154 }

 156 void
 157 dt_set_errmsg(dtrace_hdl_t *dtp, const char *errtag, const char *region,
 158     const char *filename, int lineno, const char *format, va_list ap)
 159 {
 160         size_t len, n;
 161         char *p, *s;
```

```
163         s = dtp->dt_errmsg;
164         n = sizeof (dtp->dt_errmsg);
165
166         if (errtag != NULL && (yypcb->pcb_cflags & DTRACE_C_ETAGS))
167                 (void) snprintf(s, n, "[%s] ", errtag);
168         else
169                 s[0] = '\0';
170
171         len = strlen(dtp->dt_errmsg);
172         s = dtp->dt_errmsg + len;
173         n = sizeof (dtp->dt_errmsg) - len;
174
175         if (filename == NULL)
176                 filename = dtp->dt_filetag;
177
178         if (filename != NULL)
179                 (void) snprintf(s, n, "\"%s\", line %d: ", filename, lineno);
180         else if (lineno != 0)
181                 (void) snprintf(s, n, "line %d: ", lineno);
182         else if (region != NULL)
183                 (void) snprintf(s, n, "in %s: ", region);
184
185         len = strlen(dtp->dt_errmsg);
186         s = dtp->dt_errmsg + len;
187         n = sizeof (dtp->dt_errmsg) - len;
188         (void) vsnprintf(s, n, format, ap);
189
190         if ((p = strrchr(dtp->dt_errmsg, '\n')) != NULL)
191                 *p = '\0'; /* remove trailing \n from message buffer */
192
193         dtp->dt_errtag = errtag;
194 }
195
196 /*ARGSUSED*/
197 const char *
198 dtrace_faultstr(dtrace_hdl_t *dtp, int fault)
199 {
200         int i;
201
202         static const struct {
203                 int code;
204                 const char *str;
205         } faults[] = {
206                 { DTRACEFLT_BADADDR,    "invalid address" },
207                 { DTRACEFLT_BADALIGN,   "invalid alignment" },
208                 { DTRACEFLT_ILLOP,      "illegal operation" },
209                 { DTRACEFLT_DIVZERO,    "divide-by-zero" },
210                 { DTRACEFLT_NOSCRATCH,  "out of scratch space" },
211                 { DTRACEFLT_KPRIV,      "invalid kernel access" },
212                 { DTRACEFLT_UPRIV,      "invalid user access" },
213                 { DTRACEFLT_TUPOFLOW,   "tuple stack overflow" },
214                 { DTRACEFLT_BADSTACK,   "bad stack" },
215                 { DTRACEFLT_LIBRARY,    "library-level fault" },
216                 { 0,                    NULL }
217         };
218
219         for (i = 0; faults[i].str != NULL; i++) {
220                 if (faults[i].code == fault)
221                         return (faults[i].str);
222         }
223
224         return ("unknown fault");
225 }
```

```
**********************************************************
   31295 Fri Apr 17 12:29:26 2015
new/usr/src/lib/libdtrace/common/dt_impl.h
1730 DTrace should ignore type information from modules with cth_parlabel mismat
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
**********************************************************
______unchanged_portion_omitted_

 325 /*
 326  * Values for the user arg of the ECB.
 327  */
 328 #define DT_ECB_DEFAULT          0
 329 #define DT_ECB_ERROR            1

 331 /*
 332  * Values for the dt_linkmode property, which is used by the assembler when
 333  * processing external symbol references.  User can set using -xlink=<mode>.
 334  */
 335 #define DT_LINK_KERNEL  0       /* kernel syms static, user syms dynamic */
 336 #define DT_LINK_PRIMARY 1       /* primary kernel syms static, others dynamic */
 337 #define DT_LINK_DYNAMIC 2       /* all symbols dynamic */
 338 #define DT_LINK_STATIC  3       /* all symbols static */

 340 /*
 341  * Values for the dt_linktype property, which is used by dtrace_program_link()
 342  * to determine the type of output file that is desired by the client.
 343  */
 344 #define DT_LTYP_ELF     0       /* produce ELF containing DOF */
 345 #define DT_LTYP_DOF     1       /* produce stand-alone DOF */

 347 /*
 348  * Values for the dt_xlatemode property, which is used to determine whether
 349  * references to dynamic translators are permitted.  Set using -xlate=<mode>.
 350  */
 351 #define DT_XL_STATIC    0       /* require xlators to be statically defined */
 352 #define DT_XL_DYNAMIC   1       /* produce references to dynamic translators */

 354 /*
 355  * Values for the dt_stdcmode property, which is used by the compiler when
 356  * running cpp to determine the presence and setting of the __STDC__ macro.
 357  */
 358 #define DT_STDC_XA      0       /* ISO C + K&R C compat w/o ISO: __STDC__=0 */
 359 #define DT_STDC_XC      1       /* Strict ISO C: __STDC__=1 */
 360 #define DT_STDC_XS      2       /* K&R C: __STDC__ not defined */
 361 #define DT_STDC_XT      3       /* ISO C + K&R C compat with ISO: __STDC__=0 */

 363 /*
 364  * Values for the dt_encoding property, which is used to force a particular
 365  * character encoding (overriding default behavior and/or automatic detection).
 366  */
 367 #define DT_ENCODING_UNSET       0
 368 #define DT_ENCODING_ASCII       1
 369 #define DT_ENCODING_UTF8        2

 371 /*
 372  * Macro to test whether a given pass bit is set in the dt_treedump bit-vector.
 373  * If the bit for pass 'p' is set, the D compiler displays the parse tree for
 374  * the program by printing it to stderr at the end of compiler pass 'p'.
 375  */
 376 #define DT_TREEDUMP_PASS(dtp, p)        ((dtp)->dt_treedump & (1 << ((p) - 1)))

 378 /*
 379  * Macros for accessing the cached CTF container and type ID for the common
 380  * types "int", "string", and <DYN>, which we need to use frequently in the D
 381  * compiler.  The DT_INT_* macro relies upon "int" being at index 0 in the
```



```
 382  * _dtrace_ints_* tables in dt_open.c; the others are also set up there.
 383  */
 384 #define DT_INT_CTFP(dtp)        ((dtp)->dt_ints[0].did_ctfp)
 385 #define DT_INT_TYPE(dtp)        ((dtp)->dt_ints[0].did_type)

 387 #define DT_FUNC_CTFP(dtp)       ((dtp)->dt_ddefs->dm_ctfp)
 388 #define DT_FUNC_TYPE(dtp)       ((dtp)->dt_type_func)

 390 #define DT_FPTR_CTFP(dtp)       ((dtp)->dt_ddefs->dm_ctfp)
 391 #define DT_FPTR_TYPE(dtp)       ((dtp)->dt_type_fptr)

 393 #define DT_STR_CTFP(dtp)        ((dtp)->dt_ddefs->dm_ctfp)
 394 #define DT_STR_TYPE(dtp)        ((dtp)->dt_type_str)

 396 #define DT_DYN_CTFP(dtp)        ((dtp)->dt_ddefs->dm_ctfp)
 397 #define DT_DYN_TYPE(dtp)        ((dtp)->dt_type_dyn)

 399 #define DT_STACK_CTFP(dtp)      ((dtp)->dt_ddefs->dm_ctfp)
 400 #define DT_STACK_TYPE(dtp)      ((dtp)->dt_type_stack)

 402 #define DT_SYMADDR_CTFP(dtp)    ((dtp)->dt_ddefs->dm_ctfp)
 403 #define DT_SYMADDR_TYPE(dtp)    ((dtp)->dt_type_symaddr)

 405 #define DT_USYMADDR_CTFP(dtp)   ((dtp)->dt_ddefs->dm_ctfp)
 406 #define DT_USYMADDR_TYPE(dtp)   ((dtp)->dt_type_usymaddr)

 408 /*
 409  * Actions and subroutines are both DT_NODE_FUNC nodes; to avoid confusing
 410  * an action for a subroutine (or vice versa), we assure that the DT_ACT_*
 411  * constants and the DIF_SUBR_* constants occupy non-overlapping ranges by
 412  * starting the DT_ACT_* constants at DIF_SUBR_MAX + 1.
 413  */
 414 #define DT_ACT_BASE             DIF_SUBR_MAX + 1
 415 #define DT_ACT(n)               (DT_ACT_BASE + (n))

 417 #define DT_ACT_PRINTF           DT_ACT(0)       /* printf() action */
 418 #define DT_ACT_TRACE            DT_ACT(1)       /* trace() action */
 419 #define DT_ACT_TRACEMEM         DT_ACT(2)       /* tracemem() action */
 420 #define DT_ACT_STACK            DT_ACT(3)       /* stack() action */
 421 #define DT_ACT_STOP             DT_ACT(4)       /* stop() action */
 422 #define DT_ACT_BREAKPOINT       DT_ACT(5)       /* breakpoint() action */
 423 #define DT_ACT_PANIC            DT_ACT(6)       /* panic() action */
 424 #define DT_ACT_SPECULATE        DT_ACT(7)       /* speculate() action */
 425 #define DT_ACT_COMMIT           DT_ACT(8)       /* commit() action */
 426 #define DT_ACT_DISCARD          DT_ACT(9)       /* discard() action */
 427 #define DT_ACT_CHILL            DT_ACT(10)      /* chill() action */
 428 #define DT_ACT_EXIT             DT_ACT(11)      /* exit() action */
 429 #define DT_ACT_USTACK           DT_ACT(12)      /* ustack() action */
 430 #define DT_ACT_PRINTA           DT_ACT(13)      /* printa() action */
 431 #define DT_ACT_RAISE            DT_ACT(14)      /* raise() action */
 432 #define DT_ACT_CLEAR            DT_ACT(15)      /* clear() action */
 433 #define DT_ACT_NORMALIZE        DT_ACT(16)      /* normalize() action */
 434 #define DT_ACT_DENORMALIZE      DT_ACT(17)      /* denormalize() action */
 435 #define DT_ACT_TRUNC            DT_ACT(18)      /* trunc() action */
 436 #define DT_ACT_SYSTEM           DT_ACT(19)      /* system() action */
 437 #define DT_ACT_JSTACK           DT_ACT(20)      /* jstack() action */
 438 #define DT_ACT_FTRUNCATE        DT_ACT(21)      /* ftruncate() action */
 439 #define DT_ACT_FREOPEN          DT_ACT(22)      /* freopen() action */
 440 #define DT_ACT_SYM              DT_ACT(23)      /* sym()/func() actions */
 441 #define DT_ACT_MOD              DT_ACT(24)      /* mod() action */
 442 #define DT_ACT_USYM             DT_ACT(25)      /* usym()/ufunc() actions */
 443 #define DT_ACT_UMOD             DT_ACT(26)      /* umod() action */
 444 #define DT_ACT_UADDR            DT_ACT(27)      /* uaddr() action */
 445 #define DT_ACT_SETOPT           DT_ACT(28)      /* setopt() action */
 446 #define DT_ACT_PRINT            DT_ACT(29)      /* print() action */
```

```
448 /*
449  * Sentinel to tell freopen() to restore the saved stdout.  This must not
450  * be ever valid for opening for write access via freopen(3C), which of
451  * course, "." never is.
452  */
453 #define DT_FREOPEN_RESTORE      "."

455 #define EDT_BASE        1000    /* base value for libdtrace errnos */

457 enum {
458         EDT_VERSION = EDT_BASE, /* client is requesting unsupported version */
459         EDT_VERSINVAL,          /* version string is invalid or overflows */
460         EDT_VERSUNDEF,          /* requested API version is not defined */
461         EDT_VERSREDUCED,        /* requested API version has been reduced */
462         EDT_CTF,                /* libctf called failed (dt_ctferr has more) */
463         EDT_COMPILER,           /* error in D program compilation */
464         EDT_NOTUPREG,           /* tuple register allocation failure */
465         EDT_NOMEM,              /* memory allocation failure */
466         EDT_INT2BIG,            /* integer limit exceeded */
467         EDT_STR2BIG,            /* string limit exceeded */
468         EDT_NOMOD,              /* unknown module name */
469         EDT_NOPROV,             /* unknown provider name */
470         EDT_NOPROBE,            /* unknown probe name */
471         EDT_NOSYM,              /* unknown symbol name */
472         EDT_NOSYMADDR,          /* no symbol corresponds to address */
473         EDT_NOTYPE,             /* unknown type name */
474         EDT_NOVAR,              /* unknown variable name */
475         EDT_NOAGG,              /* unknown aggregation name */
476         EDT_BADSCOPE,           /* improper use of type name scoping operator */
477         EDT_BADSPEC,            /* overspecified probe description */
478         EDT_BADSPCV,            /* bad macro variable in probe description */
479         EDT_BADID,              /* invalid probe identifier */
480         EDT_NOTLOADED,          /* module is not currently loaded */
481         EDT_NOCTF,              /* module does not contain any CTF data */
482         EDT_DATAMODEL,          /* module and program data models don't match */
483         EDT_DIFVERS,            /* library has newer DIF version than driver */
484         EDT_BADAGG,             /* unrecognized aggregating action */
485         EDT_FIO,                /* file i/o error */
486         EDT_DIFINVAL,           /* invalid DIF program */
487         EDT_DIFSIZE,            /* invalid DIF size */
488         EDT_DIFFAULT,           /* failed to copyin DIF program */
489         EDT_BADPROBE,           /* bad probe description */
490         EDT_BADPGLOB,           /* bad probe description globbing pattern */
491         EDT_NOSCOPE,            /* declaration scope stack underflow */
492         EDT_NODECL,             /* declaration stack underflow */
493         EDT_DMISMATCH,          /* record list does not match statement */
494         EDT_DOFFSET,            /* record data offset error */
495         EDT_DALIGN,             /* record data alignment error */
496         EDT_BADOPTNAME,         /* invalid dtrace_setopt option name */
497         EDT_BADOPTVAL,          /* invalid dtrace_setopt option value */
498         EDT_BADOPTCTX,          /* invalid dtrace_setopt option context */
499         EDT_CPPFORK,            /* failed to fork preprocessor */
500         EDT_CPPEXEC,            /* failed to exec preprocessor */
501         EDT_CPPENT,             /* preprocessor not found */
502         EDT_CPPERR,             /* unknown preprocessor error */
503         EDT_SYMOFLOW,           /* external symbol table overflow */
504         EDT_ACTIVE,             /* operation illegal when tracing is active */
505         EDT_DESTRUCTIVE,        /* destructive actions not allowed */
506         EDT_NOANON,             /* no anonymous tracing state */
507         EDT_ISANON,             /* can't claim anon state and enable probes */
508         EDT_ENDTOOBIG,          /* END enablings exceed size of prncpl buffer */
509         EDT_NOCONV,             /* failed to load type for printf conversion */
510         EDT_BADCONV,            /* incomplete printf conversion */
511         EDT_BADERROR,           /* invalid library ERROR action */
512         EDT_ERRABORT,           /* abort due to error */
513         EDT_DROPABORT,          /* abort due to drop */
```



```
514         EDT_DIRABORT,           /* abort explicitly directed */
515         EDT_BADRVAL,            /* invalid return value from callback */
516         EDT_BADNORMAL,          /* invalid normalization */
517         EDT_BUFTOOSMALL,        /* enabling exceeds size of buffer */
518         EDT_BADTRUNC,           /* invalid truncation */
519         EDT_BUSY,               /* device busy (active kernel debugger) */
520         EDT_ACCESS,             /* insufficient privileges to use DTrace */
521         EDT_NOENT,              /* dtrace device not available */
522         EDT_BRICKED,            /* abort due to systemic unresponsiveness */
523         EDT_HARDWIRE,           /* failed to load hard-wired definitions */
524         EDT_ELFVERSION,         /* libelf is out-of-date w.r.t libdtrace */
525         EDT_NOBUFFERED,         /* attempt to buffer output without handler */
526         EDT_UNSTABLE,           /* description matched unstable set of probes */
527         EDT_BADSETOPT,          /* invalid setopt library action */
528         EDT_BADSTACKPC,         /* invalid stack program counter size */
529         EDT_BADAGGVAR,          /* invalid aggregation variable identifier */
530         EDT_OVERSION,           /* client is requesting deprecated version */
531         EDT_ENABLING_ERR,       /* failed to enable probe */
532         EDT_NOPROBES,           /* no probes sites for declared provider */
533         EDT_CANTLOAD,           /* failed to load a module */
534         EDT_BADCTF              /* module contains invalid CTF */
533         EDT_CANTLOAD            /* failed to load a module */
535 };
______unchanged_portion_omitted_
```

```
**********************************************************
   42782 Fri Apr 17 12:29:27 2015
new/usr/src/lib/libdtrace/common/dt_module.c
libdtrace: attempt to resolve FORWARD types to concrete types
1730 DTrace should ignore type information from modules with cth_parlabel mismat
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
**********************************************************
______unchanged_portion_omitted_

 795 ctf_file_t *
 796 dt_module_getctf(dtrace_hdl_t *dtp, dt_module_t *dmp)
 797 {
 798         const char *parent;
 799         dt_module_t *pmp;
 800         ctf_file_t *pfp;
 801         int model;

 803         if (dmp->dm_ctfp != NULL || dt_module_load(dtp, dmp) != 0)
 804                 return (dmp->dm_ctfp);

 806         if (dmp->dm_ops == &dt_modops_64)
 807                 model = CTF_MODEL_LP64;
 808         else
 809                 model = CTF_MODEL_ILP32;

 811         /*
 812          * If the data model of the module does not match our program data
 813          * model, then do not permit CTF from this module to be opened and
 814          * returned to the compiler.  If we support mixed data models in the
 815          * future for combined kernel/user tracing, this can be removed.
 816          */
 817         if (dtp->dt_conf.dtc_ctfmodel != model) {
 818                 (void) dt_set_errno(dtp, EDT_DATAMODEL);
 819                 return (NULL);
 820         }

 822         if (dmp->dm_ctdata.cts_size == 0) {
 823                 (void) dt_set_errno(dtp, EDT_NOCTF);
 824                 return (NULL);
 825         }

 827         dmp->dm_ctfp = ctf_bufopen(&dmp->dm_ctdata,
 828             &dmp->dm_symtab, &dmp->dm_strtab, &dtp->dt_ctferr);

 830         if (dmp->dm_ctfp == NULL) {
 831                 (void) dt_set_errno(dtp, EDT_CTF);
 832                 return (NULL);
 833         }

 835         (void) ctf_setmodel(dmp->dm_ctfp, model);
 836         ctf_setspecific(dmp->dm_ctfp, dmp);

 838         if ((parent = ctf_parent_name(dmp->dm_ctfp)) != NULL) {
 839                 if ((pmp = dt_module_create(dtp, parent)) == NULL ||
 840                     (pfp = dt_module_getctf(dtp, pmp)) == NULL) {
 841                         if (pmp == NULL)
 842                                 (void) dt_set_errno(dtp, EDT_NOMEM);
 843                         goto err;
 844                 }

 846                 /*
 847                  * If the label we claim the parent must have is not actually
 848                  * present in the parent module, ignore the CTF entirely
 849                  * rather than acquiring possibly bad type references.
 850                  */
```



```
 851                 if (ctf_label_info(pfp, ctf_parent_label(dmp->dm_ctfp),
 852                     NULL) == CTF_ERR) {
 853                         (void) dt_set_errno(dtp, EDT_BADCTF);
 854                         goto err;
 855                 }

 857 #endif /* ! codereview */
 858                 if (ctf_import(dmp->dm_ctfp, pfp) == CTF_ERR) {
 859                         dtp->dt_ctferr = ctf_errno(dmp->dm_ctfp);
 860                         (void) dt_set_errno(dtp, EDT_CTF);
 861                         goto err;
 862                 }
 863         }

 865         dt_dprintf("loaded CTF container for %s (%p)\n",
 866             dmp->dm_name, (void *)dmp->dm_ctfp);

 868         return (dmp->dm_ctfp);

 870 err:
 871         dt_dprintf("could not load CTF container for %s: %s\n",
 872             dmp->dm_name, dtrace_errmsg(dtp, dtrace_errno(dtp)));
 873 #endif /* ! codereview */
 874         ctf_close(dmp->dm_ctfp);
 875         dmp->dm_ctfp = NULL;
 876         return (NULL);
 877 }

 879 /*ARGSUSED*/
 880 void
 881 dt_module_unload(dtrace_hdl_t *dtp, dt_module_t *dmp)
 882 {
 883         int i;

 885         ctf_close(dmp->dm_ctfp);
 886         dmp->dm_ctfp = NULL;

 888         if (dmp->dm_libctfp != NULL) {
 889                 for (i = 0; i < dmp->dm_nctflibs; i++) {
 890                         ctf_close(dmp->dm_libctfp[i]);
 891                         free(dmp->dm_libctfn[i]);
 892                 }
 893                 free(dmp->dm_libctfp);
 894                 free(dmp->dm_libctfn);
 895                 dmp->dm_libctfp = NULL;
 896                 dmp->dm_nctflibs = 0;
 897         }

 899         bzero(&dmp->dm_ctdata, sizeof (ctf_sect_t));
 900         bzero(&dmp->dm_symtab, sizeof (ctf_sect_t));
 901         bzero(&dmp->dm_strtab, sizeof (ctf_sect_t));

 903         if (dmp->dm_symbuckets != NULL) {
 904                 free(dmp->dm_symbuckets);
 905                 dmp->dm_symbuckets = NULL;
 906         }

 908         if (dmp->dm_symchains != NULL) {
 909                 free(dmp->dm_symchains);
 910                 dmp->dm_symchains = NULL;
 911         }

 913         if (dmp->dm_asmap != NULL) {
 914                 free(dmp->dm_asmap);
 915                 dmp->dm_asmap = NULL;
 916         }
```

```
        dmp->dm_symfree = 0;
        dmp->dm_nsymbuckets = 0;
        dmp->dm_nsymelems = 0;
        dmp->dm_asrsv = 0;
        dmp->dm_aslen = 0;

        dmp->dm_text_va = NULL;
        dmp->dm_text_size = 0;
        dmp->dm_data_va = NULL;
        dmp->dm_data_size = 0;
        dmp->dm_bss_va = NULL;
        dmp->dm_bss_size = 0;

        if (dmp->dm_extern != NULL) {
                dt_idhash_destroy(dmp->dm_extern);
                dmp->dm_extern = NULL;
        }

        (void) elf_end(dmp->dm_elf);
        dmp->dm_elf = NULL;

        dmp->dm_pid = 0;

        dmp->dm_flags &= ~DT_DM_LOADED;
}

void
dt_module_destroy(dtrace_hdl_t *dtp, dt_module_t *dmp)
{
        uint_t h = dt_strtab_hash(dmp->dm_name, NULL) % dtp->dt_modbuckets;
        dt_module_t **dmpp = &dtp->dt_mods[h];

        dt_list_delete(&dtp->dt_modlist, dmp);
        assert(dtp->dt_nmods != 0);
        dtp->dt_nmods--;

        /*
         * Now remove this module from its hash chain.  We expect to always
         * find the module on its hash chain, so in this loop we assert that
         * we don't run off the end of the list.
         */
        while (*dmpp != dmp) {
                dmpp = &((*dmpp)->dm_next);
                assert(*dmpp != NULL);
        }

        *dmpp = dmp->dm_next;

        dt_module_unload(dtp, dmp);
        free(dmp);
}

/*
 * Insert a new external symbol reference into the specified module.  The new
 * symbol will be marked as undefined and is assigned a symbol index beyond
 * any existing cached symbols from this module.  We use the ident's di_data
 * field to store a pointer to a copy of the dtrace_syminfo_t for this symbol.
 */
dt_ident_t *
dt_module_extern(dtrace_hdl_t *dtp, dt_module_t *dmp,
    const char *name, const dtrace_typeinfo_t *tip)
{
        dtrace_syminfo_t *sip;
        dt_ident_t *idp;
        uint_t id;
```



```
        if (dmp->dm_extern == NULL && (dmp->dm_extern = dt_idhash_create(
            "extern", NULL, dmp->dm_nsymelems, UINT_MAX)) == NULL) {
                (void) dt_set_errno(dtp, EDT_NOMEM);
                return (NULL);
        }

        if (dt_idhash_nextid(dmp->dm_extern, &id) == -1) {
                (void) dt_set_errno(dtp, EDT_SYMOFLOW);
                return (NULL);
        }

        if ((sip = malloc(sizeof (dtrace_syminfo_t))) == NULL) {
                (void) dt_set_errno(dtp, EDT_NOMEM);
                return (NULL);
        }

        idp = dt_idhash_insert(dmp->dm_extern, name, DT_IDENT_SYMBOL, 0, id,
            _dtrace_symattr, 0, &dt_idops_thaw, NULL, dtp->dt_gen);

        if (idp == NULL) {
                (void) dt_set_errno(dtp, EDT_NOMEM);
                free(sip);
                return (NULL);
        }

        sip->dts_object = dmp->dm_name;
        sip->dts_name = idp->di_name;
        sip->dts_id = idp->di_id;

        idp->di_data = sip;
        idp->di_ctfp = tip->dtt_ctfp;
        idp->di_type = tip->dtt_type;

        return (idp);
}

const char *
dt_module_modelname(dt_module_t *dmp)
{
        if (dmp->dm_ops == &dt_modops_64)
                return ("64-bit");
        else
                return ("32-bit");
}

/* ARGSUSED */
int
dt_module_getlibid(dtrace_hdl_t *dtp, dt_module_t *dmp, const ctf_file_t *fp)
{
        int i;

        for (i = 0; i < dmp->dm_nctflibs; i++) {
                if (dmp->dm_libctfp[i] == fp)
                        return (i);
        }

        return (-1);
}

/* ARGSUSED */
ctf_file_t *
dt_module_getctflib(dtrace_hdl_t *dtp, dt_module_t *dmp, const char *name)
{
        int i;
```

```
1049         for (i = 0; i < dmp->dm_nctflibs; i++) {
1050                 if (strcmp(dmp->dm_libctfn[i], name) == 0)
1051                         return (dmp->dm_libctfp[i]);
1052         }

1054         return (NULL);
1055 }

1057 /*
1058  * Update our module cache by adding an entry for the specified module 'name'.
1059  * We create the dt_module_t and populate it using /system/object/<name>/.
1060  */
1061 static void
1062 dt_module_update(dtrace_hdl_t *dtp, const char *name)
1063 {
1064         char fname[MAXPATHLEN];
1065         struct stat64 st;
1066         int fd, err, bits;

1068         dt_module_t *dmp;
1069         const char *s;
1070         size_t shstrs;
1071         GElf_Shdr sh;
1072         Elf_Data *dp;
1073         Elf_Scn *sp;

1075         (void) snprintf(fname, sizeof (fname),
1076             "%s/%s/object", OBJFS_ROOT, name);

1078         if ((fd = open(fname, O_RDONLY)) == -1 || fstat64(fd, &st) == -1 ||
1079             (dmp = dt_module_create(dtp, name)) == NULL) {
1080                 dt_dprintf("failed to open %s: %s\n", fname, strerror(errno));
1081                 (void) close(fd);
1082                 return;
1083         }

1085         /*
1086          * Since the module can unload out from under us (and /system/object
1087          * will return ENOENT), tell libelf to cook the entire file now and
1088          * then close the underlying file descriptor immediately.  If this
1089          * succeeds, we know that we can continue safely using dmp->dm_elf.
1090          */
1091         dmp->dm_elf = elf_begin(fd, ELF_C_READ, NULL);
1092         err = elf_cntl(dmp->dm_elf, ELF_C_FDREAD);
1093         (void) close(fd);

1095         if (dmp->dm_elf == NULL || err == -1 ||
1096             elf_getshdrstrndx(dmp->dm_elf, &shstrs) == -1) {
1097                 dt_dprintf("failed to load %s: %s\n",
1098                     fname, elf_errmsg(elf_errno()));
1099                 dt_module_destroy(dtp, dmp);
1100                 return;
1101         }

1103         switch (gelf_getclass(dmp->dm_elf)) {
1104         case ELFCLASS32:
1105                 dmp->dm_ops = &dt_modops_32;
1106                 bits = 32;
1107                 break;
1108         case ELFCLASS64:
1109                 dmp->dm_ops = &dt_modops_64;
1110                 bits = 64;
1111                 break;
1112         default:
1113                 dt_dprintf("failed to load %s: unknown ELF class\n", fname);
1114                 dt_module_destroy(dtp, dmp);
```



```
1115                 return;
1116         }

1118         /*
1119          * Iterate over the section headers locating various sections of
1120          * interest and use their attributes to flesh out the dt_module_t.
1121          */
1122         for (sp = NULL; (sp = elf_nextscn(dmp->dm_elf, sp)) != NULL; ) {
1123                 if (gelf_getshdr(sp, &sh) == NULL || sh.sh_type == SHT_NULL ||
1124                     (s = elf_strptr(dmp->dm_elf, shstrs, sh.sh_name)) == NULL)
1125                         continue; /* skip any malformed sections */

1127                 if (strcmp(s, ".text") == 0) {
1128                         dmp->dm_text_size = sh.sh_size;
1129                         dmp->dm_text_va = sh.sh_addr;
1130                 } else if (strcmp(s, ".data") == 0) {
1131                         dmp->dm_data_size = sh.sh_size;
1132                         dmp->dm_data_va = sh.sh_addr;
1133                 } else if (strcmp(s, ".bss") == 0) {
1134                         dmp->dm_bss_size = sh.sh_size;
1135                         dmp->dm_bss_va = sh.sh_addr;
1136                 } else if (strcmp(s, ".info") == 0 &&
1137                     (dp = elf_getdata(sp, NULL)) != NULL) {
1138                         bcopy(dp->d_buf, &dmp->dm_info,
1139                             MIN(sh.sh_size, sizeof (dmp->dm_info)));
1140                 } else if (strcmp(s, ".filename") == 0 &&
1141                     (dp = elf_getdata(sp, NULL)) != NULL) {
1142                         (void) strlcpy(dmp->dm_file,
1143                             dp->d_buf, sizeof (dmp->dm_file));
1144                 }
1145         }

1147         dmp->dm_flags |= DT_DM_KERNEL;
1148         dmp->dm_modid = (int)OBJFS_MODID(st.st_ino);

1150         if (dmp->dm_info.objfs_info_primary)
1151                 dmp->dm_flags |= DT_DM_PRIMARY;

1153         dt_dprintf("opened %d-bit module %s (%s) [%d]\n",
1154             bits, dmp->dm_name, dmp->dm_file, dmp->dm_modid);
1155 }

1157 /*
1158  * Unload all the loaded modules and then refresh the module cache with the
1159  * latest list of loaded modules and their address ranges.
1160  */
1161 void
1162 dtrace_update(dtrace_hdl_t *dtp)
1163 {
1164         dt_module_t *dmp;
1165         DIR *dirp;

1167         for (dmp = dt_list_next(&dtp->dt_modlist);
1168             dmp != NULL; dmp = dt_list_next(dmp))
1169                 dt_module_unload(dtp, dmp);

1171         /*
1172          * Open /system/object and attempt to create a libdtrace module for
1173          * each kernel module that is loaded on the current system.
1174          */
1175         if (!(dtp->dt_oflags & DTRACE_O_NOSYS) &&
1176             (dirp = opendir(OBJFS_ROOT)) != NULL) {
1177                 struct dirent *dp;

1179                 while ((dp = readdir(dirp)) != NULL) {
1180                         if (dp->d_name[0] != '.')
```

```
1181                                dt_module_update(dtp, dp->d_name);
1182                }
1183
1184                (void) closedir(dirp);
1185        }
1186
1187        /*
1188         * Look up all the macro identifiers and set di_id to the latest value.
1189         * This code collaborates with dt_lex.l on the use of di_id.  We will
1190         * need to implement something fancier if we need to support non-ints.
1191         */
1192        dt_idhash_lookup(dtp->dt_macros, "egid")->di_id = getegid();
1193        dt_idhash_lookup(dtp->dt_macros, "euid")->di_id = geteuid();
1194        dt_idhash_lookup(dtp->dt_macros, "gid")->di_id = getgid();
1195        dt_idhash_lookup(dtp->dt_macros, "pid")->di_id = getpid();
1196        dt_idhash_lookup(dtp->dt_macros, "pgid")->di_id = getpgid(0);
1197        dt_idhash_lookup(dtp->dt_macros, "ppid")->di_id = getppid();
1198        dt_idhash_lookup(dtp->dt_macros, "projid")->di_id = getprojid();
1199        dt_idhash_lookup(dtp->dt_macros, "sid")->di_id = getsid(0);
1200        dt_idhash_lookup(dtp->dt_macros, "taskid")->di_id = gettaskid();
1201        dt_idhash_lookup(dtp->dt_macros, "uid")->di_id = getuid();
1202
1203        /*
1204         * Cache the pointers to the modules representing the base executable
1205         * and the run-time linker in the dtrace client handle. Note that on
1206         * x86 krtld is folded into unix, so if we don't find it, use unix
1207         * instead.
1208         */
1209        dtp->dt_exec = dt_module_lookup_by_name(dtp, "genunix");
1210        dtp->dt_rtld = dt_module_lookup_by_name(dtp, "krtld");
1211        if (dtp->dt_rtld == NULL)
1212                dtp->dt_rtld = dt_module_lookup_by_name(dtp, "unix");
1213
1214        /*
1215         * If this is the first time we are initializing the module list,
1216         * remove the module for genunix from the module list and then move it
1217         * to the front of the module list.  We do this so that type and symbol
1218         * queries encounter genunix and thereby optimize for the common case
1219         * in dtrace_lookup_by_name() and dtrace_lookup_by_type(), below.
1220         */
1221        if (dtp->dt_exec != NULL &&
1222            dtp->dt_cdefs == NULL && dtp->dt_ddefs == NULL) {
1223                dt_list_delete(&dtp->dt_modlist, dtp->dt_exec);
1224                dt_list_prepend(&dtp->dt_modlist, dtp->dt_exec);
1225        }
1226 }
1227
1228 static dt_module_t *
1229 dt_module_from_object(dtrace_hdl_t *dtp, const char *object)
1230 {
1231        int err = EDT_NOMOD;
1232        dt_module_t *dmp;
1233
1234        switch ((uintptr_t)object) {
1235        case (uintptr_t)DTRACE_OBJ_EXEC:
1236                dmp = dtp->dt_exec;
1237                break;
1238        case (uintptr_t)DTRACE_OBJ_RTLD:
1239                dmp = dtp->dt_rtld;
1240                break;
1241        case (uintptr_t)DTRACE_OBJ_CDEFS:
1242                dmp = dtp->dt_cdefs;
1243                break;
1244        case (uintptr_t)DTRACE_OBJ_DDEFS:
1245                dmp = dtp->dt_ddefs;
1246                break;
```



```
1247        default:
1248                dmp = dt_module_create(dtp, object);
1249                err = EDT_NOMEM;
1250        }
1251
1252        if (dmp == NULL)
1253                (void) dt_set_errno(dtp, err);
1254
1255        return (dmp);
1256 }
1257
1258 /*
1259  * Exported interface to look up a symbol by name.  We return the GElf_Sym and
1260  * complete symbol information for the matching symbol.
1261  */
1262 int
1263 dtrace_lookup_by_name(dtrace_hdl_t *dtp, const char *object, const char *name,
1264     GElf_Sym *symp, dtrace_syminfo_t *sip)
1265 {
1266        dt_module_t *dmp;
1267        dt_ident_t *idp;
1268        uint_t n, id;
1269        GElf_Sym sym;
1270
1271        uint_t mask = 0; /* mask of dt_module flags to match */
1272        uint_t bits = 0; /* flag bits that must be present */
1273
1274        if (object != DTRACE_OBJ_EVERY &&
1275            object != DTRACE_OBJ_KMODS &&
1276            object != DTRACE_OBJ_UMODS) {
1277                if ((dmp = dt_module_from_object(dtp, object)) == NULL)
1278                        return (-1); /* dt_errno is set for us */
1279
1280                if (dt_module_load(dtp, dmp) == -1)
1281                        return (-1); /* dt_errno is set for us */
1282                n = 1;
1283
1284        } else {
1285                if (object == DTRACE_OBJ_KMODS)
1286                        mask = bits = DT_DM_KERNEL;
1287                else if (object == DTRACE_OBJ_UMODS)
1288                        mask = DT_DM_KERNEL;
1289
1290                dmp = dt_list_next(&dtp->dt_modlist);
1291                n = dtp->dt_nmods;
1292        }
1293
1294        if (symp == NULL)
1295                symp = &sym;
1296
1297        for (; n > 0; n--, dmp = dt_list_next(dmp)) {
1298                if ((dmp->dm_flags & mask) != bits)
1299                        continue; /* failed to match required attributes */
1300
1301                if (dt_module_load(dtp, dmp) == -1)
1302                        continue; /* failed to load symbol table */
1303
1304                if (dmp->dm_ops->do_symname(dmp, name, symp, &id) != NULL) {
1305                        if (sip != NULL) {
1306                                sip->dts_object = dmp->dm_name;
1307                                sip->dts_name = (const char *)
1308                                    dmp->dm_strtab.cts_data + symp->st_name;
1309                                sip->dts_id = id;
1310                        }
1311                        return (0);
1312                }
```

```
1314                if (dmp->dm_extern != NULL &&
1315                    (idp = dt_idhash_lookup(dmp->dm_extern, name)) != NULL) {
1316                        if (symp != &sym) {
1317                                symp->st_name = (uintptr_t)idp->di_name;
1318                                symp->st_info =
1319                                    GELF_ST_INFO(STB_GLOBAL, STT_NOTYPE);
1320                                symp->st_other = 0;
1321                                symp->st_shndx = SHN_UNDEF;
1322                                symp->st_value = 0;
1323                                symp->st_size =
1324                                    ctf_type_size(idp->di_ctfp, idp->di_type);
1325                        }

1327                        if (sip != NULL) {
1328                                sip->dts_object = dmp->dm_name;
1329                                sip->dts_name = idp->di_name;
1330                                sip->dts_id = idp->di_id;
1331                        }

1333                        return (0);
1334                }
1335        }

1337        return (dt_set_errno(dtp, EDT_NOSYM));
1338 }

1340 /*
1341  * Exported interface to look up a symbol by address.  We return the GElf_Sym
1342  * and complete symbol information for the matching symbol.
1343  */
1344 int
1345 dtrace_lookup_by_addr(dtrace_hdl_t *dtp, GElf_Addr addr,
1346     GElf_Sym *symp, dtrace_syminfo_t *sip)
1347 {
1348        dt_module_t *dmp;
1349        uint_t id;
1350        const dtrace_vector_t *v = dtp->dt_vector;

1352        if (v != NULL)
1353                return (v->dtv_lookup_by_addr(dtp->dt_varg, addr, symp, sip));

1355        for (dmp = dt_list_next(&dtp->dt_modlist); dmp != NULL;
1356            dmp = dt_list_next(dmp)) {
1357                if (addr - dmp->dm_text_va < dmp->dm_text_size ||
1358                    addr - dmp->dm_data_va < dmp->dm_data_size ||
1359                    addr - dmp->dm_bss_va < dmp->dm_bss_size)
1360                        break;
1361        }

1363        if (dmp == NULL)
1364                return (dt_set_errno(dtp, EDT_NOSYMADDR));

1366        if (dt_module_load(dtp, dmp) == -1)
1367                return (-1); /* dt_errno is set for us */

1369        if (symp != NULL) {
1370                if (dmp->dm_ops->do_symaddr(dmp, addr, symp, &id) == NULL)
1371                        return (dt_set_errno(dtp, EDT_NOSYMADDR));
1372        }

1374        if (sip != NULL) {
1375                sip->dts_object = dmp->dm_name;

1377                if (symp != NULL) {
1378                        sip->dts_name = (const char *)
```



```
1379                            dmp->dm_strtab.cts_data + symp->st_name;
1380                        sip->dts_id = id;
1381                } else {
1382                        sip->dts_name = NULL;
1383                        sip->dts_id = 0;
1384                }
1385        }

1387        return (0);
1388 }

1390 boolean_t
1391 dt_is_forward_decl(ctf_file_t *file, ctf_id_t type)
1392 {
1393        ctf_id_t kind = ctf_type_kind(file, type);

1395        while ((type = ctf_type_reference(file, type)) != CTF_ERR) {
1396                type = ctf_type_resolve(file, type);
1397                kind = ctf_type_kind(file, type);
1398        }

1400        return (kind == CTF_K_FORWARD);
1401 }

1403 void
1404 dt_resolve_forward_decl(ctf_file_t **ctfp, ctf_id_t *type)
1405 {
1406        char name[DT_TYPE_NAMELEN];

1408        while (dt_is_forward_decl(*ctfp, *type)) {
1409                char *tag = ctf_type_name(*ctfp, *type, name, sizeof (name));
1410                dtrace_typeinfo_t dtt;

1412                if (tag != NULL && dt_type_lookup(tag, &dtt) == 0 &&
1413                    (dtt.dtt_ctfp != *ctfp) || dtt.dtt_type != *type) {
1414                        *ctfp = dtt.dtt_ctfp;
1415                        *type = dtt.dtt_type;
1416                } else {
1417                        /* All we have is the forward definition */
1418                        break;
1419                }
1420        }
1421 }

1423 #endif /* ! codereview */
1424 int
1425 dtrace_lookup_by_type(dtrace_hdl_t *dtp, const char *object, const char *name,
1426     dtrace_typeinfo_t *tip)
1427 {
1428        dtrace_typeinfo_t ti;
1429        dt_module_t *dmp;
1430        int found = 0;
1431        ctf_id_t id;
1432        uint_t n, i;
1433        int justone;
1434        ctf_file_t *fp;
1435        char *buf, *p, *q;

1437        uint_t mask = 0; /* mask of dt_module flags to match */
1438        uint_t bits = 0; /* flag bits that must be present */

1440        if (object != DTRACE_OBJ_EVERY &&
1441            object != DTRACE_OBJ_KMODS &&
1442            object != DTRACE_OBJ_UMODS) {
1443                if ((dmp = dt_module_from_object(dtp, object)) == NULL)
1444                        return (-1); /* dt_errno is set for us */
```

```
1446                 if (dt_module_load(dtp, dmp) == -1)
1447                         return (-1); /* dt_errno is set for us */
1448                 n = 1;
1449                 justone = 1;
1450         } else {
1451                 if (object == DTRACE_OBJ_KMODS)
1452                         mask = bits = DT_DM_KERNEL;
1453                 else if (object == DTRACE_OBJ_UMODS)
1454                         mask = DT_DM_KERNEL;
1455
1456                 dmp = dt_list_next(&dtp->dt_modlist);
1457                 n = dtp->dt_nmods;
1458                 justone = 0;
1459         }
1460
1461         if (tip == NULL)
1462                 tip = &ti;
1463
1464         for (; n > 0; n--, dmp = dt_list_next(dmp)) {
1465                 if ((dmp->dm_flags & mask) != bits)
1466                         continue; /* failed to match required attributes */
1467
1468                 /*
1469                  * If we can't load the CTF container, continue on to the next
1470                  * module.  If our search was scoped to only one module then
1471                  * return immediately leaving dt_errno unmodified.
1472                  */
1473                 if (dt_module_hasctf(dtp, dmp) == 0) {
1474                         if (justone)
1475                                 return (-1);
1476                         continue;
1477                 }
1478
1479                 /*
1480                  * Look up the type in the module's CTF container.  If our
1481                  * match is a forward declaration tag, save this choice in
1482                  * 'tip' and keep going in the hope that we will locate the
1483                  * underlying structure definition.  Otherwise just return.
1484                  */
1485                 if (dmp->dm_pid == 0) {
1486                         id = ctf_lookup_by_name(dmp->dm_ctfp, name);
1487                         fp = dmp->dm_ctfp;
1488                 } else {
1489                         if ((p = strchr(name, ':')) != NULL) {
1490                                 buf = strdup(name);
1491                                 if (buf == NULL)
1492                                         return (dt_set_errno(dtp, EDT_NOMEM));
1493                                 p = strchr(buf, ':');
1494                                 if ((q = strchr(p + 1, ':')) != NULL)
1495                                         p = q;
1496                                 *p = '\0';
1497                                 fp = dt_module_getctflib(dtp, dmp, buf);
1498                                 if (fp == NULL || (id = ctf_lookup_by_name(fp,
1499                                     p + 1)) == CTF_ERR)
1500                                         id = CTF_ERR;
1501                                 free(buf);
1502                         } else {
1503                                 for (i = 0; i < dmp->dm_nctflibs; i++) {
1504                                         fp = dmp->dm_libctfp[i];
1505                                         id = ctf_lookup_by_name(fp, name);
1506                                         if (id != CTF_ERR)
1507                                                 break;
1508                                 }
1509                         }
1510                 }
```



```
1511                 if (id != CTF_ERR) {
1512                         tip->dtt_object = dmp->dm_name;
1513                         tip->dtt_ctfp = fp;
1514                         tip->dtt_type = id;
1515                         if (!dt_is_forward_decl(fp, ctf_type_resolve(fp, id)))
 846                         if (ctf_type_kind(fp, ctf_type_resolve(fp, id)) !=
 847                             CTF_K_FORWARD)
1516                                 return (0);
1517
1518                         found++;
1519                 }
1520         }
1521
1522         if (found == 0)
1523                 return (dt_set_errno(dtp, EDT_NOTYPE));
1524
1525         return (0);
1526 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    2175 Fri Apr 17 12:29:27 2015
new/usr/src/lib/libdtrace/common/dt_module.h
libdtrace: attempt to resolve FORWARD types to concrete types
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013, Joyent, Inc.  All rights reserved.
 */

#ifndef _DT_MODULE_H
#define _DT_MODULE_H

#include <dt_impl.h>

#ifdef  __cplusplus
extern "C" {
#endif

extern dt_module_t *dt_module_create(dtrace_hdl_t *, const char *);
extern int dt_module_load(dtrace_hdl_t *, dt_module_t *);
extern void dt_module_unload(dtrace_hdl_t *, dt_module_t *);
extern void dt_module_destroy(dtrace_hdl_t *, dt_module_t *);

extern dt_module_t *dt_module_lookup_by_name(dtrace_hdl_t *, const char *);
extern dt_module_t *dt_module_lookup_by_ctf(dtrace_hdl_t *, ctf_file_t *);

extern int dt_module_hasctf(dtrace_hdl_t *, dt_module_t *);
extern ctf_file_t *dt_module_getctf(dtrace_hdl_t *, dt_module_t *);
extern dt_ident_t *dt_module_extern(dtrace_hdl_t *, dt_module_t *,
    const char *, const dtrace_typeinfo_t *);

extern const char *dt_module_modelname(dt_module_t *);
extern int dt_module_getlibid(dtrace_hdl_t *, dt_module_t *,
    const ctf_file_t *);
extern ctf_file_t *dt_module_getctflib(dtrace_hdl_t *, dt_module_t *,
    const char *);

extern boolean_t dt_is_forward_decl(ctf_file_t *, ctf_id_t);
extern void dt_resolve_forward_decl(ctf_file_t **, ctf_id_t *);

#endif /* ! codereview */
```



```
#ifdef  __cplusplus
}
#endif

#endif  /* _DT_MODULE_H */
```

```
**********************************************************
  143174 Fri Apr 17 12:29:28 2015
new/usr/src/lib/libdtrace/common/dt_parser.c
libdtrace: attempt to resolve FORWARD types to concrete types
libdtrace: treat DTT_FL_USER properly in the parser
**********************************************************
______unchanged_portion_omitted_

1339 /*
1340  * Create an empty node of type corresponding to the given declaration.
1341  * Explicit references to user types (C or D) are assigned the default
1342  * stability; references to other types are _dtrace_typattr (Private).
1343  */
1344 dt_node_t *
1345 dt_node_type(dt_decl_t *ddp)
1346 {
1347         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
1348         dtrace_typeinfo_t dtt;
1349         dt_node_t *dnp;
1350         char *name = NULL;
1351         int err;

1353         /*
1354          * If 'ddp' is NULL, we get a decl by popping the decl stack.  This
1355          * form of dt_node_type() is used by parameter rules in dt_grammar.y.
1356          */
1357         if (ddp == NULL)
1358                 ddp = dt_decl_pop_param(&name);

1360         err = dt_decl_type(ddp, &dtt);
1361         dt_decl_free(ddp);

1363         if (err != 0) {
1364                 free(name);
1365                 longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);
1366         }

1368         dnp = dt_node_alloc(DT_NODE_TYPE);
1369         dnp->dn_op = DT_TOK_IDENT;
1370         dnp->dn_string = name;

1372         dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type,
1373             dtt.dtt_flags & DTT_FL_USER ? B_TRUE : B_FALSE);
1372         dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type, dtt.dtt_flags);

1375         if (dtt.dtt_ctfp == dtp->dt_cdefs->dm_ctfp ||
1376             dtt.dtt_ctfp == dtp->dt_ddefs->dm_ctfp)
1377                 dt_node_attr_assign(dnp, _dtrace_defattr);
1378         else
1379                 dt_node_attr_assign(dnp, _dtrace_typattr);

1381         return (dnp);
1382 }
______unchanged_portion_omitted_

1787 /*
1788  * The offsetof() function is special because it takes a type name as an
1789  * argument.  It does not actually construct its own node; after looking up the
1790  * structure or union offset, we just return an integer node with the offset.
1791  */
1792 dt_node_t *
1793 dt_node_offsetof(dt_decl_t *ddp, char *s)
1794 {
1795         dtrace_typeinfo_t dtt;
1796         dt_node_t dn;
1797         char *name;
```



```
1798         int err;

1800         ctf_membinfo_t ctm;
1801         ctf_id_t type;
1802         ctf_file_t *ctfp;
1803 #endif /* ! codereview */
1804         uint_t kind;

1806         name = strdupa(s);
1807         free(s);

1809         err = dt_decl_type(ddp, &dtt);
1810         dt_decl_free(ddp);

1812         if (err != 0)
1813                 longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);

1815         type = ctf_type_resolve(dtt.dtt_ctfp, dtt.dtt_type);
1816         kind = ctf_type_kind(dtt.dtt_ctfp, type);

1818         if (kind != CTF_K_STRUCT && kind != CTF_K_UNION) {
1819                 xyerror(D_OFFSETOF_TYPE,
1820                     "offsetof operand must be a struct or union type\n");
1821         }

1823         if (ctf_member_info(dtt.dtt_ctfp, type, name, &ctm) == CTF_ERR) {
1824                 xyerror(D_UNKNOWN, "failed to determine offset of %s: %s\n",
1825                     name, ctf_errmsg(ctf_errno(dtt.dtt_ctfp)));
1826         }

1828         bzero(&dn, sizeof (dn));
1829         /*
1830          * Resolution of CTF_K_FORWARD is unnecessary here, since it can't be
1831          * both forward _and_ a bitfield, but is done for completeness.
1832          */
1833         type = ctm.ctm_type;
1834         ctfp = dtt.dtt_ctfp;

1836         dt_resolve_forward_decl(&ctfp, &type);
1837         dt_node_type_assign(&dn, ctfp, type, B_FALSE);
1801         dt_node_type_assign(&dn, dtt.dtt_ctfp, ctm.ctm_type, B_FALSE);

1839         if (dn.dn_flags & DT_NF_BITFIELD) {
1840                 xyerror(D_OFFSETOF_BITFIELD,
1841                     "cannot take offset of a bit-field: %s\n", name);
1842         }

1844         return (dt_node_int(ctm.ctm_offset / NBBY));
1845 }
______unchanged_portion_omitted_

2416 dt_node_t *
2417 dt_node_member(dt_decl_t *ddp, char *name, dt_node_t *expr)
2418 {
2419         dtrace_typeinfo_t dtt;
2420         dt_node_t *dnp;
2421         int err;

2423         if (ddp != NULL) {
2424                 err = dt_decl_type(ddp, &dtt);
2425                 dt_decl_free(ddp);

2427                 if (err != 0)
2428                         longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);
2429         }
```

```
2431         dnp = dt_node_alloc(DT_NODE_MEMBER);
2432         dnp->dn_membname = name;
2433         dnp->dn_membexpr = expr;

2435         if (ddp != NULL)
2436                 dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type,
2437                     dtt.dtt_flags & DTT_FL_USER ? B_TRUE : B_FALSE);
2401                     dtt.dtt_flags);

2439         return (dnp);
2440 }
```

_____unchanged_portion_omitted_

```
2625 /*
2626  * This function provides the underlying implementation of cooking an
2627  * identifier given its node, a hash of dynamic identifiers, an identifier
2628  * kind, and a boolean flag indicating whether we are allowed to instantiate
2629  * a new identifier if the string is not found.  This function is either
2630  * called from dt_cook_ident(), below, or directly by the various cooking
2631  * routines that are allowed to instantiate identifiers (e.g. op2 TOK_ASGN).
2632  */
2633 static void
2634 dt_xcook_ident(dt_node_t *dnp, dt_idhash_t *dhp, uint_t idkind, int create)
2635 {
2636         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
2637         const char *sname = dt_idhash_name(dhp);
2638         int uref = 0;

2640         dtrace_attribute_t attr = _dtrace_defattr;
2641         dt_ident_t *idp;
2642         dtrace_syminfo_t dts;
2643         GElf_Sym sym;

2645         const char *scope, *mark;
2646         uchar_t dnkind;
2647         char *name;

2649         /*
2650          * Look for scoping marks in the identifier.  If one is found, set our
2651          * scope to either DTRACE_OBJ_KMODS or UMODS or to the first part of
2652          * the string that specifies the scope using an explicit module name.
2653          * If two marks in a row are found, set 'uref' (user symbol reference).
2654          * Otherwise we set scope to DTRACE_OBJ_EXEC, indicating that normal
2655          * scope is desired and we should search the specified idhash.
2656          */
2657         if ((name = strrchr(dnp->dn_string, '`')) != NULL) {
2658                 if (name > dnp->dn_string && name[-1] == '`') {
2659                         uref++;
2660                         name[-1] = '\0';
2661                 }

2663                 if (name == dnp->dn_string + uref)
2664                         scope = uref ? DTRACE_OBJ_UMODS : DTRACE_OBJ_KMODS;
2665                 else
2666                         scope = dnp->dn_string;

2668                 *name++ = '\0'; /* leave name pointing after scoping mark */
2669                 dnkind = DT_NODE_VAR;

2671         } else if (idkind == DT_IDENT_AGG) {
2672                 scope = DTRACE_OBJ_EXEC;
2673                 name = dnp->dn_string + 1;
2674                 dnkind = DT_NODE_AGG;
2675         } else {
2676                 scope = DTRACE_OBJ_EXEC;
2677                 name = dnp->dn_string;
```



```
2678                 dnkind = DT_NODE_VAR;
2679         }

2681         /*
2682          * If create is set to false, and we fail our idhash lookup, preset
2683          * the errno code to EDT_NOVAR for our final error message below.
2684          * If we end up calling dtrace_lookup_by_name(), it will reset the
2685          * errno appropriately and that error will be reported instead.
2686          */
2687         (void) dt_set_errno(dtp, EDT_NOVAR);
2688         mark = uref ? "``" : "`";

2690         if (scope == DTRACE_OBJ_EXEC && (
2691             (dhp != dtp->dt_globals &&
2692             (idp = dt_idhash_lookup(dhp, name)) != NULL) ||
2693             (dhp == dtp->dt_globals &&
2694             (idp = dt_idstack_lookup(&yypcb->pcb_globals, name)) != NULL))) {
2695                 /*
2696                  * Check that we are referencing the ident in the manner that
2697                  * matches its type if this is a global lookup.  In the TLS or
2698                  * local case, we don't know how the ident will be used until
2699                  * the time operator -> is seen; more parsing is needed.
2700                  */
2701                 if (idp->di_kind != idkind && dhp == dtp->dt_globals) {
2702                         xyerror(D_IDENT_BADREF, "%s `%s' may not be referenced "
2703                             "as %s\n", dt_idkind_name(idp->di_kind),
2704                             idp->di_name, dt_idkind_name(idkind));
2705                 }

2707                 /*
2708                  * Arrays and aggregations are not cooked individually. They
2709                  * have dynamic types and must be referenced using operator [].
2710                  * This is handled explicitly by the code for DT_TOK_LBRAC.
2711                  */
2712                 if (idp->di_kind != DT_IDENT_ARRAY &&
2713                     idp->di_kind != DT_IDENT_AGG)
2714                         attr = dt_ident_cook(dnp, idp, NULL);
2715                 else {
2716                         dt_node_type_assign(dnp,
2717                             DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp), B_FALSE);
2718                         attr = idp->di_attr;
2719                 }

2721                 free(dnp->dn_string);
2722                 dnp->dn_string = NULL;
2723                 dnp->dn_kind = dnkind;
2724                 dnp->dn_ident = idp;
2725                 dnp->dn_flags |= DT_NF_LVALUE;

2727                 if (idp->di_flags & DT_IDFLG_WRITE)
2728                         dnp->dn_flags |= DT_NF_WRITABLE;

2730                 dt_node_attr_assign(dnp, attr);

2732         } else if (dhp == dtp->dt_globals && scope != DTRACE_OBJ_EXEC &&
2733             dtrace_lookup_by_name(dtp, scope, name, &sym, &dts) == 0) {

2735                 dt_module_t *mp = dt_module_lookup_by_name(dtp, dts.dts_object);
2736                 int umod = (mp->dm_flags & DT_DM_KERNEL) == 0;
2737                 static const char *const kunames[] = { "kernel", "user" };

2739                 dtrace_typeinfo_t dtt;
2740                 dtrace_syminfo_t *sip;

2742                 if (uref ^ umod) {
2743                         xyerror(D_SYM_BADREF, "%s module `%s' symbol `%s' may "
```

```
2744                            "not be referenced as a %s symbol\n", kunames[umod],
2745                            dts.dts_object, dts.dts_name, kunames[uref]);
2746                }

2748                if (dtrace_symbol_type(dtp, &sym, &dts, &dtt) != 0) {
2749                        /*
2750                         * For now, we special-case EDT_DATAMODEL to clarify
2751                         * that mixed data models are not currently supported.
2752                         */
2753                        if (dtp->dt_errno == EDT_DATAMODEL) {
2754                                xyerror(D_SYM_MODEL, "cannot use %s symbol "
2755                                    "%s%s%s in a %s D program\n",
2756                                    dt_module_modelname(mp),
2757                                    dts.dts_object, mark, dts.dts_name,
2758                                    dt_module_modelname(dtp->dt_ddefs));
2759                        }

2761                        xyerror(D_SYM_NOTYPES,
2762                            "no symbolic type information is available for "
2763                            "%s%s%s: %s\n", dts.dts_object, mark, dts.dts_name,
2764                            dtrace_errmsg(dtp, dtrace_errno(dtp)));
2765                }

2767                idp = dt_ident_create(name, DT_IDENT_SYMBOL, 0, 0,
2768                    _dtrace_symattr, 0, &dt_idops_thaw, NULL, dtp->dt_gen);

2770                if (idp == NULL)
2771                        longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2773                if (mp->dm_flags & DT_DM_PRIMARY)
2774                        idp->di_flags |= DT_IDFLG_PRIM;

2776                idp->di_next = dtp->dt_externs;
2777                dtp->dt_externs = idp;

2779                if ((sip = malloc(sizeof (dtrace_syminfo_t))) == NULL)
2780                        longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2782                bcopy(&dts, sip, sizeof (dtrace_syminfo_t));
2783                idp->di_data = sip;
2784                idp->di_ctfp = dtt.dtt_ctfp;
2785                idp->di_type = dtt.dtt_type;

2787                free(dnp->dn_string);
2788                dnp->dn_string = NULL;
2789                dnp->dn_kind = DT_NODE_SYM;
2790                dnp->dn_ident = idp;
2791                dnp->dn_flags |= DT_NF_LVALUE;

2793                dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type,
2794                    dtt.dtt_flags & DTT_FL_USER ? B_TRUE : B_FALSE);
2758                    dtt.dtt_flags);
2795                dt_node_attr_assign(dnp, _dtrace_symattr);

2797                if (uref) {
2798                        idp->di_flags |= DT_IDFLG_USER;
2799                        dnp->dn_flags |= DT_NF_USERLAND;
2800                }

2802        } else if (scope == DTRACE_OBJ_EXEC && create == B_TRUE) {
2803                uint_t flags = DT_IDFLG_WRITE;
2804                uint_t id;

2806                if (dt_idhash_nextid(dhp, &id) == -1) {
2807                        xyerror(D_ID_OFLOW, "cannot create %s: limit on number "
2808                            "of %s variables exceeded\n", name, sname);
```



```
2809                }

2811                if (dhp == yypcb->pcb_locals)
2812                        flags |= DT_IDFLG_LOCAL;
2813                else if (dhp == dtp->dt_tls)
2814                        flags |= DT_IDFLG_TLS;

2816                dt_dprintf("create %s %s variable %s, id=%u\n",
2817                    sname, dt_idkind_name(idkind), name, id);

2819                if (idkind == DT_IDENT_ARRAY || idkind == DT_IDENT_AGG) {
2820                        idp = dt_idhash_insert(dhp, name,
2821                            idkind, flags, id, _dtrace_defattr, 0,
2822                            &dt_idops_assc, NULL, dtp->dt_gen);
2823                } else {
2824                        idp = dt_idhash_insert(dhp, name,
2825                            idkind, flags, id, _dtrace_defattr, 0,
2826                            &dt_idops_thaw, NULL, dtp->dt_gen);
2827                }

2829                if (idp == NULL)
2830                        longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2832                /*
2833                 * Arrays and aggregations are not cooked individually. They
2834                 * have dynamic types and must be referenced using operator [].
2835                 * This is handled explicitly by the code for DT_TOK_LBRAC.
2836                 */
2837                if (idp->di_kind != DT_IDENT_ARRAY &&
2838                    idp->di_kind != DT_IDENT_AGG)
2839                        attr = dt_ident_cook(dnp, idp, NULL);
2840                else {
2841                        dt_node_type_assign(dnp,
2842                            DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp), B_FALSE);
2843                        attr = idp->di_attr;
2844                }

2846                free(dnp->dn_string);
2847                dnp->dn_string = NULL;
2848                dnp->dn_kind = dnkind;
2849                dnp->dn_ident = idp;
2850                dnp->dn_flags |= DT_NF_LVALUE | DT_NF_WRITABLE;

2852                dt_node_attr_assign(dnp, attr);

2854        } else if (scope != DTRACE_OBJ_EXEC) {
2855                xyerror(D_IDENT_UNDEF, "failed to resolve %s%s%s: %s\n",
2856                    dnp->dn_string, mark, name,
2857                    dtrace_errmsg(dtp, dtrace_errno(dtp)));
2858        } else {
2859                xyerror(D_IDENT_UNDEF, "failed to resolve %s: %s\n",
2860                    dnp->dn_string, dtrace_errmsg(dtp, dtrace_errno(dtp)));
2861        }
2862 }
______unchanged_portion_omitted_

2910 static dt_node_t *
2911 dt_cook_op1(dt_node_t *dnp, uint_t idflags)
2912 {
2913         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
2914         dt_node_t *cp = dnp->dn_child;

2916         char n[DT_TYPE_NAMELEN];
2917         dtrace_typeinfo_t dtt;
2918         dt_ident_t *idp;
```

```
2920         ctf_encoding_t e;
2921         ctf_arinfo_t r;
2922         ctf_id_t type, base;
2923         uint_t kind;

2925         if (dnp->dn_op == DT_TOK_PREINC || dnp->dn_op == DT_TOK_POSTINC ||
2926             dnp->dn_op == DT_TOK_PREDEC || dnp->dn_op == DT_TOK_POSTDEC)
2927                 idflags = DT_IDFLG_REF | DT_IDFLG_MOD;
2928         else
2929                 idflags = DT_IDFLG_REF;

2931         /*
2932          * We allow the unary ++ and -- operators to instantiate new scalar
2933          * variables if applied to an identifier; otherwise just cook as usual.
2934          */
2935         if (cp->dn_kind == DT_NODE_IDENT && (idflags & DT_IDFLG_MOD))
2936                 dt_xcook_ident(cp, dtp->dt_globals, DT_IDENT_SCALAR, B_TRUE);

2938         cp = dnp->dn_child = dt_node_cook(cp, 0); /* don't set idflags yet */

2940         if (cp->dn_kind == DT_NODE_VAR && dt_ident_unref(cp->dn_ident)) {
2941                 if (dt_type_lookup("int64_t", &dtt) != 0)
2942                         xyerror(D_TYPE_ERR, "failed to lookup int64_t\n");

2944                 dt_ident_type_assign(cp->dn_ident, dtt.dtt_ctfp, dtt.dtt_type);
2945                 dt_node_type_assign(cp, dtt.dtt_ctfp, dtt.dtt_type,
2946                     dtt.dtt_flags & DTT_FL_USER ? B_TRUE : B_FALSE);
2910                     dtt.dtt_flags);
2947         }

2949         if (cp->dn_kind == DT_NODE_VAR)
2950                 cp->dn_ident->di_flags |= idflags;

2952         switch (dnp->dn_op) {
2953         case DT_TOK_DEREF:
2954                 /*
2955                  * If the deref operator is applied to a translated pointer,
2956                  * we set our output type to the output of the translation.
2957                  */
2958                 if ((idp = dt_node_resolve(cp, DT_IDENT_XLPTR)) != NULL) {
2959                         dt_xlator_t *dxp = idp->di_data;

2961                         dnp->dn_ident = &dxp->dx_souid;
2962                         dt_node_type_assign(dnp,
2963                             dnp->dn_ident->di_ctfp, dnp->dn_ident->di_type,
2964                             cp->dn_flags & DT_NF_USERLAND);
2965                         break;
2966                 }

2968                 type = ctf_type_resolve(cp->dn_ctfp, cp->dn_type);
2969                 kind = ctf_type_kind(cp->dn_ctfp, type);

2971                 if (kind == CTF_K_ARRAY) {
2972                         if (ctf_array_info(cp->dn_ctfp, type, &r) != 0) {
2973                                 dtp->dt_ctferr = ctf_errno(cp->dn_ctfp);
2974                                 longjmp(yypcb->pcb_jmpbuf, EDT_CTF);
2975                         } else
2976                                 type = r.ctr_contents;
2977                 } else if (kind == CTF_K_POINTER) {
2978                         type = ctf_type_reference(cp->dn_ctfp, type);
2979                 } else {
2980                         xyerror(D_DEREF_NONPTR,
2981                             "cannot dereference non-pointer type\n");
2982                 }

2984                 dt_node_type_assign(dnp, cp->dn_ctfp, type,
```



```
2985                     cp->dn_flags & DT_NF_USERLAND);
2986                 base = ctf_type_resolve(cp->dn_ctfp, type);
2987                 kind = ctf_type_kind(cp->dn_ctfp, base);

2989                 if (kind == CTF_K_INTEGER && ctf_type_encoding(cp->dn_ctfp,
2990                     base, &e) == 0 && IS_VOID(e)) {
2991                         xyerror(D_DEREF_VOID,
2992                             "cannot dereference pointer to void\n");
2993                 }

2995                 if (kind == CTF_K_FUNCTION) {
2996                         xyerror(D_DEREF_FUNC,
2997                             "cannot dereference pointer to function\n");
2998                 }

3000                 if (kind != CTF_K_ARRAY || dt_node_is_string(dnp))
3001                         dnp->dn_flags |= DT_NF_LVALUE; /* see K&R[A7.4.3] */

3003                 /*
3004                  * If we propagated the l-value bit and the child operand was
3005                  * a writable D variable or a binary operation of the form
3006                  * a + b where a is writable, then propagate the writable bit.
3007                  * This is necessary to permit assignments to scalar arrays,
3008                  * which are converted to expressions of the form *(a + i).
3009                  */
3010                 if ((cp->dn_flags & DT_NF_WRITABLE) ||
3011                     (cp->dn_kind == DT_NODE_OP2 && cp->dn_op == DT_TOK_ADD &&
3012                     (cp->dn_left->dn_flags & DT_NF_WRITABLE)))
3013                         dnp->dn_flags |= DT_NF_WRITABLE;

3015                 if ((cp->dn_flags & DT_NF_USERLAND) &&
3016                     (kind == CTF_K_POINTER || (dnp->dn_flags & DT_NF_REF)))
3017                         dnp->dn_flags |= DT_NF_USERLAND;
3018                 break;

3020         case DT_TOK_IPOS:
3021         case DT_TOK_INEG:
3022                 if (!dt_node_is_arith(cp)) {
3023                         xyerror(D_OP_ARITH, "operator %s requires an operand "
3024                             "of arithmetic type\n", opstr(dnp->dn_op));
3025                 }
3026                 dt_node_type_propagate(cp, dnp); /* see K&R[A7.4.4-6] */
3027                 break;

3029         case DT_TOK_BNEG:
3030                 if (!dt_node_is_integer(cp)) {
3031                         xyerror(D_OP_INT, "operator %s requires an operand of "
3032                             "integral type\n", opstr(dnp->dn_op));
3033                 }
3034                 dt_node_type_propagate(cp, dnp); /* see K&R[A7.4.4-6] */
3035                 break;

3037         case DT_TOK_LNEG:
3038                 if (!dt_node_is_scalar(cp)) {
3039                         xyerror(D_OP_SCALAR, "operator %s requires an operand "
3040                             "of scalar type\n", opstr(dnp->dn_op));
3041                 }
3042                 dt_node_type_assign(dnp, DT_INT_CTFP(dtp), DT_INT_TYPE(dtp),
3043                     B_FALSE);
3044                 break;

3046         case DT_TOK_ADDROF:
3047                 if (cp->dn_kind == DT_NODE_VAR || cp->dn_kind == DT_NODE_AGG) {
3048                         xyerror(D_ADDROF_VAR,
3049                             "cannot take address of dynamic variable\n");
3050                 }
```

```
3052                 if (dt_node_is_dynamic(cp)) {
3053                         xyerror(D_ADDROF_VAR,
3054                             "cannot take address of dynamic object\n");
3055                 }

3057                 if (!(cp->dn_flags & DT_NF_LVALUE)) {
3058                         xyerror(D_ADDROF_LVAL, /* see K&R[A7.4.2] */
3059                             "unacceptable operand for unary & operator\n");
3060                 }

3062                 if (cp->dn_flags & DT_NF_BITFIELD) {
3063                         xyerror(D_ADDROF_BITFIELD,
3064                             "cannot take address of bit-field\n");
3065                 }

3067                 dtt.dtt_object = NULL;
3068                 dtt.dtt_ctfp = cp->dn_ctfp;
3069                 dtt.dtt_type = cp->dn_type;

3071                 if (dt_type_pointer(&dtt) == -1) {
3072                         xyerror(D_TYPE_ERR, "cannot find type for \"&\": %s*\n",
3073                             dt_node_type_name(cp, n, sizeof (n)));
3074                 }

3076                 dt_node_type_assign(dnp, dtt.dtt_ctfp, dtt.dtt_type,
3077                     cp->dn_flags & DT_NF_USERLAND);
3078                 break;

3080         case DT_TOK_SIZEOF:
3081                 if (cp->dn_flags & DT_NF_BITFIELD) {
3082                         xyerror(D_SIZEOF_BITFIELD,
3083                             "cannot apply sizeof to a bit-field\n");
3084                 }

3086                 if (dt_node_sizeof(cp) == 0) {
3087                         xyerror(D_SIZEOF_TYPE, "cannot apply sizeof to an "
3088                             "operand of unknown size\n");
3089                 }

3091                 dt_node_type_assign(dnp, dtp->dt_ddefs->dm_ctfp,
3092                     ctf_lookup_by_name(dtp->dt_ddefs->dm_ctfp, "size_t"),
3093                     B_FALSE);
3094                 break;

3096         case DT_TOK_STRINGOF:
3097                 if (!dt_node_is_scalar(cp) && !dt_node_is_pointer(cp) &&
3098                     !dt_node_is_strcompat(cp)) {
3099                         xyerror(D_STRINGOF_TYPE,
3100                             "cannot apply stringof to a value of type %s\n",
3101                             dt_node_type_name(cp, n, sizeof (n)));
3102                 }
3103                 dt_node_type_assign(dnp, DT_STR_CTFP(dtp), DT_STR_TYPE(dtp),
3104                     cp->dn_flags & DT_NF_USERLAND);
3105                 break;

3107         case DT_TOK_PREINC:
3108         case DT_TOK_POSTINC:
3109         case DT_TOK_PREDEC:
3110         case DT_TOK_POSTDEC:
3111                 if (dt_node_is_scalar(cp) == 0) {
3112                         xyerror(D_OP_SCALAR, "operator %s requires operand of "
3113                             "scalar type\n", opstr(dnp->dn_op));
3114                 }

3116                 if (dt_node_is_vfptr(cp)) {
```



```
3117                         xyerror(D_OP_VFPTR, "operator %s requires an operand "
3118                             "of known size\n", opstr(dnp->dn_op));
3119                 }

3121                 if (!(cp->dn_flags & DT_NF_LVALUE)) {
3122                         xyerror(D_OP_LVAL, "operator %s requires modifiable "
3123                             "lvalue as an operand\n", opstr(dnp->dn_op));
3124                 }

3126                 if (!(cp->dn_flags & DT_NF_WRITABLE)) {
3127                         xyerror(D_OP_WRITE, "operator %s can only be applied "
3128                             "to a writable variable\n", opstr(dnp->dn_op));
3129                 }

3131                 dt_node_type_propagate(cp, dnp); /* see K&R[A7.4.1] */
3132                 break;

3134         default:
3135                 xyerror(D_UNKNOWN, "invalid unary op %s\n", opstr(dnp->dn_op));
3136         }

3138         dt_node_attr_assign(dnp, cp->dn_attr);
3139         return (dnp);
3140 }
______unchanged_portion_omitted_

3167 static dt_node_t *
3168 dt_cook_op2(dt_node_t *dnp, uint_t idflags)
3169 {
3170         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
3171         dt_node_t *lp = dnp->dn_left;
3172         dt_node_t *rp = dnp->dn_right;
3173         int op = dnp->dn_op;

3175         ctf_membinfo_t m;
3176         ctf_file_t *ctfp;
3177         ctf_id_t type;
3178         int kind, val, uref;
3179         dt_ident_t *idp;

3181         char n1[DT_TYPE_NAMELEN];
3182         char n2[DT_TYPE_NAMELEN];

3184         /*
3185          * The expression E1[E2] is identical by definition to *((E1)+(E2)) so
3186          * we convert "[" to "+" and glue on "*" at the end (see K&R[A7.3.1])
3187          * unless the left-hand side is an untyped D scalar, associative array,
3188          * or aggregation.  In these cases, we proceed to case DT_TOK_LBRAC and
3189          * handle associative array and aggregation references there.
3190          */
3191         if (op == DT_TOK_LBRAC) {
3192                 if (lp->dn_kind == DT_NODE_IDENT) {
3193                         dt_idhash_t *dhp;
3194                         uint_t idkind;

3196                         if (lp->dn_op == DT_TOK_AGG) {
3197                                 dhp = dtp->dt_aggs;
3198                                 idp = dt_idhash_lookup(dhp, lp->dn_string + 1);
3199                                 idkind = DT_IDENT_AGG;
3200                         } else {
3201                                 dhp = dtp->dt_globals;
3202                                 idp = dt_idstack_lookup(
3203                                     &yypcb->pcb_globals, lp->dn_string);
3204                                 idkind = DT_IDENT_ARRAY;
3205                         }
```

```
3207                        if (idp == NULL || dt_ident_unref(idp))
3208                                dt_xcook_ident(lp, dhp, idkind, B_TRUE);
3209                        else
3210                                dt_xcook_ident(lp, dhp, idp->di_kind, B_FALSE);
3211                } else
3212                        lp = dnp->dn_left = dt_node_cook(lp, 0);

3214                /*
3215                 * Switch op to '+' for *(E1 + E2) array mode in these cases:
3216                 * (a) lp is a DT_IDENT_ARRAY variable that has already been
3217                 *     referenced using [] notation (dn_args != NULL).
3218                 * (b) lp is a non-ARRAY variable that has already been given
3219                 *     a type by assignment or declaration (!dt_ident_unref())
3220                 * (c) lp is neither a variable nor an aggregation
3221                 */
3222                if (lp->dn_kind == DT_NODE_VAR) {
3223                        if (lp->dn_ident->di_kind == DT_IDENT_ARRAY) {
3224                                if (lp->dn_args != NULL)
3225                                        op = DT_TOK_ADD;
3226                        } else if (!dt_ident_unref(lp->dn_ident))
3227                                op = DT_TOK_ADD;
3228                } else if (lp->dn_kind != DT_NODE_AGG)
3229                        op = DT_TOK_ADD;
3230        }

3232        switch (op) {
3233        case DT_TOK_BAND:
3234        case DT_TOK_XOR:
3235        case DT_TOK_BOR:
3236                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3237                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3239                if (!dt_node_is_integer(lp) || !dt_node_is_integer(rp)) {
3240                        xyerror(D_OP_INT, "operator %s requires operands of "
3241                            "integral type\n", opstr(op));
3242                }

3244                dt_node_promote(lp, rp, dnp); /* see K&R[A7.11-13] */
3245                break;

3247        case DT_TOK_LSH:
3248        case DT_TOK_RSH:
3249                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3250                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3252                if (!dt_node_is_integer(lp) || !dt_node_is_integer(rp)) {
3253                        xyerror(D_OP_INT, "operator %s requires operands of "
3254                            "integral type\n", opstr(op));
3255                }

3257                dt_node_type_propagate(lp, dnp); /* see K&R[A7.8] */
3258                dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));
3259                break;

3261        case DT_TOK_MOD:
3262                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3263                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3265                if (!dt_node_is_integer(lp) || !dt_node_is_integer(rp)) {
3266                        xyerror(D_OP_INT, "operator %s requires operands of "
3267                            "integral type\n", opstr(op));
3268                }

3270                dt_node_promote(lp, rp, dnp); /* see K&R[A7.6] */
3271                break;
```



```
3273        case DT_TOK_MUL:
3274        case DT_TOK_DIV:
3275                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3276                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3278                if (!dt_node_is_arith(lp) || !dt_node_is_arith(rp)) {
3279                        xyerror(D_OP_ARITH, "operator %s requires operands of "
3280                            "arithmetic type\n", opstr(op));
3281                }

3283                dt_node_promote(lp, rp, dnp); /* see K&R[A7.6] */
3284                break;

3286        case DT_TOK_LAND:
3287        case DT_TOK_LXOR:
3288        case DT_TOK_LOR:
3289                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3290                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3292                if (!dt_node_is_scalar(lp) || !dt_node_is_scalar(rp)) {
3293                        xyerror(D_OP_SCALAR, "operator %s requires operands "
3294                            "of scalar type\n", opstr(op));
3295                }

3297                dt_node_type_assign(dnp, DT_INT_CTFP(dtp), DT_INT_TYPE(dtp),
3298                    B_FALSE);
3299                dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));
3300                break;

3302        case DT_TOK_LT:
3303        case DT_TOK_LE:
3304        case DT_TOK_GT:
3305        case DT_TOK_GE:
3306        case DT_TOK_EQU:
3307        case DT_TOK_NEQ:
3308                /*
3309                 * The D comparison operators provide the ability to transform
3310                 * a right-hand identifier into a corresponding enum tag value
3311                 * if the left-hand side is an enum type.  To do this, we cook
3312                 * the left-hand side, and then see if the right-hand side is
3313                 * an unscoped identifier defined in the enum.  If so, we
3314                 * convert into an integer constant node with the tag's value.
3315                 */
3316                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);

3318                kind = ctf_type_kind(lp->dn_ctfp,
3319                    ctf_type_resolve(lp->dn_ctfp, lp->dn_type));

3321                if (kind == CTF_K_ENUM && rp->dn_kind == DT_NODE_IDENT &&
3322                    strchr(rp->dn_string, '`') == NULL && ctf_enum_value(
3323                    lp->dn_ctfp, lp->dn_type, rp->dn_string, &val) == 0) {

3325                        if ((idp = dt_idstack_lookup(&yypcb->pcb_globals,
3326                            rp->dn_string)) != NULL) {
3327                                xyerror(D_IDENT_AMBIG,
3328                                    "ambiguous use of operator %s: %s is "
3329                                    "both a %s enum tag and a global %s\n",
3330                                    opstr(op), rp->dn_string,
3331                                    dt_node_type_name(lp, n1, sizeof (n1)),
3332                                    dt_idkind_name(idp->di_kind));
3333                        }

3335                        free(rp->dn_string);
3336                        rp->dn_string = NULL;
3337                        rp->dn_kind = DT_NODE_INT;
3338                        rp->dn_flags |= DT_NF_COOKED;
```

```
3339                        rp->dn_op = DT_TOK_INT;
3340                        rp->dn_value = (intmax_t)val;

3342                        dt_node_type_assign(rp, lp->dn_ctfp, lp->dn_type,
3343                            B_FALSE);
3344                        dt_node_attr_assign(rp, _dtrace_symattr);
3345                }

3347                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3349                /*
3350                 * The rules for type checking for the relational operators are
3351                 * described in the ANSI-C spec (see K&R[A7.9-10]).  We perform
3352                 * the various tests in order from least to most expensive.  We
3353                 * also allow derived strings to be compared as a first-class
3354                 * type (resulting in a strcmp(3C)-style comparison), and we
3355                 * slightly relax the A7.9 rules to permit void pointer
3356                 * comparisons as in A7.10.  Our users won't be confused by
3357                 * this since they understand pointers are just numbers, and
3358                 * relaxing this constraint simplifies the implementation.
3359                 */
3360                if (ctf_type_compat(lp->dn_ctfp, lp->dn_type,
3361                    rp->dn_ctfp, rp->dn_type))
3362                        /*EMPTY*/;
3363                else if (dt_node_is_integer(lp) && dt_node_is_integer(rp))
3364                        /*EMPTY*/;
3365                else if (dt_node_is_strcompat(lp) && dt_node_is_strcompat(rp) &&
3366                    (dt_node_is_string(lp) || dt_node_is_string(rp)))
3367                        /*EMPTY*/;
3368                else if (dt_node_is_ptrcompat(lp, rp, NULL, NULL) == 0) {
3369                        xyerror(D_OP_INCOMPAT, "operands have "
3370                            "incompatible types: \"%s\" %s \"%s\"\n",
3371                            dt_node_type_name(lp, n1, sizeof (n1)), opstr(op),
3372                            dt_node_type_name(rp, n2, sizeof (n2)));
3373                }

3375                dt_node_type_assign(dnp, DT_INT_CTFP(dtp), DT_INT_TYPE(dtp),
3376                    B_FALSE);
3377                dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));
3378                break;

3380        case DT_TOK_ADD:
3381        case DT_TOK_SUB: {
3382                /*
3383                 * The rules for type checking for the additive operators are
3384                 * described in the ANSI-C spec (see K&R[A7.7]).  Pointers and
3385                 * integers may be manipulated according to specific rules.  In
3386                 * these cases D permits strings to be treated as pointers.
3387                 */
3388                int lp_is_ptr, lp_is_int, rp_is_ptr, rp_is_int;

3390                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3391                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3393                lp_is_ptr = dt_node_is_string(lp) ||
3394                    (dt_node_is_pointer(lp) && !dt_node_is_vfptr(lp));
3395                lp_is_int = dt_node_is_integer(lp);

3397                rp_is_ptr = dt_node_is_string(rp) ||
3398                    (dt_node_is_pointer(rp) && !dt_node_is_vfptr(rp));
3399                rp_is_int = dt_node_is_integer(rp);

3401                if (lp_is_int && rp_is_int) {
3402                        dt_type_promote(lp, rp, &ctfp, &type);
3403                        uref = 0;
3404                } else if (lp_is_ptr && rp_is_int) {
```



```
3405                        ctfp = lp->dn_ctfp;
3406                        type = lp->dn_type;
3407                        uref = lp->dn_flags & DT_NF_USERLAND;
3408                } else if (lp_is_int && rp_is_ptr && op == DT_TOK_ADD) {
3409                        ctfp = rp->dn_ctfp;
3410                        type = rp->dn_type;
3411                        uref = rp->dn_flags & DT_NF_USERLAND;
3412                } else if (lp_is_ptr && rp_is_ptr && op == DT_TOK_SUB &&
3413                    dt_node_is_ptrcompat(lp, rp, NULL, NULL)) {
3414                        ctfp = dtp->dt_ddefs->dm_ctfp;
3415                        type = ctf_lookup_by_name(ctfp, "ptrdiff_t");
3416                        uref = 0;
3417                } else {
3418                        xyerror(D_OP_INCOMPAT, "operands have incompatible "
3419                            "types: \"%s\" %s \"%s\"\n",
3420                            dt_node_type_name(lp, n1, sizeof (n1)), opstr(op),
3421                            dt_node_type_name(rp, n2, sizeof (n2)));
3422                }

3424                dt_node_type_assign(dnp, ctfp, type, B_FALSE);
3425                dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));

3427                if (uref)
3428                        dnp->dn_flags |= DT_NF_USERLAND;
3429                break;
3430        }

3432        case DT_TOK_OR_EQ:
3433        case DT_TOK_XOR_EQ:
3434        case DT_TOK_AND_EQ:
3435        case DT_TOK_LSH_EQ:
3436        case DT_TOK_RSH_EQ:
3437        case DT_TOK_MOD_EQ:
3438                if (lp->dn_kind == DT_NODE_IDENT) {
3439                        dt_xcook_ident(lp, dtp->dt_globals,
3440                            DT_IDENT_SCALAR, B_TRUE);
3441                }

3443                lp = dnp->dn_left =
3444                    dt_node_cook(lp, DT_IDFLG_REF | DT_IDFLG_MOD);

3446                rp = dnp->dn_right =
3447                    dt_node_cook(rp, DT_IDFLG_REF | DT_IDFLG_MOD);

3449                if (!dt_node_is_integer(lp) || !dt_node_is_integer(rp)) {
3450                        xyerror(D_OP_INT, "operator %s requires operands of "
3451                            "integral type\n", opstr(op));
3452                }
3453                goto asgn_common;

3455        case DT_TOK_MUL_EQ:
3456        case DT_TOK_DIV_EQ:
3457                if (lp->dn_kind == DT_NODE_IDENT) {
3458                        dt_xcook_ident(lp, dtp->dt_globals,
3459                            DT_IDENT_SCALAR, B_TRUE);
3460                }

3462                lp = dnp->dn_left =
3463                    dt_node_cook(lp, DT_IDFLG_REF | DT_IDFLG_MOD);

3465                rp = dnp->dn_right =
3466                    dt_node_cook(rp, DT_IDFLG_REF | DT_IDFLG_MOD);

3468                if (!dt_node_is_arith(lp) || !dt_node_is_arith(rp)) {
3469                        xyerror(D_OP_ARITH, "operator %s requires operands of "
3470                            "arithmetic type\n", opstr(op));
```

```
3471                }
3472                goto asgn_common;

3474        case DT_TOK_ASGN:
3475                /*
3476                 * If the left-hand side is an identifier, attempt to resolve
3477                 * it as either an aggregation or scalar variable.  We pass
3478                 * B_TRUE to dt_xcook_ident to indicate that a new variable can
3479                 * be created if no matching variable exists in the namespace.
3480                 */
3481                if (lp->dn_kind == DT_NODE_IDENT) {
3482                        if (lp->dn_op == DT_TOK_AGG) {
3483                                dt_xcook_ident(lp, dtp->dt_aggs,
3484                                    DT_IDENT_AGG, B_TRUE);
3485                        } else {
3486                                dt_xcook_ident(lp, dtp->dt_globals,
3487                                    DT_IDENT_SCALAR, B_TRUE);
3488                        }
3489                }

3491                lp = dnp->dn_left = dt_node_cook(lp, 0); /* don't set mod yet */
3492                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);

3494                /*
3495                 * If the left-hand side is an aggregation, verify that we are
3496                 * assigning it the result of an aggregating function.  Once
3497                 * we've done so, hide the func node in the aggregation and
3498                 * return the aggregation itself up to the parse tree parent.
3499                 * This transformation is legal since the assigned function
3500                 * cannot change identity across disjoint cooking passes and
3501                 * the argument list subtree is retained for later cooking.
3502                 */
3503                if (lp->dn_kind == DT_NODE_AGG) {
3504                        const char *aname = lp->dn_ident->di_name;
3505                        dt_ident_t *oid = lp->dn_ident->di_iarg;

3507                        if (rp->dn_kind != DT_NODE_FUNC ||
3508                            rp->dn_ident->di_kind != DT_IDENT_AGGFUNC) {
3509                                xyerror(D_AGG_FUNC,
3510                                    "@%s must be assigned the result of "
3511                                    "an aggregating function\n", aname);
3512                        }

3514                        if (oid != NULL && oid != rp->dn_ident) {
3515                                xyerror(D_AGG_REDEF,
3516                                    "aggregation redefined: @%s\n\t "
3517                                    "current: @%s = %s( )\n\tprevious: @%s = "
3518                                    "%s( ) : line %d\n", aname, aname,
3519                                    rp->dn_ident->di_name, aname, oid->di_name,
3520                                    lp->dn_ident->di_lineno);
3521                        } else if (oid == NULL)
3522                                lp->dn_ident->di_iarg = rp->dn_ident;

3524                        /*
3525                         * Do not allow multiple aggregation assignments in a
3526                         * single statement, e.g. (@a = count()) = count();
3527                         * We produce a message as if the result of aggregating
3528                         * function does not propagate DT_NF_LVALUE.
3529                         */
3530                        if (lp->dn_aggfun != NULL) {
3531                                xyerror(D_OP_LVAL, "operator = requires "
3532                                    "modifiable lvalue as an operand\n");
3533                        }

3535                        lp->dn_aggfun = rp;
3536                        lp = dt_node_cook(lp, DT_IDFLG_MOD);
```



```
3538                        dnp->dn_left = dnp->dn_right = NULL;
3539                        dt_node_free(dnp);

3541                        return (lp);
3542                }

3544                /*
3545                 * If the right-hand side is a dynamic variable that is the
3546                 * output of a translator, our result is the translated type.
3547                 */
3548                if ((idp = dt_node_resolve(rp, DT_IDENT_XLSOU)) != NULL) {
3549                        ctfp = idp->di_ctfp;
3550                        type = idp->di_type;
3551                        uref = idp->di_flags & DT_IDFLG_USER;
3552                } else {
3553                        ctfp = rp->dn_ctfp;
3554                        type = rp->dn_type;
3555                        uref = rp->dn_flags & DT_NF_USERLAND;
3556                }

3558                /*
3559                 * If the left-hand side of an assignment statement is a virgin
3560                 * variable created by this compilation pass, reset the type of
3561                 * this variable to the type of the right-hand side.
3562                 */
3563                if (lp->dn_kind == DT_NODE_VAR &&
3564                    dt_ident_unref(lp->dn_ident)) {
3565                        dt_node_type_assign(lp, ctfp, type, B_FALSE);
3566                        dt_ident_type_assign(lp->dn_ident, ctfp, type);

3568                        if (uref) {
3569                                lp->dn_flags |= DT_NF_USERLAND;
3570                                lp->dn_ident->di_flags |= DT_IDFLG_USER;
3571                        }
3572                }

3574                if (lp->dn_kind == DT_NODE_VAR)
3575                        lp->dn_ident->di_flags |= DT_IDFLG_MOD;

3577                /*
3578                 * The rules for type checking for the assignment operators are
3579                 * described in the ANSI-C spec (see K&R[A7.17]).  We share
3580                 * most of this code with the argument list checking code.
3581                 */
3582                if (!dt_node_is_string(lp)) {
3583                        kind = ctf_type_kind(lp->dn_ctfp,
3584                            ctf_type_resolve(lp->dn_ctfp, lp->dn_type));

3586                        if (kind == CTF_K_ARRAY || kind == CTF_K_FUNCTION) {
3587                                xyerror(D_OP_ARRFUN, "operator %s may not be "
3588                                    "applied to operand of type \"%s\"\n",
3589                                    opstr(op),
3590                                    dt_node_type_name(lp, n1, sizeof (n1)));
3591                        }
3592                }

3594                if (idp != NULL && idp->di_kind == DT_IDENT_XLSOU &&
3595                    ctf_type_compat(lp->dn_ctfp, lp->dn_type, ctfp, type))
3596                        goto asgn_common;

3598                if (dt_node_is_argcompat(lp, rp))
3599                        goto asgn_common;

3601                xyerror(D_OP_INCOMPAT,
3602                    "operands have incompatible types: \"%s\" %s \"%s\"\n",
```

```
3603                    dt_node_type_name(lp, n1, sizeof (n1)), opstr(op),
3604                    dt_node_type_name(rp, n2, sizeof (n2)));
3605                /*NOTREACHED*/

3607        case DT_TOK_ADD_EQ:
3608        case DT_TOK_SUB_EQ:
3609                if (lp->dn_kind == DT_NODE_IDENT) {
3610                        dt_xcook_ident(lp, dtp->dt_globals,
3611                            DT_IDENT_SCALAR, B_TRUE);
3612                }

3614                lp = dnp->dn_left =
3615                    dt_node_cook(lp, DT_IDFLG_REF | DT_IDFLG_MOD);

3617                rp = dnp->dn_right =
3618                    dt_node_cook(rp, DT_IDFLG_REF | DT_IDFLG_MOD);

3620                if (dt_node_is_string(lp) || dt_node_is_string(rp)) {
3621                        xyerror(D_OP_INCOMPAT, "operands have "
3622                            "incompatible types: \"%s\" %s \"%s\"\n",
3623                            dt_node_type_name(lp, n1, sizeof (n1)), opstr(op),
3624                            dt_node_type_name(rp, n2, sizeof (n2)));
3625                }

3627                /*
3628                 * The rules for type checking for the assignment operators are
3629                 * described in the ANSI-C spec (see K&R[A7.17]).  To these
3630                 * rules we add that only writable D nodes can be modified.
3631                 */
3632                if (dt_node_is_integer(lp) == 0 ||
3633                    dt_node_is_integer(rp) == 0) {
3634                        if (!dt_node_is_pointer(lp) || dt_node_is_vfptr(lp)) {
3635                                xyerror(D_OP_VFPTR,
3636                                    "operator %s requires left-hand scalar "
3637                                    "operand of known size\n", opstr(op));
3638                        } else if (dt_node_is_integer(rp) == 0 &&
3639                            dt_node_is_ptrcompat(lp, rp, NULL, NULL) == 0) {
3640                                xyerror(D_OP_INCOMPAT, "operands have "
3641                                    "incompatible types: \"%s\" %s \"%s\"\n",
3642                                    dt_node_type_name(lp, n1, sizeof (n1)),
3643                                    opstr(op),
3644                                    dt_node_type_name(rp, n2, sizeof (n2)));
3645                        }
3646                }
3647 asgn_common:
3648                dt_assign_common(dnp);
3649                break;

3651        case DT_TOK_PTR:
3652                /*
3653                 * If the left-hand side of operator -> is the name "self",
3654                 * then we permit a TLS variable to be created or referenced.
3655                 */
3656                if (lp->dn_kind == DT_NODE_IDENT &&
3657                    strcmp(lp->dn_string, "self") == 0) {
3658                        if (rp->dn_kind != DT_NODE_VAR) {
3659                                dt_xcook_ident(rp, dtp->dt_tls,
3660                                    DT_IDENT_SCALAR, B_TRUE);
3661                        }

3663                        if (idflags != 0)
3664                                rp = dt_node_cook(rp, idflags);

3666                        dnp->dn_right = dnp->dn_left; /* avoid freeing rp */
3667                        dt_node_free(dnp);
3668                        return (rp);
```



```
3669                }

3671                /*
3672                 * If the left-hand side of operator -> is the name "this",
3673                 * then we permit a local variable to be created or referenced.
3674                 */
3675                if (lp->dn_kind == DT_NODE_IDENT &&
3676                    strcmp(lp->dn_string, "this") == 0) {
3677                        if (rp->dn_kind != DT_NODE_VAR) {
3678                                dt_xcook_ident(rp, yypcb->pcb_locals,
3679                                    DT_IDENT_SCALAR, B_TRUE);
3680                        }

3682                        if (idflags != 0)
3683                                rp = dt_node_cook(rp, idflags);

3685                        dnp->dn_right = dnp->dn_left; /* avoid freeing rp */
3686                        dt_node_free(dnp);
3687                        return (rp);
3688                }

3690                /*FALLTHRU*/

3692        case DT_TOK_DOT:
3693                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);

3695                if (rp->dn_kind != DT_NODE_IDENT) {
3696                        xyerror(D_OP_IDENT, "operator %s must be followed by "
3697                            "an identifier\n", opstr(op));
3698                }

3700                if ((idp = dt_node_resolve(lp, DT_IDENT_XLSOU)) != NULL ||
3701                    (idp = dt_node_resolve(lp, DT_IDENT_XLPTR)) != NULL) {
3702                        /*
3703                         * If the left-hand side is a translated struct or ptr,
3704                         * the type of the left is the translation output type.
3705                         */
3706                        dt_xlator_t *dxp = idp->di_data;

3708                        if (dt_xlator_member(dxp, rp->dn_string) == NULL) {
3709                                xyerror(D_XLATE_NOCONV,
3710                                    "translator does not define conversion "
3711                                    "for member: %s\n", rp->dn_string);
3712                        }

3714                        ctfp = idp->di_ctfp;
3715                        type = ctf_type_resolve(ctfp, idp->di_type);
3716                        uref = idp->di_flags & DT_IDFLG_USER;
3717                } else {
3718                        ctfp = lp->dn_ctfp;
3719                        type = ctf_type_resolve(ctfp, lp->dn_type);
3720                        uref = lp->dn_flags & DT_NF_USERLAND;
3721                }

3723                kind = ctf_type_kind(ctfp, type);

3725                if (op == DT_TOK_PTR) {
3726                        if (kind != CTF_K_POINTER) {
3727                                xyerror(D_OP_PTR, "operator %s must be "
3728                                    "applied to a pointer\n", opstr(op));
3729                        }
3730                        type = ctf_type_reference(ctfp, type);
3731                        type = ctf_type_resolve(ctfp, type);
3732                        kind = ctf_type_kind(ctfp, type);
3733                }
```

```
3735                 /*
3736                  * If we follow a reference to a forward declaration tag,
3737                  * search the entire type space for the actual definition.
3738                  */
3739                 dt_resolve_forward_decl(&ctfp, &type);
3740                 kind = ctf_type_kind(ctfp, type);
3703                 while (kind == CTF_K_FORWARD) {
3704                         char *tag = ctf_type_name(ctfp, type, n1, sizeof (n1));
3705                         dtrace_typeinfo_t dtt;

3742                 if (kind == CTF_K_FORWARD) {
3707                         if (tag != NULL && dt_type_lookup(tag, &dtt) == 0 &&
3708                             (dtt.dtt_ctfp != ctfp || dtt.dtt_type != type)) {
3709                                 ctfp = dtt.dtt_ctfp;
3710                                 type = ctf_type_resolve(ctfp, dtt.dtt_type);
3711                                 kind = ctf_type_kind(ctfp, type);
3712                         } else {
3743                         xyerror(D_OP_INCOMPLETE,
3744                             "operator %s cannot be applied to a "
3745                             "forward declaration: no %s definition "
3746                             "is available\n", opstr(op),
3747                             ctf_type_name(ctfp, type, n1, sizeof (n1)));
3748                 } else if (kind != CTF_K_STRUCT && kind != CTF_K_UNION) {
3716                                 "is available\n", opstr(op), tag);
3717                         }
3718                 }

3720                 if (kind != CTF_K_STRUCT && kind != CTF_K_UNION) {
3749                         if (op == DT_TOK_PTR) {
3750                                 xyerror(D_OP_SOU, "operator -> cannot be "
3751                                     "applied to pointer to type \"%s\"; must "
3752                                     "be applied to a struct or union pointer\n",
3753                                     ctf_type_name(ctfp, type, n1, sizeof (n1)));
3754                         } else {
3755                                 xyerror(D_OP_SOU, "operator %s cannot be "
3756                                     "applied to type \"%s\"; must be applied "
3757                                     "to a struct or union\n", opstr(op),
3758                                     ctf_type_name(ctfp, type, n1, sizeof (n1)));
3759                         }
3760                 }

3762                 if (ctf_member_info(ctfp, type, rp->dn_string, &m) == CTF_ERR) {
3763                         xyerror(D_TYPE_MEMBER,
3764                             "%s is not a member of %s\n", rp->dn_string,
3765                             ctf_type_name(ctfp, type, n1, sizeof (n1)));
3766                 }

3768                 type = m.ctm_type;
3740                 type = ctf_type_resolve(ctfp, m.ctm_type);
3741                 kind = ctf_type_kind(ctfp, type);

3770                 dt_resolve_forward_decl(&ctfp, &type);
3771                 dt_node_type_assign(dnp, ctfp, type, B_FALSE);
3743                 dt_node_type_assign(dnp, ctfp, m.ctm_type, B_FALSE);
3772                 dt_node_attr_assign(dnp, lp->dn_attr);

3774                 type = ctf_type_resolve(ctfp, type);
3775                 kind = ctf_type_kind(ctfp, type);

3777 #endif /* ! codereview */
3778                 if (op == DT_TOK_PTR && (kind != CTF_K_ARRAY ||
3779                     dt_node_is_string(dnp)))
3780                         dnp->dn_flags |= DT_NF_LVALUE; /* see K&R[A7.3.3] */

3782                 if (op == DT_TOK_DOT && (lp->dn_flags & DT_NF_LVALUE) &&
3783                     (kind != CTF_K_ARRAY || dt_node_is_string(dnp)))
```



```
3784                         dnp->dn_flags |= DT_NF_LVALUE; /* see K&R[A7.3.3] */

3786                 if (lp->dn_flags & DT_NF_WRITABLE)
3787                         dnp->dn_flags |= DT_NF_WRITABLE;

3789                 if (uref && (kind == CTF_K_POINTER ||
3790                     (dnp->dn_flags & DT_NF_REF)))
3791                         dnp->dn_flags |= DT_NF_USERLAND;
3792                 break;

3794         case DT_TOK_LBRAC: {
3795                 /*
3796                  * If op is DT_TOK_LBRAC, we know from the special-case code at
3797                  * the top that lp is either a D variable or an aggregation.
3798                  */
3799                 dt_node_t *lnp;

3801                 /*
3802                  * If the left-hand side is an aggregation, just set dn_aggtup
3803                  * to the right-hand side and return the cooked aggregation.
3804                  * This transformation is legal since we are just collapsing
3805                  * nodes to simplify later processing, and the entire aggtup
3806                  * parse subtree is retained for subsequent cooking passes.
3807                  */
3808                 if (lp->dn_kind == DT_NODE_AGG) {
3809                         if (lp->dn_aggtup != NULL) {
3810                                 xyerror(D_AGG_MDIM, "improper attempt to "
3811                                     "reference @%s as a multi-dimensional "
3812                                     "array\n", lp->dn_ident->di_name);
3813                         }

3815                         lp->dn_aggtup = rp;
3816                         lp = dt_node_cook(lp, 0);

3818                         dnp->dn_left = dnp->dn_right = NULL;
3819                         dt_node_free(dnp);

3821                         return (lp);
3822                 }

3824                 assert(lp->dn_kind == DT_NODE_VAR);
3825                 idp = lp->dn_ident;

3827                 /*
3828                  * If the left-hand side is a non-global scalar that hasn't yet
3829                  * been referenced or modified, it was just created by self->
3830                  * or this-> and we can convert it from scalar to assoc array.
3831                  */
3832                 if (idp->di_kind == DT_IDENT_SCALAR && dt_ident_unref(idp) &&
3833                     (idp->di_flags & (DT_IDFLG_LOCAL | DT_IDFLG_TLS)) != 0) {

3835                         if (idp->di_flags & DT_IDFLG_LOCAL) {
3836                                 xyerror(D_ARR_LOCAL,
3837                                     "local variables may not be used as "
3838                                     "associative arrays: %s\n", idp->di_name);
3839                         }

3841                         dt_dprintf("morph variable %s (id %u) from scalar to "
3842                             "array\n", idp->di_name, idp->di_id);

3844                         dt_ident_morph(idp, DT_IDENT_ARRAY,
3845                             &dt_idops_assc, NULL);
3846                 }

3848                 if (idp->di_kind != DT_IDENT_ARRAY) {
3849                         xyerror(D_IDENT_BADREF, "%s '%s' may not be referenced "
```

```
3850                            "as %s\n", dt_idkind_name(idp->di_kind),
3851                            idp->di_name, dt_idkind_name(DT_IDENT_ARRAY));
3852                }
3853
3854                /*
3855                 * Now that we've confirmed our left-hand side is a DT_NODE_VAR
3856                 * of idkind DT_IDENT_ARRAY, we need to splice the [ node from
3857                 * the parse tree and leave a cooked DT_NODE_VAR in its place
3858                 * where dn_args for the VAR node is the right-hand 'rp' tree,
3859                 * as shown in the parse tree diagram below:
3860                 *
3861                 *       /                       /
3862                 * [ OP2 "[" ]=dnp             [ VAR ]=dnp
3863                 *      /     \         =>       |
3864                 *     /       \                 +- dn_args -> [ ??? ]=rp
3865                 * [ VAR ]=lp  [ ??? ]=rp
3866                 *
3867                 * Since the final dt_node_cook(dnp) can fail using longjmp we
3868                 * must perform the transformations as a group first by over-
3869                 * writing 'dnp' to become the VAR node, so that the parse tree
3870                 * is guaranteed to be in a consistent state if the cook fails.
3871                 */
3872                assert(lp->dn_kind == DT_NODE_VAR);
3873                assert(lp->dn_args == NULL);
3874
3875                lnp = dnp->dn_link;
3876                bcopy(lp, dnp, sizeof (dt_node_t));
3877                dnp->dn_link = lnp;
3878
3879                dnp->dn_args = rp;
3880                dnp->dn_list = NULL;
3881
3882                dt_node_free(lp);
3883                return (dt_node_cook(dnp, idflags));
3884        }
3885
3886        case DT_TOK_XLATE: {
3887                dt_xlator_t *dxp;
3888
3889                assert(lp->dn_kind == DT_NODE_TYPE);
3890                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);
3891                dxp = dt_xlator_lookup(dtp, rp, lp, DT_XLATE_FUZZY);
3892
3893                if (dxp == NULL) {
3894                        xyerror(D_XLATE_NONE,
3895                            "cannot translate from \"%s\" to \"%s\"\n",
3896                            dt_node_type_name(rp, n1, sizeof (n1)),
3897                            dt_node_type_name(lp, n2, sizeof (n2)));
3898                }
3899
3900                dnp->dn_ident = dt_xlator_ident(dxp, lp->dn_ctfp, lp->dn_type);
3901                dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp),
3902                    B_FALSE);
3903                dt_node_attr_assign(dnp,
3904                    dt_attr_min(rp->dn_attr, dnp->dn_ident->di_attr));
3905                break;
3906        }
3907
3908        case DT_TOK_LPAR: {
3909                ctf_id_t ltype, rtype;
3910                uint_t lkind, rkind;
3911
3912                assert(lp->dn_kind == DT_NODE_TYPE);
3913                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);
3914
3915                ltype = ctf_type_resolve(lp->dn_ctfp, lp->dn_type);
```



```
3916                lkind = ctf_type_kind(lp->dn_ctfp, ltype);
3917
3918                rtype = ctf_type_resolve(rp->dn_ctfp, rp->dn_type);
3919                rkind = ctf_type_kind(rp->dn_ctfp, rtype);
3920
3921                /*
3922                 * The rules for casting are loosely explained in K&R[A7.5]
3923                 * and K&R[A6].  Basically, we can cast to the same type or
3924                 * same base type, between any kind of scalar values, from
3925                 * arrays to pointers, and we can cast anything to void.
3926                 * To these rules D adds casts from scalars to strings.
3927                 */
3928                if (ctf_type_compat(lp->dn_ctfp, lp->dn_type,
3929                    rp->dn_ctfp, rp->dn_type))
3930                        /*EMPTY*/;
3931                else if (dt_node_is_scalar(lp) &&
3932                    (dt_node_is_scalar(rp) || rkind == CTF_K_FUNCTION))
3933                        /*EMPTY*/;
3934                else if (dt_node_is_void(lp))
3935                        /*EMPTY*/;
3936                else if (lkind == CTF_K_POINTER && dt_node_is_pointer(rp))
3937                        /*EMPTY*/;
3938                else if (dt_node_is_string(lp) && (dt_node_is_scalar(rp) ||
3939                    dt_node_is_pointer(rp) || dt_node_is_strcompat(rp)))
3940                        /*EMPTY*/;
3941                else {
3942                        xyerror(D_CAST_INVAL,
3943                            "invalid cast expression: \"%s\" to \"%s\"\n",
3944                            dt_node_type_name(rp, n1, sizeof (n1)),
3945                            dt_node_type_name(lp, n2, sizeof (n2)));
3946                }
3947
3948                dt_node_type_propagate(lp, dnp); /* see K&R[A7.5] */
3949                dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));
3950
3951                /*
3952                 * If it's a pointer then should be able to (attempt to)
3953                 * assign to it.
3954                 */
3955                if (lkind == CTF_K_POINTER)
3956                        dnp->dn_flags |= DT_NF_WRITABLE;
3957
3958                break;
3959        }
3960
3961        case DT_TOK_COMMA:
3962                lp = dnp->dn_left = dt_node_cook(lp, DT_IDFLG_REF);
3963                rp = dnp->dn_right = dt_node_cook(rp, DT_IDFLG_REF);
3964
3965                if (dt_node_is_dynamic(lp) || dt_node_is_dynamic(rp)) {
3966                        xyerror(D_OP_DYN, "operator %s operands "
3967                            "cannot be of dynamic type\n", opstr(op));
3968                }
3969
3970                if (dt_node_is_actfunc(lp) || dt_node_is_actfunc(rp)) {
3971                        xyerror(D_OP_ACT, "operator %s operands "
3972                            "cannot be actions\n", opstr(op));
3973                }
3974
3975                dt_node_type_propagate(rp, dnp); /* see K&R[A7.18] */
3976                dt_node_attr_assign(dnp, dt_attr_min(lp->dn_attr, rp->dn_attr));
3977                break;
3978
3979        default:
3980                xyerror(D_UNKNOWN, "invalid binary op %s\n", opstr(op));
3981        }
```

```
3983         /*
3984          * Complete the conversion of E1[E2] to *((E1)+(E2)) that we started
3985          * at the top of our switch() above (see K&R[A7.3.1]).  Since E2 is
3986          * parsed as an argument_expression_list by dt_grammar.y, we can
3987          * end up with a comma-separated list inside of a non-associative
3988          * array reference.  We check for this and report an appropriate error.
3989          */
3990         if (dnp->dn_op == DT_TOK_LBRAC && op == DT_TOK_ADD) {
3991                 dt_node_t *pnp;
3992
3993                 if (rp->dn_list != NULL) {
3994                         xyerror(D_ARR_BADREF,
3995                             "cannot access %s as an associative array\n",
3996                             dt_node_name(lp, n1, sizeof (n1)));
3997                 }
3998
3999                 dnp->dn_op = DT_TOK_ADD;
4000                 pnp = dt_node_op1(DT_TOK_DEREF, dnp);
4001
4002                 /*
4003                  * Cook callbacks are not typically permitted to allocate nodes.
4004                  * When we do, we must insert them in the middle of an existing
4005                  * allocation list rather than having them appended to the pcb
4006                  * list because the sub-expression may be part of a definition.
4007                  */
4008                 assert(yypcb->pcb_list == pnp);
4009                 yypcb->pcb_list = pnp->dn_link;
4010
4011                 pnp->dn_link = dnp->dn_link;
4012                 dnp->dn_link = pnp;
4013
4014                 return (dt_node_cook(pnp, DT_IDFLG_REF));
4015         }
4016
4017         return (dnp);
4018 }
4019
4020 /*ARGSUSED*/
4021 static dt_node_t *
4022 dt_cook_op3(dt_node_t *dnp, uint_t idflags)
4023 {
4024         dt_node_t *lp, *rp;
4025         ctf_file_t *ctfp;
4026         ctf_id_t type;
4027
4028         dnp->dn_expr = dt_node_cook(dnp->dn_expr, DT_IDFLG_REF);
4029         lp = dnp->dn_left = dt_node_cook(dnp->dn_left, DT_IDFLG_REF);
4030         rp = dnp->dn_right = dt_node_cook(dnp->dn_right, DT_IDFLG_REF);
4031
4032         if (!dt_node_is_scalar(dnp->dn_expr)) {
4033                 xyerror(D_OP_SCALAR,
4034                     "operator ?: expression must be of scalar type\n");
4035         }
4036
4037         if (dt_node_is_dynamic(lp) || dt_node_is_dynamic(rp)) {
4038                 xyerror(D_OP_DYN,
4039                     "operator ?: operands cannot be of dynamic type\n");
4040         }
4041
4042         /*
4043          * The rules for type checking for the ternary operator are complex and
4044          * are described in the ANSI-C spec (see K&R[A7.16]).  We implement
4045          * the various tests in order from least to most expensive.
4046          */
4047         if (ctf_type_compat(lp->dn_ctfp, lp->dn_type,
```



```
4048             rp->dn_ctfp, rp->dn_type)) {
4049                 ctfp = lp->dn_ctfp;
4050                 type = lp->dn_type;
4051         } else if (dt_node_is_integer(lp) && dt_node_is_integer(rp)) {
4052                 dt_type_promote(lp, rp, &ctfp, &type);
4053         } else if (dt_node_is_strcompat(lp) && dt_node_is_strcompat(rp) &&
4054             (dt_node_is_string(lp) || dt_node_is_string(rp))) {
4055                 ctfp = DT_STR_CTFP(yypcb->pcb_hdl);
4056                 type = DT_STR_TYPE(yypcb->pcb_hdl);
4057         } else if (dt_node_is_ptrcompat(lp, rp, &ctfp, &type) == 0) {
4058                 xyerror(D_OP_INCOMPAT,
4059                     "operator ?: operands must have compatible types\n");
4060         }
4061
4062         if (dt_node_is_actfunc(lp) || dt_node_is_actfunc(rp)) {
4063                 xyerror(D_OP_ACT, "action cannot be "
4064                     "used in a conditional context\n");
4065         }
4066
4067         dt_node_type_assign(dnp, ctfp, type, B_FALSE);
4068         dt_node_attr_assign(dnp, dt_attr_min(dnp->dn_expr->dn_attr,
4069             dt_attr_min(lp->dn_attr, rp->dn_attr)));
4070
4071         return (dnp);
4072 }
4073
4074 static dt_node_t *
4075 dt_cook_statement(dt_node_t *dnp, uint_t idflags)
4076 {
4077         dnp->dn_expr = dt_node_cook(dnp->dn_expr, idflags);
4078         dt_node_attr_assign(dnp, dnp->dn_expr->dn_attr);
4079
4080         return (dnp);
4081 }
4082
4083 /*
4084  * If dn_aggfun is set, this node is a collapsed aggregation assignment (see
4085  * the special case code for DT_TOK_ASGN in dt_cook_op2() above), in which
4086  * case we cook both the tuple and the function call.  If dn_aggfun is NULL,
4087  * this node is just a reference to the aggregation's type and attributes.
4088  */
4089 /*ARGSUSED*/
4090 static dt_node_t *
4091 dt_cook_aggregation(dt_node_t *dnp, uint_t idflags)
4092 {
4093         dtrace_hdl_t *dtp = yypcb->pcb_hdl;
4094
4095         if (dnp->dn_aggfun != NULL) {
4096                 dnp->dn_aggfun = dt_node_cook(dnp->dn_aggfun, DT_IDFLG_REF);
4097                 dt_node_attr_assign(dnp, dt_ident_cook(dnp,
4098                     dnp->dn_ident, &dnp->dn_aggtup));
4099         } else {
4100                 dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp),
4101                     B_FALSE);
4102                 dt_node_attr_assign(dnp, dnp->dn_ident->di_attr);
4103         }
4104
4105         return (dnp);
4106 }
4107
4108 /*
4109  * Since D permits new variable identifiers to be instantiated in any program
4110  * expression, we may need to cook a clause's predicate either before or after
4111  * the action list depending on the program code in question.  Consider:
4112  *
4113  * probe-description-list       probe-description-list
```

```
4114  * /x++/                         /x == 0/
4115  * {                             {
4116  *     trace(x);                     trace(x++);
4117  * }                             }
4118  *
4119  * In the left-hand example, the predicate uses operator ++ to instantiate 'x'
4120  * as a variable of type int64_t.  The predicate must be cooked first because
4121  * otherwise the statement trace(x) refers to an unknown identifier.  In the
4122  * right-hand example, the action list uses ++ to instantiate 'x'; the action
4123  * list must be cooked first because otherwise the predicate x == 0 refers to
4124  * an unknown identifier.  In order to simplify programming, we support both.
4125  *
4126  * When cooking a clause, we cook the action statements before the predicate by
4127  * default, since it seems more common to create or modify identifiers in the
4128  * action list.  If cooking fails due to an unknown identifier, we attempt to
4129  * cook the predicate (i.e. do it first) and then go back and cook the actions.
4130  * If this, too, fails (or if we get an error other than D_IDENT_UNDEF) we give
4131  * up and report failure back to the user.  There are five possible paths:
4132  *
4133  * cook actions = OK, cook predicate = OK -> OK
4134  * cook actions = OK, cook predicate = ERR -> ERR
4135  * cook actions = ERR, cook predicate = ERR -> ERR
4136  * cook actions = ERR, cook predicate = OK, cook actions = OK -> OK
4137  * cook actions = ERR, cook predicate = OK, cook actions = ERR -> ERR
4138  *
4139  * The programmer can still defeat our scheme by creating circular definition
4140  * dependencies between predicates and actions, as in this example clause:
4141  *
4142  * probe-description-list
4143  * /x++ && y == 0/
4144  * {
4145  *     trace(x + y++);
4146  * }
4147  *
4148  * but it doesn't seem worth the complexity to handle such rare cases.  The
4149  * user can simply use the D variable declaration syntax to work around them.
4150  */
4151 static dt_node_t *
4152 dt_cook_clause(dt_node_t *dnp, uint_t idflags)
4153 {
4154         volatile int err, tries;
4155         jmp_buf ojb;
4156
4157         /*
4158          * Before assigning dn_ctxattr, temporarily assign the probe attribute
4159          * to 'dnp' itself to force an attribute check and minimum violation.
4160          */
4161         dt_node_attr_assign(dnp, yypcb->pcb_pinfo.dtp_attr);
4162         dnp->dn_ctxattr = yypcb->pcb_pinfo.dtp_attr;
4163
4164         bcopy(yypcb->pcb_jmpbuf, ojb, sizeof (jmp_buf));
4165         tries = 0;
4166
4167         if (dnp->dn_pred != NULL && (err = setjmp(yypcb->pcb_jmpbuf)) != 0) {
4168                 bcopy(ojb, yypcb->pcb_jmpbuf, sizeof (jmp_buf));
4169                 if (tries++ != 0 || err != EDT_COMPILER || (
4170                     yypcb->pcb_hdl->dt_errtag != dt_errtag(D_IDENT_UNDEF) &&
4171                     yypcb->pcb_hdl->dt_errtag != dt_errtag(D_VAR_UNDEF)))
4172                         longjmp(yypcb->pcb_jmpbuf, err);
4173         }
4174
4175         if (tries == 0) {
4176                 yylabel("action list");
4177
4178                 dt_node_attr_assign(dnp,
4179                     dt_node_list_cook(&dnp->dn_acts, idflags));
```



```
4181                 bcopy(ojb, yypcb->pcb_jmpbuf, sizeof (jmp_buf));
4182                 yylabel(NULL);
4183         }
4184
4185         if (dnp->dn_pred != NULL) {
4186                 yylabel("predicate");
4187
4188                 dnp->dn_pred = dt_node_cook(dnp->dn_pred, idflags);
4189                 dt_node_attr_assign(dnp,
4190                     dt_attr_min(dnp->dn_attr, dnp->dn_pred->dn_attr));
4191
4192                 if (!dt_node_is_scalar(dnp->dn_pred)) {
4193                         xyerror(D_PRED_SCALAR,
4194                             "predicate result must be of scalar type\n");
4195                 }
4196
4197                 yylabel(NULL);
4198         }
4199
4200         if (tries != 0) {
4201                 yylabel("action list");
4202
4203                 dt_node_attr_assign(dnp,
4204                     dt_node_list_cook(&dnp->dn_acts, idflags));
4205
4206                 yylabel(NULL);
4207         }
4208
4209         return (dnp);
4210 }
4211
4212 /*ARGSUSED*/
4213 static dt_node_t *
4214 dt_cook_inline(dt_node_t *dnp, uint_t idflags)
4215 {
4216         dt_idnode_t *inp = dnp->dn_ident->di_iarg;
4217         dt_ident_t *rdp;
4218
4219         char n1[DT_TYPE_NAMELEN];
4220         char n2[DT_TYPE_NAMELEN];
4221
4222         assert(dnp->dn_ident->di_flags & DT_IDFLG_INLINE);
4223         assert(inp->din_root->dn_flags & DT_NF_COOKED);
4224
4225         /*
4226          * If we are inlining a translation, verify that the inline declaration
4227          * type exactly matches the type that is returned by the translation.
4228          * Otherwise just use dt_node_is_argcompat() to check the types.
4229          */
4230         if ((rdp = dt_node_resolve(inp->din_root, DT_IDENT_XLSOU)) != NULL ||
4231             (rdp = dt_node_resolve(inp->din_root, DT_IDENT_XLPTR)) != NULL) {
4232
4233                 ctf_file_t *lctfp = dnp->dn_ctfp;
4234                 ctf_id_t ltype = ctf_type_resolve(lctfp, dnp->dn_type);
4235
4236                 dt_xlator_t *dxp = rdp->di_data;
4237                 ctf_file_t *rctfp = dxp->dx_dst_ctfp;
4238                 ctf_id_t rtype = dxp->dx_dst_base;
4239
4240                 if (ctf_type_kind(lctfp, ltype) == CTF_K_POINTER) {
4241                         ltype = ctf_type_reference(lctfp, ltype);
4242                         ltype = ctf_type_resolve(lctfp, ltype);
4243                 }
4244
4245                 if (ctf_type_compat(lctfp, ltype, rctfp, rtype) == 0) {
```

```
4246                        dnerror(dnp, D_OP_INCOMPAT,
4247                            "inline %s definition uses incompatible types: "
4248                            "\"%s\" = \"%s\"\n", dnp->dn_ident->di_name,
4249                            dt_type_name(lctfp, ltype, n1, sizeof (n1)),
4250                            dt_type_name(rctfp, rtype, n2, sizeof (n2)));
4251                }

4253        } else if (dt_node_is_argcompat(dnp, inp->din_root) == 0) {
4254                dnerror(dnp, D_OP_INCOMPAT,
4255                    "inline %s definition uses incompatible types: "
4256                    "\"%s\" = \"%s\"\n", dnp->dn_ident->di_name,
4257                    dt_node_type_name(dnp, n1, sizeof (n1)),
4258                    dt_node_type_name(inp->din_root, n2, sizeof (n2)));
4259        }

4261        return (dnp);
4262 }

4264 static dt_node_t *
4265 dt_cook_member(dt_node_t *dnp, uint_t idflags)
4266 {
4267        dnp->dn_membexpr = dt_node_cook(dnp->dn_membexpr, idflags);
4268        dt_node_attr_assign(dnp, dnp->dn_membexpr->dn_attr);
4269        return (dnp);
4270 }

4272 /*ARGSUSED*/
4273 static dt_node_t *
4274 dt_cook_xlator(dt_node_t *dnp, uint_t idflags)
4275 {
4276        dtrace_hdl_t *dtp = yypcb->pcb_hdl;
4277        dt_xlator_t *dxp = dnp->dn_xlator;
4278        dt_node_t *mnp;

4280        char n1[DT_TYPE_NAMELEN];
4281        char n2[DT_TYPE_NAMELEN];

4283        dtrace_attribute_t attr = _dtrace_maxattr;
4284        ctf_membinfo_t ctm;
4285        ctf_id_t type;
4286        ctf_file_t *ctfp;
4287 #endif /* ! codereview */

4289        /*
4290         * Before cooking each translator member, we push a reference to the
4291         * hash containing translator-local identifiers on to pcb_globals to
4292         * temporarily interpose these identifiers in front of other globals.
4293         */
4294        dt_idstack_push(&yypcb->pcb_globals, dxp->dx_locals);

4296        for (mnp = dnp->dn_members; mnp != NULL; mnp = mnp->dn_list) {
4297                if (ctf_member_info(dxp->dx_dst_ctfp, dxp->dx_dst_type,
4298                    mnp->dn_membname, &ctm) == CTF_ERR) {
4299                        xyerror(D_XLATE_MEMB,
4300                            "translator member %s is not a member of %s\n",
4301                            mnp->dn_membname, ctf_type_name(dxp->dx_dst_ctfp,
4302                            dxp->dx_dst_type, n1, sizeof (n1)));
4303                }

4305                (void) dt_node_cook(mnp, DT_IDFLG_REF);
4306                ctfp = dxp->dx_dst_ctfp;
4307                type = ctm.ctm_type;

4309                /*
4310                 *
4311                 * This probably doesn't need to be resolved, because it's of
```



```
4312                 * the translator, but is done for completeness right now.
4313                 */
4314                dt_resolve_forward_decl(&ctfp, &type);
4315                dt_node_type_assign(mnp, ctfp, type,
3746                dt_node_type_assign(mnp, dxp->dx_dst_ctfp, ctm.ctm_type,
4316                    B_FALSE);
4317                attr = dt_attr_min(attr, mnp->dn_attr);

4319                if (dt_node_is_argcompat(mnp, mnp->dn_membexpr) == 0) {
4320                        xyerror(D_XLATE_INCOMPAT,
4321                            "translator member %s definition uses "
4322                            "incompatible types: \"%s\" = \"%s\"\n",
4323                            mnp->dn_membname,
4324                            dt_node_type_name(mnp, n1, sizeof (n1)),
4325                            dt_node_type_name(mnp->dn_membexpr,
4326                            n2, sizeof (n2)));
4327                }
4328        }

4330        dt_idstack_pop(&yypcb->pcb_globals, dxp->dx_locals);

4332        dxp->dx_souid.di_attr = attr;
4333        dxp->dx_ptrid.di_attr = attr;

4335        dt_node_type_assign(dnp, DT_DYN_CTFP(dtp), DT_DYN_TYPE(dtp), B_FALSE);
4336        dt_node_attr_assign(dnp, _dtrace_defattr);

4338        return (dnp);
4339 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   11776 Fri Apr 17 12:29:29 2015
new/usr/src/lib/libdtrace/common/dt_xlator.c
libdtrace: attempt to resolve FORWARD types to concrete types
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2013 Joyent, Inc. All rights reserved.
 */

#include <strings.h>
#include <assert.h>

#include <dt_xlator.h>
#include <dt_parser.h>
#include <dt_grammar.h>
#include <dt_module.h>
#include <dt_impl.h>

/*
 * Create a member node corresponding to one of the output members of a dynamic
 * translator.  We set the member's dn_membexpr to a DT_NODE_XLATOR node that
 * has dn_op set to DT_TOK_XLATE and refers back to the translator itself.  The
 * code generator will then use this as the indicator for dynamic translation.
 */
/*ARGSUSED*/
static int
dt_xlator_create_member(const char *name, ctf_id_t type, ulong_t off, void *arg)
{
        dt_xlator_t *dxp = arg;
        dtrace_hdl_t *dtp = dxp->dx_hdl;
        dt_node_t *enp, *mnp;
        ctf_file_t *ctfp;
#endif /* ! codereview */

        if ((enp = dt_node_xalloc(dtp, DT_NODE_XLATOR)) == NULL)
                return (dt_set_errno(dtp, EDT_NOMEM));

        enp->dn_link = dxp->dx_nodes;
        dxp->dx_nodes = enp;
```



```
        if ((mnp = dt_node_xalloc(dtp, DT_NODE_MEMBER)) == NULL)
                return (dt_set_errno(dtp, EDT_NOMEM));

        mnp->dn_link = dxp->dx_nodes;
        dxp->dx_nodes = mnp;

        /*
         * For the member expression, we use a DT_NODE_XLATOR/TOK_XLATE whose
         * xlator refers back to the translator and whose dn_xmember refers to
         * the current member.  These refs will be used by dt_cg.c and dt_as.c.
         */
        enp->dn_op = DT_TOK_XLATE;
        enp->dn_xlator = dxp;
        enp->dn_xmember = mnp;
        /*
         * XXX: Is it ok for the CTF of the type to not be from the dst ctf?
         *
         * I suspect it's actually unnecessary, but I'm also unclear on
         * dynamic translators
         */
        dt_resolve_forward_decl(&ctfp, &type);
        dt_node_type_assign(enp, ctfp, type, B_FALSE);
        dt_node_type_assign(enp, dxp->dx_dst_ctfp, type, B_FALSE);

        /*
         * For the member itself, we use a DT_NODE_MEMBER as usual with the
         * appropriate name, output type, and member expression set to 'enp'.
         */
        if (dxp->dx_members != NULL) {
                assert(enp->dn_link->dn_kind == DT_NODE_MEMBER);
                enp->dn_link->dn_list = mnp;
        } else
                dxp->dx_members = mnp;

        mnp->dn_membname = strdup(name);
        mnp->dn_membexpr = enp;
        dt_node_type_assign(mnp, dxp->dx_dst_ctfp, type, B_FALSE);

        if (mnp->dn_membname == NULL)
                return (dt_set_errno(dtp, EDT_NOMEM));

        return (0);
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    6259 Fri Apr 17 12:29:30 2015
new/usr/src/uts/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2012 Nexenta Systems, Inc. All rights reserved.
#
# include global definitions
include ../Makefile.master

#
# List of architectures to build as part of the standard build.
#
# Some of these architectures are built in parallel (see i386_PARALLEL and
# sparc_PARALLEL). This requires building some parts first before parallel build
# can start. Platform make files know what should be built as a prerequisite for
# the parallel build to work. The i386_PREREQ and sparc_PREREQ variables tell
# which platform directory to enter to start making prerequisite dependencies.
#
sparc_ARCHITECTURES = sun4v sun4u sparc

i386_ARCHITECTURES = i86pc i86xpv intel

#
# For i386 all architectures can be compiled in parallel.
#
# intel/Makefile knows how to build prerequisites needed for parallel build.
#
i386_PREREQ = intel
i386_PARALLEL = $(i386_ARCHITECTURES)

#
# For sparc all architectures can be compiled in parallel.
#
# sun4/Makefile knows how to build prerequisites needed for parallel build.
# can start.
#
sparc_PREREQ = sun4
sparc_PARALLEL = $(sparc_ARCHITECTURES)

#
# Platforms defined in $(MACH)_PARALLEL are built in parallel. DUMMY is placed
```



```
# at the end in case $(MACH)_PARALLEL is empty to prevent everything going in
# parallel.
#
.PARALLEL: $($(MACH)_PARALLEL) DUMMY

#
# For build prerequisites we use a special target which is constructed by adding
# '.prereq' suffix to the $(MACH)_PREREQ.
#
PREREQ_TARGET = $($(MACH)_PREREQ:%=%.prereq)


def             :=      TARGET= def
all             :=      TARGET= all
install         :=      TARGET= install
install_h       :=      TARGET= install_h
clean           :=      TARGET= clean
clobber         :=      TARGET= clobber
clobber_h       :=      TARGET= clobber
lint            :=      TARGET= lint
clean.lint      :=      TARGET= clean.lint
check           :=      TARGET= check
modlist         :=      TARGET= modlist
modlist         :=      NO_STATE= -K $$MODSTATE$$$$

.KEEP_STATE:

def all lint: all_h $(PMTMO_FILE) $($(MACH)_ARCHITECTURES)

install: all_h install_dirs $(PMTMO_FILE) $($(MACH)_ARCHITECTURES)

install_dirs:
        @cd ..; pwd; $(MAKE) rootdirs
        @pwd

#
# Rule to build prerequisites. The left part of the pattern will match
# PREREQ_TARGET.
#
# The location of the Makefile is determined by strippinng '.prereq' suffix from
# the target name. We add '.prereq' suffix to the target passed to the child
# Makefile so that it can distinguish prerequisite build from the regular one.
#
#
%.prereq:
        @cd $(@:%.prereq=%); pwd; $(MAKE) $(NO_STATE) $(TARGET).prereq

#
# Rule to build architecture files. Build all required prerequisites and then
# build the rest (potentially in parallel).
#
$($(MACH)_ARCHITECTURES): $(PREREQ_TARGET) FRC
        @cd $@; pwd; $(MAKE) $(NO_STATE) $(TARGET)

$(PMTMO_FILE) pmtmo_file: $(PATCH_MAKEUP_TABLE)
        @if [ -z "$(PATCH_MAKEUP_TABLE)" ] ; then \
                echo 'ERROR: $$(PATCH_MAKEUP_TABLE) not set' \
                    'in environment' >&2 ; \
                exit 1 ; \
        fi
        RELEASE="$(RELEASE)" MACH="$(MACH)" \
            $(CTFCVTPTBL) -o $(PMTMO_FILE) $(PATCH_MAKEUP_TABLE)

#
# The following is the list of directories which contain Makefiles with
# targets to install header file. The machine independent headers are
```

```
118 # installed by invoking the Makefile in the directory containing the
119 # header files. Machine and architecture dependent headers are installed
120 # by invoking the main makefile for that architecture/machine which,
121 # in turn, is responsible for invoking the Makefiles which install headers.
122 # It is done this way so as not to assume that all of the header files in
123 # the architecture/machine dependent subdirectories are in completely
124 # isomorphic locations.
125 #
126 COMMON_HDRDIRS= common/avs \
127                 common/c2 \
128                 common/des \
129                 common/fs \
130                 common/gssapi \
131                 common/idmap \
132                 common/klm \
133                 common/inet \
134                 common/inet/ipf/netinet \
135                 common/inet/kssl \
136                 common/inet/nca \
137                 common/inet/sockmods/netpacket \
138                 common/io/bpf/net \
139                 common/io/fibre-channel/fca/qlc \
140                 common/io/lvm/md \
141                 common/ipp \
142                 common/net \
143                 common/netinet \
144                 common/nfs \
145                 common/pcmcia/sys \
146                 common/rpc \
147                 common/rpcsvc \
148                 common/sharefs \
149                 common/smb \
150                 common/smbsrv \
151                 common/sys \
152                 common/vm


155 #
156 # Subset of COMMON_HDRDIRS in which at least one header is generated
157 # at runtime (e.g., rpcgen), and in which "make clean" should run.
158 # Other directories should be included here, but do not yet have the
159 # necessary Makefile support (make clean).  See 6414855.
160 #
161 DYNHDRDIRS =    common/avs \
162                 common/gssapi \
163                 common/idmap \
164                 common/io/fibre-channel/fca/qlc \
165                 common/io/lvm/md \
166                 common/klm \
167                 common/rpc \
168                 common/rpcsvc \
169                 common/sys

171 sparc_HDRDIRS= sun/sys
172 i386_HDRDIRS= i86pc/vm i86xpv/vm

174 HDRDIRS= $(COMMON_HDRDIRS) $($(MACH)_HDRDIRS)
175 install_h check: $(HDRDIRS) $($(MACH)_ARCHITECTURES)

177 $(HDRDIRS): FRC
178         @cd $@; pwd; $(MAKE) $(TARGET)

180 # ensures that headers made by rpcgen and others are available in uts source
181 # for kernel builds to reference without building install_h
182 #
183 all_h: FRC
```



```
184         @cd common/sys; pwd; $(MAKE) $@
185         @cd common/rpc; pwd; $(MAKE) $@
186         @cd common/rpcsvc; pwd; $(MAKE) $@
187         @cd common/gssapi; pwd; $(MAKE) $@
188         @cd common/idmap; pwd; $(MAKE) $@
189         @cd common/klm; pwd; $(MAKE) $@

191 clean clobber: $($(MACH)_ARCHITECTURES) $(DYNHDRDIRS)
200         @if [ '$(PATCH_BUILD)' != '#' ] ; then \
201                 echo $(RM) $(PMTMO_FILE) ; \
202                 $(RM) $(PMTMO_FILE) ; \
203         fi

193 # testing convenience
194 clobber_h: $(DYNHDRDIRS)

196 clean.lint modlist: $($(MACH)_ARCHITECTURES)

198 #
199 # Cross-reference customization: build a cross-reference over all of
200 # the supported architectures.  Although there's no correct way to set
201 # the include path (since we don't know what architecture is the one
202 # the user will be interested in), it's historically been set to
203 # mirror the $(XRDIRS) list, and that works kinda sorta okay.
204 #
205 XRDIRS = $(sparc_ARCHITECTURES) $(i386_ARCHITECTURES) sun4 sfmmu        \
206         sun common

208 XRINCDIRS = $(XRDIRS)

210 cscope.out tags: FRC
211         $(XREF) -x $@

213 FRC:
```

```
**********************************************************
   13776 Fri Apr 17 12:29:30 2015
new/usr/src/uts/Makefile.targ
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
```

```
#
# 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) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
#
#       This Makefiles contains the common targets and definitions for
#       all kernels. It is to be included in the Makefiles for specific
#       implementation architectures and processor architecture dependent
#       modules: i.e.: all driving kernel Makefiles.
#


#
#       Default rule for building the lint library directory:
#
$(LINT_LIB_DIR):
        -@mkdir -p $@ 2> /dev/null

#
#       All C objects depend on inline files. However, cc(1) doesn't generate
#       the correct dependency info. Also, these Makefiles don't contain a
#       separate list of C-derived object files (but it is light weight to
#       let the assembler files think they depend upon this when they don't).
#       Fortunately, the inline files won't change very often. So, for now,
#       all objects depend on the inline files. Remove this when the inliner
#       is fixed to drop correct dependency information.
#
$(OBJECTS): $(INLINES)

#
#       Partially link .o files to generate the kmod. The fake dependency
#       on modstubs simplifies things...
#       ELFSIGN_MOD is defined in the individual KCF plug-in modules Makefiles,
#       and will sign the ELF objects using elfsign(1).
#
$(BINARY):              $(OBJECTS)
        $(LD) -r $(LDFLAGS) -o $@ $(OBJECTS)
        $(CTFMERGE_MODULE)
        $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
        $(POST_PROCESS)
        $(ELFSIGN_MOD)
```



```
#
#       This target checks each kmod for undefined entry points. It does not
#       modify the kmod in any way.
#
$(MODULE).check:        FRC
        @BUILD_TYPE=DBG32 $(MAKE) $(MODULE).check.targ

$(MODULE).check.targ:   $(BINARY) $(OBJECTS) $(EXTRA_CHECK_OBJS) $(UNIX_O) $(MOD
        $(LD) -o /dev/null $(OBJECTS) $(EXTRA_CHECK_OBJS) $(UNIX_O) $(MODSTUBS_O

#
#       Module lint library construction targets.
#
MOD_LINT_LIB    = $(LINT_LIB_DIR)/llib-l$(LINT_MODULE).ln

$(MOD_LINT_LIB):        $(LINT_LIB_DIR) $(LINTS)
        @-$(ECHO) "\n$(OBJS_DIR)/$(MODULE): (library construction):"
        @($(LINT) -o $(LINT_MODULE)-$(OBJS_DIR) \
                $(LINTFLAGS) $(LINTS) $(LTAIL))
        @$(MV) llib-l$(LINT_MODULE)-$(OBJS_DIR).ln $@

$(LINT_MODULE).lint:    $(MOD_LINT_LIB) $(LINT_LIB) $(GEN_LINT_LIB)
        @-$(ECHO) "\n$(OBJS_DIR)/$(LINT_MODULE): global crosschecks:"
        @($(LINT) $(LINTFLAGS) $(MOD_LINT_LIB) \
                $(LINT_LIB) $(GEN_LINT_LIB) $(LTAIL))

#
# Since assym.h is a derived file, the dependency must be explicit for
# all files including this file. (This is only actually required in the
# instance when the .nse_depinfo file does not exist.) It may seem that
# the lint targets should also have a similar dependency, but they don't
# since only C headers are included when #defined(lint) is true. The
# actual lists are defined in */Makefile.files.
#
$(ASSYM_DEPS:%=$(OBJS_DIR)/%):  $(DSF_DIR)/$(OBJS_DIR)/assym.h

#
#       Everybody need to know how to create a modstubs.o built with the
#       appropriate flags and located in the appropriate location.
#
$(MODSTUBS_O):  $(MODSTUBS)
        $(COMPILE.s) -o $@ $(MODSTUBS)

$(LINTS_DIR)/modstubs.ln:       $(MODSTUBS)
        @($(LHEAD) $(LINT.s) $(MODSTUBS) $(LTAIL))

#
# Build the source file which contains the kernel's utsname,
# with release, version and machine set as follows:
#
#       release: contents of $(RELEASE) (Spaces replaced by '_')
#       version: contents of $(PATCHID) (Spaces replaced by '_')
#       machine: contents of $(UNAME_M)
#
# Build environment information is only contained in the comment section.
#

$(OBJS_DIR)/vers.o: $(OBJECTS)
        $(COMPILE.c) -DUTS_RELEASE=\"'$(ECHO) $(RELEASE) | sed -e 's/ /_/g''\" \
            -DUTS_VERSION=\"'$(ECHO) $(PATCHID) | sed -e 's/ /_/g''\" \
            -DUTS_PLATFORM=\"$(UNAME_M)\" -o $@ $(SRC)/uts/common/os/vers.c
        $(CTFCONVERT_O)
        $(POST_PROCESS_O)

$(LINTS_DIR)/vers.ln: $(SRC)/uts/common/os/vers.c
        @($(LHEAD) $(LINT.c) -DUTS_RELEASE=\"\" -DUTS_VERSION=\"\" \
```

```
125             -DUTS_PLATFORM=\"\" $(SRC)/uts/common/os/vers.c $(LTAIL))

127 #
128 #       Installation targets and rules:
129 #
130 $(ROOT_MOD_DIR) $(USR_MOD_DIR):
131         -$(INS.dir)

133 $(ROOT_MOD_DIRS_32):    $(ROOT_MOD_DIR)
134         -$(INS.dir)

136 $(USR_MOD_DIRS_32):     $(USR_MOD_DIR)
137         -$(INS.dir)

139 $(ROOT_MOD_DIR)/%:      $(OBJS_DIR)/% $(ROOT_MOD_DIR) FRC
140         $(INS.file)

142 $(ROOT_CPU_DIR)/%:      $(OBJS_DIR)/% $(ROOT_CPU_DIR) FRC
143         $(INS.file)

145 $(ROOT_DRV_DIR)/%:      $(OBJS_DIR)/% $(ROOT_DRV_DIR) FRC
146         $(INS.file)

148 $(ROOT_DTRACE_DIR)/%:   $(OBJS_DIR)/% $(ROOT_DTRACE_DIR) FRC
149         $(INS.file)

151 $(ROOT_EXEC_DIR)/%:     $(OBJS_DIR)/% $(ROOT_EXEC_DIR) FRC
152         $(INS.file)

154 $(ROOT_FS_DIR)/%:       $(OBJS_DIR)/% $(ROOT_FS_DIR) FRC
155         $(INS.file)

157 $(ROOT_SCHED_DIR)/%:    $(OBJS_DIR)/% $(ROOT_SCHED_DIR) FRC
158         $(INS.file)

160 $(ROOT_SOCK_DIR)/%:     $(OBJS_DIR)/% $(ROOT_SOCK_DIR) FRC
161         $(INS.file)

163 $(ROOT_STRMOD_DIR)/%:   $(OBJS_DIR)/% $(ROOT_STRMOD_DIR) FRC
164         $(INS.file)

166 $(ROOT_IPP_DIR)/%:      $(OBJS_DIR)/% $(ROOT_IPP_DIR) FRC
167         $(INS.file)

169 $(ROOT_SYS_DIR)/%:      $(OBJS_DIR)/% $(ROOT_SYS_DIR) FRC
170         $(INS.file)

172 $(ROOT_MISC_DIR)/%:     $(OBJS_DIR)/% $(ROOT_MISC_DIR) FRC
173         $(INS.file)

175 $(ROOT_DACF_DIR)/%:     $(OBJS_DIR)/% $(ROOT_DACF_DIR) FRC
176         $(INS.file)

178 $(ROOT_BRAND_DIR)/%:    $(OBJS_DIR)/% $(ROOT_BRAND_DIR) FRC
179         $(INS.file)

181 $(ROOT_CRYPTO_DIR)/%:   $(OBJS_DIR)/% $(ROOT_CRYPTO_DIR) FRC
182         $(INS.file)

184 $(ROOT_KGSS_DIR)/%:     $(OBJS_DIR)/% $(ROOT_KGSS_DIR) FRC
185         $(INS.file)

187 $(ROOT_SCSI_VHCI_DIR)/%: $(OBJS_DIR)/% $(ROOT_SCSI_VHCI_DIR) FRC
188         $(INS.file)

190 $(ROOT_PMCS_FW_DIR)/%:  $(OBJS_DIR)/% $(ROOT_PMCS_FW_DIR) FRC
```



```
191         $(INS.file)

193 $(ROOT_QLC_FW_DIR)/%:   $(OBJS_DIR)/% $(ROOT_QLC_FW_DIR) FRC
194         $(INS.file)

196 $(ROOT_EMLXS_FW_DIR)/%: $(OBJS_DIR)/% $(ROOT_EMLXS_FW_DIR) FRC
197         $(INS.file)

199 $(ROOT_MACH_DIR)/%:     $(OBJS_DIR)/% $(ROOT_MACH_DIR) FRC
200         $(INS.file)

202 $(ROOT_FONT_DIR)/%:     $(OBJS_DIR)/% $(ROOT_MOD_DIR) $(ROOT_FONT_DIR) FRC
203         $(INS.file)

205 $(ROOT_MAC_DIR)/%:      $(OBJS_DIR)/% $(ROOT_MOD_DIR) $(ROOT_MAC_DIR) FRC
206         $(INS.file)

208 $(USR_DRV_DIR)/%:       $(OBJS_DIR)/% $(USR_DRV_DIR) FRC
209         $(INS.file)

211 $(USR_EXEC_DIR)/%:      $(OBJS_DIR)/% $(USR_EXEC_DIR) FRC
212         $(INS.file)

214 $(USR_FS_DIR)/%:        $(OBJS_DIR)/% $(USR_FS_DIR) FRC
215         $(INS.file)

217 $(USR_SCHED_DIR)/%:     $(OBJS_DIR)/% $(USR_SCHED_DIR) FRC
218         $(INS.file)

220 $(USR_SOCK_DIR)/%:      $(OBJS_DIR)/% $(USR_SOCK_DIR) FRC
221         $(INS.file)

223 $(USR_STRMOD_DIR)/%:    $(OBJS_DIR)/% $(USR_STRMOD_DIR) FRC
224         $(INS.file)

226 $(USR_SYS_DIR)/%:       $(OBJS_DIR)/% $(USR_SYS_DIR) FRC
227         $(INS.file)

229 $(USR_MISC_DIR)/%:      $(OBJS_DIR)/% $(USR_MISC_DIR) FRC
230         $(INS.file)

232 $(USR_DACF_DIR)/%:      $(OBJS_DIR)/% $(USR_DACF_DIR) FRC
233         $(INS.file)

235 $(USR_PCBE_DIR)/%:      $(OBJS_DIR)/% $(USR_PCBE_DIR) FRC
236         $(INS.file)

238 $(USR_DTRACE_DIR)/%:    $(OBJS_DIR)/% $(USR_DTRACE_DIR) FRC
239         $(INS.file)

241 $(USR_BRAND_DIR)/%:     $(OBJS_DIR)/% $(USR_BRAND_DIR) FRC
242         $(INS.file)

244 $(ROOT_KICONV_DIR)/%:   $(OBJS_DIR)/% $(ROOT_KICONV_DIR) FRC
245         $(INS.file)

247 include $(SRC)/Makefile.psm.targ

249 #
250 #       Target for 64b modules
251 #
252 $(ROOT_KERN_DIR_64):
253         -$(INS.dir)

255 $(ROOT_KERN_DIR_64)/%:  $(OBJS_DIR)/% $(ROOT_KERN_DIR_64) FRC
256         $(INS.file)
```

```
%/$(SUBDIR64):          %
        -$(INS.dir)

#
#       Targets for '.conf' file installation.
#
$(ROOT_CONFFILE):       $(SRC_CONFFILE) $(ROOT_CONFFILE:%/$(CONFFILE)=%)
        $(INS.conffile)

#
#       Targets for creating links between common platforms. ROOT_PLAT_LINKS
#       are are the /platform level while ROOT_PLAT_LINKS_2 are one level
#       down (/platform/`uname -i`/{lib|sbin|kernel}.
#
$(ROOT_PLAT_LINKS):
        $(INS.slink1)

$(ROOT_PLAT_LINKS_2):
        $(INS.slink2)

$(USR_PLAT_LINKS):
        $(INS.slink1)

$(USR_PLAT_LINKS_2):
        $(INS.slink2)

#
# multiple builds support
#
def $(DEF_DEPS)                 := TARGET = def
all $(ALL_DEPS)                 := TARGET = all
clean $(CLEAN_DEPS)             := TARGET = clean
clobber $(CLOBBER_DEPS)         := TARGET = clobber
lint $(LINT_DEPS)               := TARGET = lint
modlintlib $(MODLINTLIB_DEPS)   := TARGET = modlintlib
modlist $(MODLIST_DEPS)         := TARGET = modlist
modlist $(MODLIST_DEPS)         := NO_STATE= -K $$MODSTATE$$$$
clean.lint $(CLEAN_LINT_DEPS)   := TARGET = clean.lint
install $(INSTALL_DEPS)         := TARGET = install
symcheck $(SYM_DEPS)            := TARGET = symcheck

ALL_TARGS       = def all clean clobber lint modlintlib \
                  clean.lint lintlib install symcheck

ALL_OBJ32       = $(ALL_TARGS:%=%.obj32)

$(ALL_OBJ32):   FRC
        @BUILD_TYPE=OBJ32 VERSION='$(VERSION)' $(MAKE) $(NO_STATE) $(TARGET).tar

ALL_DEBUG32     = $(ALL_TARGS:%=%.debug32)

$(ALL_DEBUG32): FRC
        @BUILD_TYPE=DBG32 VERSION='$(VERSION)' $(MAKE) $(NO_STATE) $(TARGET).tar

ALL_OBJ64       = $(ALL_TARGS:%=%.obj64)

$(ALL_OBJ64):   FRC
        @BUILD_TYPE=OBJ64 VERSION='$(VERSION)' $(MAKE) $(NO_STATE) $(TARGET).tar

ALL_DEBUG64     = $(ALL_TARGS:%=%.debug64)

$(ALL_DEBUG64): FRC
        @BUILD_TYPE=DBG64 VERSION='$(VERSION)' $(MAKE) $(NO_STATE) $(TARGET).tar

#
```



```
#       Currently only the IP module needs symbol checking on obj64.
#       Other modules have the same global-objs nm output for debug64 and obj64.
#
$(SISCHECK_DEPS):       $(DEF_DEPS)
        @TARG=`$(ECHO) $@ | $(CUT) -d'.' -f2`; \
        MODSYMS=$(MODULE).symbols.$$TARG; \
        if [ -f "$(MODULE).global-objs.$$TARG" ]; then \
                $(GREP) -v '#' $(MODULE).global-objs.$$TARG |$(GREP) . | \
                    $(SORT) -u > $$MODSYMS.tmp; \
                $(NM) $$TARG/$(MODULE) |$(GREP) OBJT |$(GREP) -v UNDEF | \
                    $(CUT) -d'|' -f8 |$(GREP) -v '^___const_' | \
                    $(GREP) -v '\.[0-9]*$$' |$(SORT) -u \
                    > $$MODSYMS.tmp.new; \
                $(DIFF) $$MODSYMS.tmp $$MODSYMS.tmp.new > $$MODSYMS.diff || \
                    ($(ECHO) "warning: $(MODULE) symbol checking:" \
                    "global variable(s) introduced and/or removed."; \
                    $(CAT) $$MODSYMS.diff; exit 1) \
        fi

$(SISCLEAN_DEPS):
        -TARG=`$(ECHO) $@ | $(CUT) -d'.' -f2`; \
        MODSYMS=$(MODULE).symbols.$$TARG; \
        $(RM) $$MODSYMS.tmp $$MODSYMS.tmp.new $$MODSYMS.diff Nothing_to_remove


$(OBJS_DIR):
        -@mkdir -p $@ 2> /dev/null

def.targ:               $(OBJS_DIR) $(ALL_TARGET)

all.targ:               $(OBJS_DIR) $(ALL_TARGET)

lint.targ:              $(OBJS_DIR) $(LINT_TARGET)

modlintlib.targ:        $(OBJS_DIR) $(MOD_LINT_LIB)

install.targ:           $(OBJS_DIR) $(INSTALL_TARGET)

#
# Support for Install.sh.
#

modlist:        $(MODLIST_DEPS)

# paths relative to $(ROOT).
RELMODULE = $(ROOTMODULE:$(ROOT)/%=%)
RELCONF = $(ROOT_CONFFILE:$(ROOT)/%=%)
RELLINK = $(ROOTLINK:$(ROOT)/%=%)
RELUNIX = $(UNIX32_LINK:$(ROOT)/%=%)
RELSOFTLINKS = $(ROOTSOFTLINKS:$(ROOT)/%=%)

MODSRC:sh=              pwd

#
# Generate module information for Install.sh, i.e., specify what files
# Install.sh should include.  Each line looks like
#   <tag> <srcdir> <arg1> <arg2> ...
# where <tag> specifies the type of file, <srcdir> gives the source
# path (useful if there is an error), and <argN> is one or more
# additional bits of information that Install.sh needs (e.g., source
# directory, install directory, filtering tags).  See Install.sh for
# details on the arguments for each tag type, especially the functions
# copymod, filtmod, and filtimpl.
#
# Changes to this target may require corresponding changes to
# Install.sh.
```

```
389 #
390 # Don't issue a MOD entry if it's not in the install list.
391 #

393 $(MODLIST_DEPS): FRC
394         @case $@ in \
395         *32) \
396                 class=32; \
397                 [ -n "$(RELMODULE)" ] && relmodule=`dirname $(RELMODULE)`;; \
398         *64) \
399                 class=64; \
400                 [ -n "$(RELMODULE)" ] && \
401                     relmodule=`dirname $(RELMODULE)`/$(SUBDIR64);; \
402         esac; \
403         if [ -z "$(THISIMPL)" ]; then \
404                 impl=all; \
405         else \
406                 impl=$(THISIMPL); \
407         fi; \
408         if [ -n "$(ROOTMODULE)" -a -n "$(INSTALL_TARGET)" ]; then \
409                 if [ -z "$(MODULE)" ]; then \
410                         module=`basename $(ROOTMODULE)`; \
411                 else \
412                         module=$(MODULE); \
413                 fi; \
414                 tinstall="$(INSTALL_TARGET)"; \
415                 for t in $$tinstall; do \
416                         if [ "$(ROOTMODULE)" = $$t ]; then \
417                                 echo MOD $(MODSRC) $$module $$relmodule \
418                                     $$class $$impl; \
419                                 break; \
420                         fi \
421                 done \
422         fi; \
423         if [ -n "$(CONF_SRCDIR)" ]; then \
424                 tinstall="$(INSTALL_TARGET)"; \
425                 for t in $$tinstall; do \
426                         if [ $(ROOT_CONFFILE) = $$t ]; then \
427                                 echo CONF $(MODSRC) $(RELCONF) \
428                                     $(MODSRC)/$(CONF_SRCDIR) $$impl $$module; \
429                                 break; \
430                         fi \
431                 done \
432         fi; \
433         if [ -n "$(ROOTLINK)" ]; then \
434                 rellinks="$(RELLINK)"; \
435                 for r in $$rellinks; do \
436                         if [ $$class = 32 ]; then \
437                                 linkdir=`dirname $$r`; \
438                         else \
439                                 linkdir=`dirname $$r`/$(SUBDIR64); \
440                         fi; \
441                         echo LINK $(MODSRC) $$relmodule $$module \
442                                 $$linkdir `basename $$r` $$impl; \
443                 done \
444         fi; \
445         if [ -n "$(UNIX32_LINK)" ]; then \
446                 echo SYMLINK $(MODSRC) $(SUBDIR64)/$(UNIX) \
447                     `dirname $(RELUNIX)` unix $$impl $$module; \
448         fi; \
449         trelsoftlinks="$(RELSOFTLINKS)"; \
450         for t in $$trelsoftlinks; do \
451                 if [ $$class = 32 ]; then \
452                         linkdir=`dirname $$t`; \
453                 else \
454                         linkdir=`dirname $$t`/$(SUBDIR64); \
```



```
455                 fi; \
456                 linkname=`basename $$t`; \
457                 echo SYMLINK $(MODSRC) $(MODULE) $$linkdir $$linkname \
458                     $$impl $$module; \
459         done

461 #
462 #       Cleanliness is next to ...
463 #
464 clean.targ:
465         -$(RM) $(CLEANFILES) Nothing_to_remove

467 clobber.targ:
468         -$(RM) $(CLOBBERFILES) Nothing_to_remove

470 clean.lint.targ:
471         -$(RM) $(CLEANLINTFILES) Nothing_to_remove

473 #
474 #       Create fake lintlibs in the 64b dirs so
475 #       global linting works
476 #
477 lint64:
478         @$(ECHO) $(MODULE) fake lints
479         @for dir in $(LINT64_DIRS); do \
480                 if [ ! -d $$dir ]; then mkdir $$dir; fi \
481         done
482         @for file in $(LINT64_FILES); do \
483                 if [ ! -f $$file ]; then touch $$file; fi \
484         done

486 #
487 #       In some places we also need to create fake lintlibs for 32b
488 #       dirs so global linting works
489 #
490 lint32:
491         @$(ECHO) $(MODULE) fake lints
492         @for dir in $(LINT32_DIRS); do \
493                 if [ ! -d $$dir ]; then mkdir $$dir; fi \
494         done
495         @for file in $(LINT32_FILES); do \
496                 if [ ! -f $$file ]; then touch $$file; fi \
497         done

499 FRC:
```

```
**********************************************************
   19351 Fri Apr 17 12:29:31 2015
new/usr/src/uts/Makefile.uts
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# 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) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2011 Bayard G. Bell. All rights reserved.
# Copyright (c) 2011 by Delphix. All rights reserved.
# Copyright (c) 2013 Andrew Stormont.  All rights reserved.
#

#
#       This Makefile contains the common targets and definitions for
#       all kernels. It is to be included in the Makefiles for specific
#       implementation architectures and processor architecture dependent
#       modules: i.e.: all driving kernel Makefiles.
#
#       Include global definitions:
#
include $(SRC)/Makefile.master

#
#       No text domain in the kernel.
#
DTEXTDOM =

#
#       Keep references to $(SRC)/common relative.
COMMONBASE=     $(UTSBASE)/../common

#
#       Setup build-specific vars
#       To add a build type:
#               add name to ALL_BUILDS32 & ALL_BUILDS64
#               set CLASS_name and OBJ_DIR_name
#               add targets to Makefile.targ
#


#
#       DEF_BUILDS is for def, lint, sischeck, and install
#       ALL_BUILDS is for everything else (all, clean, ...)
#
```



```
# The NOT_RELEASE_BUILD noise is to maintain compatibility with the
# gatekeeper's nightly build script.
#
DEF_BUILDS32                            = obj32
DEF_BUILDS64                            = obj64
DEF_BUILDSONLY64                        = obj64
$(NOT_RELEASE_BUILD)DEF_BUILDS32        = debug32
$(NOT_RELEASE_BUILD)DEF_BUILDS64        = debug64
$(NOT_RELEASE_BUILD)DEF_BUILDSONLY64    = debug64
ALL_BUILDS32                            = obj32 debug32
ALL_BUILDS64                            = obj64 debug64
ALL_BUILDSONLY64                        = obj64 debug64

#
#       For modules in 64b dirs that aren't built 64b
#       or modules in 64b dirs that aren't built 32b we
#       need to create empty modlintlib files so global lint works
#
LINT32_BUILDS   = debug32
LINT64_BUILDS   = debug64

#
#       Build class (32b or 64b)
#
CLASS_OBJ32     = 32
CLASS_DBG32     = 32
CLASS_OBJ64     = 64
CLASS_DBG64     = 64
CLASS           = $(CLASS_$(BUILD_TYPE))

#
#       Build subdirectory
#
OBJS_DIR_OBJ32  = obj32
OBJS_DIR_DBG32  = debug32
OBJS_DIR_OBJ64  = obj64
OBJS_DIR_DBG64  = debug64
OBJS_DIR        = $(OBJS_DIR_$(BUILD_TYPE))

#
#       Create defaults so empty rules don't
#       confuse make
#
CLASS_          = 32
OBJS_DIR_       = debug32

#
#       Build tools
#
CC_sparc_32     = $(sparc_CC)
CC_sparc_64     = $(sparcv9_CC)

CC_i386_32      = $(i386_CC)
CC_i386_64      = $(amd64_CC)
CC_amd64_64     = $(amd64_CC)

CC              = $(CC_$(MACH)_$(CLASS))

AS_sparc_32     = $(sparc_AS)
AS_sparc_64     = $(sparcv9_AS)

AS_i386_32      = $(i386_AS)
AS_i386_64      = $(amd64_AS)
AS_amd64_64     = $(amd64_AS)

AS              = $(AS_$(MACH)_$(CLASS))
```

```
127 LD_sparc_32     = $(sparc_LD)
128 LD_sparc_64     = $(sparcv9_LD)

130 LD_i386_32      = $(i386_LD)
131 LD_i386_64      = $(amd64_LD)
132 LD_amd64_64     = $(amd64_LD)

134 LD              = $(LD_$(MACH)_$(CLASS))

136 LINT_sparc_32   = $(sparc_LINT)
137 LINT_sparc_64   = $(sparcv9_LINT)

139 LINT_i386_32    = $(i386_LINT)
140 LINT_i386_64    = $(amd64_LINT)
141 LINT_amd64_64   = $(amd64_LINT)

143 LINT            = $(LINT_$(MACH)_$(CLASS))

145 MODEL_32        = ilp32
146 MODEL_64        = lp64
147 MODEL           = $(MODEL_$(CLASS))

149 #
150 #       Build rules for linting the kernel.
151 #
152 LHEAD = $(ECHO) "\n$@";

154 # Note: egrep returns "failure" if there are no matches, which is
155 # exactly the opposite of what we need.
156 LGREP.2 =       if egrep -v ' (_init|_fini|_info) '; then false; else true; fi

158 LTAIL =

160 LINT.c =        $(LINT) -c -dirout=$(LINTS_DIR) $(LINTFLAGS) $(LINT_DEFS) $(CPPF

162 # Please do not add new erroff directives here.  If you need to disable
163 # lint warnings in your module for things that cannot be fixed in any
164 # reasonable manner, please augment LINTTAGS in your module Makefile
165 # instead.
166 LINTTAGS        = -erroff=E_INCONS_ARG_DECL2
167 LINTTAGS        += -erroff=E_INCONS_VAL_TYPE_DECL2

169 LINTFLAGS_sparc_32      = $(LINTCCMODE) -nsxmuF -errtags=yes
170 LINTFLAGS_sparc_64      = $(LINTFLAGS_sparc_32) -m64
171 LINTFLAGS_i386_32       = $(LINTCCMODE) -nsxmuF -errtags=yes
172 LINTFLAGS_i386_64       = $(LINTFLAGS_i386_32) -m64

174 LINTFLAGS       = $(LINTFLAGS_$(MACH)_$(CLASS)) $(LINTTAGS)
175 LINTFLAGS       += $(C99LMODE)

177 #
178 #       Override this variable to modify the name of the lint target.
179 #
180 LINT_MODULE=    $(MODULE)

182 #
183 #       Build the compile/assemble lines:
184 #
185 EXTRA_OPTIONS           =
186 AS_DEFS                 = -D_ASM -D__STDC__=0

188 ALWAYS_DEFS_32          = -D_KERNEL -D_SYSCALL32 -D_DDI_STRICT
189 ALWAYS_DEFS_64          = -D_KERNEL -D_SYSCALL32 -D_SYSCALL32_IMPL -D_ELF64 \
190                         -D_DDI_STRICT
191 #
```



```
192 # XX64  This should be defined by the compiler!
193 #
194 ALWAYS_DEFS_64          += -Dsun -D__sun -D__SVR4
195 ALWAYS_DEFS             = $(ALWAYS_DEFS_$(CLASS))

197 #
198 #       CPPFLAGS is deliberatly set with a "=" and not a "+=".  For the kernel
199 #       the header include path should not look for header files outside of
200 #       the kernel code.  This "=" removes the search path built in
201 #       Makefile.master inside CPPFLAGS.  Ditto for AS_CPPFLAGS.
202 #
203 CPPFLAGS        = $(ALWAYS_DEFS) $(ALL_DEFS) $(CONFIG_DEFS) \
204                     $(INCLUDE_PATH) $(EXTRA_OPTIONS)
205 ASFLAGS         += -P
206 AS_CPPFLAGS     = $(ALWAYS_DEFS) $(ALL_DEFS) $(CONFIG_DEFS) $(AS_DEFS) \
207                     $(AS_INC_PATH) $(EXTRA_OPTIONS)

209 #
210 #       Make it (relatively) easy to share compilation options between
211 #       all kernel implementations.
212 #

214 # Override the default, the kernel is squeaky clean
215 CERRWARN = -errtags=yes -errwarn=%all

217 CERRWARN += -_gcc=-Wno-missing-braces
218 CERRWARN += -_gcc=-Wno-sign-compare
219 CERRWARN += -_gcc=-Wno-unknown-pragmas
220 CERRWARN += -_gcc=-Wno-unused-parameter
221 CERRWARN += -_gcc=-Wno-missing-field-initializers

223 # DEBUG v. -nd make for frequent unused variables, empty conditions, etc. in
224 # -nd builds
225 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-unused
226 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-empty-body

228 C99MODE = $(C99_ENABLE)

230 CFLAGS_uts              =
231 CFLAGS_uts              += $(STAND_FLAGS_$(CLASS))
232 CFLAGS_uts              += $(CCVERBOSE)
233 CFLAGS_uts              += $(ILDOFF)
234 CFLAGS_uts              += $(XAOPT)
235 CFLAGS_uts              += $(CTF_FLAGS_$(CLASS))
236 CFLAGS_uts              += $(CERRWARN)
237 CFLAGS_uts              += $(CCNOAUTOINLINE)
238 CFLAGS_uts              += $(CGLOBALSTATIC)
239 CFLAGS_uts              += $(EXTRA_CFLAGS)
240 CFLAGS_uts              += $(CSOURCEDEBUGFLAGS)
241 CFLAGS_uts              += $(CUSERFLAGS)

243 #
244 #       Declare that $(OBJECTS) and $(LINTS) can be compiled in parallel.
245 #       The DUMMY target is for those instances where OBJECTS and LINTS
246 #       are empty (to avoid an unconditional .PARALLEL).
247 .PARALLEL:      $(OBJECTS) $(LINTS) DUMMY

249 #
250 #       Expanded dependencies
251 #
252 DEF_DEPS        = $(DEF_BUILDS:%=def.%)
253 ALL_DEPS        = $(ALL_BUILDS:%=all.%)
254 CLEAN_DEPS      = $(ALL_BUILDS:%=clean.%)
255 CLOBBER_DEPS    = $(ALL_BUILDS:%=clobber.%)
256 LINT_DEPS       = $(DEF_BUILDS:%=lint.%)
257 MODLINTLIB_DEPS = $(DEF_BUILDS:%=modlintlib.%)
```

```
258 MODLIST_DEPS    = $(DEF_BUILDS:%=modlist.%)
259 CLEAN_LINT_DEPS = $(ALL_BUILDS:%=clean.lint.%)
260 INSTALL_DEPS    = $(DEF_BUILDS:%=install.%)
261 SYM_DEPS        = $(SYM_BUILDS:%=symcheck.%)
262 SISCHECK_DEPS   = $(DEF_BUILDS:%=sischeck.%)
263 SISCLEAN_DEPS   = $(ALL_BUILDS:%=sisclean.%)

265 #
266 #       Default module name
267 #
268 BINARY          = $(OBJS_DIR)/$(MODULE)

270 #
271 #       Default cleanup definitions
272 #
273 CLEANLINTFILES  = $(LINTS) $(MOD_LINT_LIB)
274 CLEANFILES      = $(OBJECTS) $(CLEANLINTFILES)
275 CLOBBERFILES    = $(BINARY) $(CLEANFILES)

277 #
278 #       Installation constants:
279 #
280 #               FILEMODE is the mode given to the kernel modules
281 #               CFILEMODE is the mode given to the '.conf' files
282 #
283 FILEMODE         = 755
284 DIRMODE          = 755
285 CFILEMODE        = 644

287 #
288 #       Special Installation Macros for the installation of '.conf' files.
289 #
290 #       These are unique because they are not installed from the current
291 #       working directory.
292 #
293 # Sigh.  Apparently at some time in the past there was a confusion on
294 # whether the name is SRC_CONFFILE or SRC_CONFILE.  Consistency with the
295 # other names would indicate SRC_CONFFILE, but the voting is >180 Makefiles
296 # with SRC_CONFILE and about 11 with SRC_CONFFILE.  Software development
297 # isn't a popularity contest, though, and so my inclination is to define
298 # both names for now and incrementally convert to SRC_CONFFILE to be consistent
299 # with the other names.
300 #
301 CONFFILE                = $(MODULE).conf
302 SRC_CONFFILE            = $(CONF_SRCDIR)/$(CONFFILE)
303 SRC_CONFILE             = $(SRC_CONFFILE)
304 ROOT_CONFFILE_32        = $(ROOTMODULE).conf
305 ROOT_CONFFILE_64        = $(ROOTMODULE:%/$(SUBDIR64)/$(MODULE)=%/$(MODULE)).conf
306 ROOT_CONFFILE           = $(ROOT_CONFFILE_$(CLASS))


309 INS.conffile= \
310         $(RM) $@; $(INS) -s -m $(CFILEMODE) -f $(@D) $(SRC_CONFFILE)

312 CTFMERGE_MODULE=        $(CTFMERGE) $(CTFMRGFLAGS) -L VERSION -o $@ $(OBJECTS) \
313     $(CTFEXTRAOBJS)
312 #
313 # The CTF merge of child kernel modules is performed against one of the genunix
314 # modules.  For Intel builds, all modules will be used with a single genunix:
315 # the one built in intel/genunix.  For SPARC builds, a given
316 # module may be
317 # used with one of a number of genunix files, depending on what platform the
318 # module is deployed on.  We merge against the sun4u genunix to optimize for
319 # the common case.  We also merge against the ip driver since networking is
320 # typically loaded and types defined therein are shared between many modules.
321 #
```



```
322 CTFMERGE_GUDIR_sparc    = sun4u
323 CTFMERGE_GUDIR_i386     = intel
324 CTFMERGE_GUDIR          = $(CTFMERGE_GUDIR_$(MACH))

326 CTFMERGE_GENUNIX        = \
327         $(UTSBASE)/$(CTFMERGE_GUDIR)/genunix/$(OBJS_DIR)/genunix

329 #
330 # Used to uniquify a non-genunix module against genunix. $VERSION is used
331 # for the label.
332 #
333 # For the ease of developers dropping modules onto possibly unrelated systems,
334 # you can set NO_GENUNIX_UNIQUIFY= in the environment to skip uniquifying
335 # against genunix.
336 #
337 NO_GENUNIX_UNIQUIFY=$(POUND_SIGN)
338 CTFMERGE_GENUNIX_DFLAG=-d $(CTFMERGE_GENUNIX)
339 $(NO_GENUNIX_UNIQUIFY)CTF_GENUNIX_DFLAG=

341 CTFMERGE_UNIQUIFY_AGAINST_GENUNIX       = \
342         $(CTFMERGE) $(CTFMRGFLAGS) -L VERSION \
343         $(CTFMERGE_GENUNIX_DFLAG) -o $@ $(OBJECTS) $(CTFEXTRAOBJS)

345 #
346 # Used to merge the genunix module.
347 #
348 CTFMERGE_GENUNIX_MERGE          = \
349         $(CTFMERGE) $(CTFMRGFLAGS) -L VERSION -o $@ \
350         $(OBJECTS) $(CTFEXTRAOBJS) $(IPCTF_TARGET)

352 #
353 # We ctfmerge the ip objects into genunix to maximize the number of common types
354 # found there, thus maximizing the effectiveness of uniquification.  We don't
355 # want the genunix build to have to know about the individual ip objects, so we
356 # put them in an archive.  The genunix ctfmerge then includes this archive.
357 #
358 IPCTF           = $(IPDRV_DIR)/$(OBJS_DIR)/ipctf.a

315 #
316 # Rule for building fake shared libraries used for symbol resolution
317 # when building other modules.  -znoreloc is needed here to avoid
318 # tripping over code that isn't really suitable for shared libraries.
319 #
320 BUILD.SO                = \
321         $(LD) -o $@ $(GSHARED) $(ZNORELOC) -h $(SONAME)

323 #
324 # SONAME defaults for common fake shared libraries.
325 #
326 $(LIBGEN)               := SONAME = $(MODULE)
327 $(PLATLIB)              := SONAME = misc/platmod
328 $(CPULIB)               := SONAME = 'cpu/$$CPU'
329 $(DTRACESTUBS)          := SONAME = dtracestubs

331 #
332 #       Installation directories
333 #

335 #
336 #       For now, 64b modules install into a subdirectory
337 #       of their 32b brethren.
338 #
339 SUBDIR64_sparc          = sparcv9
340 SUBDIR64_i386           = amd64
341 SUBDIR64                = $(SUBDIR64_$(MACH))
```

```
343 ROOT_MOD_DIR            = $(ROOT)/kernel

345 ROOT_KERN_DIR_32        = $(ROOT_MOD_DIR)
346 ROOT_BRAND_DIR_32       = $(ROOT_MOD_DIR)/brand
347 ROOT_DRV_DIR_32         = $(ROOT_MOD_DIR)/drv
348 ROOT_DTRACE_DIR_32      = $(ROOT_MOD_DIR)/dtrace
349 ROOT_EXEC_DIR_32        = $(ROOT_MOD_DIR)/exec
350 ROOT_FS_DIR_32          = $(ROOT_MOD_DIR)/fs
351 ROOT_SCHED_DIR_32       = $(ROOT_MOD_DIR)/sched
352 ROOT_SOCK_DIR_32        = $(ROOT_MOD_DIR)/socketmod
353 ROOT_STRMOD_DIR_32      = $(ROOT_MOD_DIR)/strmod
354 ROOT_IPP_DIR_32         = $(ROOT_MOD_DIR)/ipp
355 ROOT_SYS_DIR_32         = $(ROOT_MOD_DIR)/sys
356 ROOT_MISC_DIR_32        = $(ROOT_MOD_DIR)/misc
357 ROOT_KGSS_DIR_32        = $(ROOT_MOD_DIR)/misc/kgss
358 ROOT_SCSI_VHCI_DIR_32   = $(ROOT_MOD_DIR)/misc/scsi_vhci
359 ROOT_PMCS_FW_DIR_32     = $(ROOT_MOD_DIR)/misc/pmcs
360 ROOT_QLC_FW_DIR_32      = $(ROOT_MOD_DIR)/misc/qlc
361 ROOT_EMLXS_FW_DIR_32    = $(ROOT_MOD_DIR)/misc/emlxs
362 ROOT_NLMISC_DIR_32      = $(ROOT_MOD_DIR)/misc
363 ROOT_MACH_DIR_32        = $(ROOT_MOD_DIR)/mach
364 ROOT_CPU_DIR_32         = $(ROOT_MOD_DIR)/cpu
365 ROOT_TOD_DIR_32         = $(ROOT_MOD_DIR)/tod
366 ROOT_FONT_DIR_32        = $(ROOT_MOD_DIR)/fonts
367 ROOT_DACF_DIR_32        = $(ROOT_MOD_DIR)/dacf
368 ROOT_CRYPTO_DIR_32      = $(ROOT_MOD_DIR)/crypto
369 ROOT_MAC_DIR_32         = $(ROOT_MOD_DIR)/mac
370 ROOT_KICONV_DIR_32      = $(ROOT_MOD_DIR)/kiconv

372 ROOT_KERN_DIR_64        = $(ROOT_MOD_DIR)/$(SUBDIR64)
373 ROOT_BRAND_DIR_64       = $(ROOT_MOD_DIR)/brand/$(SUBDIR64)
374 ROOT_DRV_DIR_64         = $(ROOT_MOD_DIR)/drv/$(SUBDIR64)
375 ROOT_DTRACE_DIR_64      = $(ROOT_MOD_DIR)/dtrace/$(SUBDIR64)
376 ROOT_EXEC_DIR_64        = $(ROOT_MOD_DIR)/exec/$(SUBDIR64)
377 ROOT_FS_DIR_64          = $(ROOT_MOD_DIR)/fs/$(SUBDIR64)
378 ROOT_SCHED_DIR_64       = $(ROOT_MOD_DIR)/sched/$(SUBDIR64)
379 ROOT_SOCK_DIR_64        = $(ROOT_MOD_DIR)/socketmod/$(SUBDIR64)
380 ROOT_STRMOD_DIR_64      = $(ROOT_MOD_DIR)/strmod/$(SUBDIR64)
381 ROOT_IPP_DIR_64         = $(ROOT_MOD_DIR)/ipp/$(SUBDIR64)
382 ROOT_SYS_DIR_64         = $(ROOT_MOD_DIR)/sys/$(SUBDIR64)
383 ROOT_MISC_DIR_64        = $(ROOT_MOD_DIR)/misc/$(SUBDIR64)
384 ROOT_KGSS_DIR_64        = $(ROOT_MOD_DIR)/misc/kgss/$(SUBDIR64)
385 ROOT_SCSI_VHCI_DIR_64   = $(ROOT_MOD_DIR)/misc/scsi_vhci/$(SUBDIR64)
386 ROOT_PMCS_FW_DIR_64     = $(ROOT_MOD_DIR)/misc/pmcs/$(SUBDIR64)
387 ROOT_QLC_FW_DIR_64      = $(ROOT_MOD_DIR)/misc/qlc/$(SUBDIR64)
388 ROOT_EMLXS_FW_DIR_64    = $(ROOT_MOD_DIR)/misc/emlxs/$(SUBDIR64)
389 ROOT_NLMISC_DIR_64      = $(ROOT_MOD_DIR)/misc/$(SUBDIR64)
390 ROOT_MACH_DIR_64        = $(ROOT_MOD_DIR)/mach/$(SUBDIR64)
391 ROOT_CPU_DIR_64         = $(ROOT_MOD_DIR)/cpu/$(SUBDIR64)
392 ROOT_TOD_DIR_64         = $(ROOT_MOD_DIR)/tod/$(SUBDIR64)
393 ROOT_FONT_DIR_64        = $(ROOT_MOD_DIR)/fonts/$(SUBDIR64)
394 ROOT_DACF_DIR_64        = $(ROOT_MOD_DIR)/dacf/$(SUBDIR64)
395 ROOT_CRYPTO_DIR_64      = $(ROOT_MOD_DIR)/crypto/$(SUBDIR64)
396 ROOT_MAC_DIR_64         = $(ROOT_MOD_DIR)/mac/$(SUBDIR64)
397 ROOT_KICONV_DIR_64      = $(ROOT_MOD_DIR)/kiconv/$(SUBDIR64)

399 ROOT_KERN_DIR           = $(ROOT_KERN_DIR_$(CLASS))
400 ROOT_BRAND_DIR          = $(ROOT_BRAND_DIR_$(CLASS))
401 ROOT_DRV_DIR            = $(ROOT_DRV_DIR_$(CLASS))
402 ROOT_DTRACE_DIR         = $(ROOT_DTRACE_DIR_$(CLASS))
403 ROOT_EXEC_DIR           = $(ROOT_EXEC_DIR_$(CLASS))
404 ROOT_FS_DIR             = $(ROOT_FS_DIR_$(CLASS))
405 ROOT_SCHED_DIR          = $(ROOT_SCHED_DIR_$(CLASS))
406 ROOT_SOCK_DIR           = $(ROOT_SOCK_DIR_$(CLASS))
407 ROOT_STRMOD_DIR         = $(ROOT_STRMOD_DIR_$(CLASS))
408 ROOT_IPP_DIR            = $(ROOT_IPP_DIR_$(CLASS))
```



```
409 ROOT_SYS_DIR            = $(ROOT_SYS_DIR_$(CLASS))
410 ROOT_MISC_DIR           = $(ROOT_MISC_DIR_$(CLASS))
411 ROOT_KGSS_DIR           = $(ROOT_KGSS_DIR_$(CLASS))
412 ROOT_SCSI_VHCI_DIR      = $(ROOT_SCSI_VHCI_DIR_$(CLASS))
413 ROOT_PMCS_FW_DIR        = $(ROOT_PMCS_FW_DIR_$(CLASS))
414 ROOT_QLC_FW_DIR         = $(ROOT_QLC_FW_DIR_$(CLASS))
415 ROOT_EMLXS_FW_DIR       = $(ROOT_EMLXS_FW_DIR_$(CLASS))
416 ROOT_NLMISC_DIR         = $(ROOT_NLMISC_DIR_$(CLASS))
417 ROOT_MACH_DIR           = $(ROOT_MACH_DIR_$(CLASS))
418 ROOT_CPU_DIR            = $(ROOT_CPU_DIR_$(CLASS))
419 ROOT_TOD_DIR            = $(ROOT_TOD_DIR_$(CLASS))
420 ROOT_FONT_DIR           = $(ROOT_FONT_DIR_$(CLASS))
421 ROOT_DACF_DIR           = $(ROOT_DACF_DIR_$(CLASS))
422 ROOT_CRYPTO_DIR         = $(ROOT_CRYPTO_DIR_$(CLASS))
423 ROOT_MAC_DIR            = $(ROOT_MAC_DIR_$(CLASS))
424 ROOT_KICONV_DIR         = $(ROOT_KICONV_DIR_$(CLASS))

426 ROOT_MOD_DIRS_32        = $(ROOT_BRAND_DIR_32) $(ROOT_DRV_DIR_32)
427 ROOT_MOD_DIRS_32        = $(ROOT_BRAND_DIR_32) $(ROOT_DRV_DIR_32)
428 ROOT_MOD_DIRS_32        += $(ROOT_EXEC_DIR_32) $(ROOT_DTRACE_DIR_32)
429 ROOT_MOD_DIRS_32        += $(ROOT_FS_DIR_32) $(ROOT_SCHED_DIR_32)
430 ROOT_MOD_DIRS_32        += $(ROOT_STRMOD_DIR_32) $(ROOT_SYS_DIR_32)
431 ROOT_MOD_DIRS_32        += $(ROOT_IPP_DIR_32) $(ROOT_SOCK_DIR_32)
432 ROOT_MOD_DIRS_32        += $(ROOT_MISC_DIR_32) $(ROOT_MACH_DIR_32)
433 ROOT_MOD_DIRS_32        += $(ROOT_KGSS_DIR_32)
434 ROOT_MOD_DIRS_32        += $(ROOT_SCSI_VHCI_DIR_32)
435 ROOT_MOD_DIRS_32        += $(ROOT_PMCS_FW_DIR_32)
436 ROOT_MOD_DIRS_32        += $(ROOT_QLC_FW_DIR_32)
437 ROOT_MOD_DIRS_32        += $(ROOT_EMLXS_FW_DIR_32)
438 ROOT_MOD_DIRS_32        += $(ROOT_CPU_DIR_32) $(ROOT_FONT_DIR_32)
439 ROOT_MOD_DIRS_32        += $(ROOT_TOD_DIR_32) $(ROOT_DACF_DIR_32)
440 ROOT_MOD_DIRS_32        += $(ROOT_CRYPTO_DIR_32) $(ROOT_MAC_DIR_32)
441 ROOT_MOD_DIRS_32        += $(ROOT_KICONV_DIR_32)

443 USR_MOD_DIR             = $(ROOT)/usr/kernel

445 USR_DRV_DIR_32          = $(USR_MOD_DIR)/drv
446 USR_EXEC_DIR_32         = $(USR_MOD_DIR)/exec
447 USR_FS_DIR_32           = $(USR_MOD_DIR)/fs
448 USR_SCHED_DIR_32        = $(USR_MOD_DIR)/sched
449 USR_SOCK_DIR_32         = $(USR_MOD_DIR)/socketmod
450 USR_STRMOD_DIR_32       = $(USR_MOD_DIR)/strmod
451 USR_SYS_DIR_32          = $(USR_MOD_DIR)/sys
452 USR_MISC_DIR_32         = $(USR_MOD_DIR)/misc
453 USR_DACF_DIR_32         = $(USR_MOD_DIR)/dacf
454 USR_PCBE_DIR_32         = $(USR_MOD_DIR)/pcbe
455 USR_DTRACE_DIR_32       = $(USR_MOD_DIR)/dtrace
456 USR_BRAND_DIR_32        = $(USR_MOD_DIR)/brand

458 USR_DRV_DIR_64          = $(USR_MOD_DIR)/drv/$(SUBDIR64)
459 USR_EXEC_DIR_64         = $(USR_MOD_DIR)/exec/$(SUBDIR64)
460 USR_FS_DIR_64           = $(USR_MOD_DIR)/fs/$(SUBDIR64)
461 USR_SCHED_DIR_64        = $(USR_MOD_DIR)/sched/$(SUBDIR64)
462 USR_SOCK_DIR_64         = $(USR_MOD_DIR)/socketmod/$(SUBDIR64)
463 USR_STRMOD_DIR_64       = $(USR_MOD_DIR)/strmod/$(SUBDIR64)
464 USR_SYS_DIR_64          = $(USR_MOD_DIR)/sys/$(SUBDIR64)
465 USR_MISC_DIR_64         = $(USR_MOD_DIR)/misc/$(SUBDIR64)
466 USR_DACF_DIR_64         = $(USR_MOD_DIR)/dacf/$(SUBDIR64)
467 USR_PCBE_DIR_64         = $(USR_MOD_DIR)/pcbe/$(SUBDIR64)
468 USR_DTRACE_DIR_64       = $(USR_MOD_DIR)/dtrace/$(SUBDIR64)
469 USR_BRAND_DIR_64        = $(USR_MOD_DIR)/brand/$(SUBDIR64)

471 USR_DRV_DIR             = $(USR_DRV_DIR_$(CLASS))
472 USR_EXEC_DIR            = $(USR_EXEC_DIR_$(CLASS))
473 USR_FS_DIR              = $(USR_FS_DIR_$(CLASS))
474 USR_SCHED_DIR           = $(USR_SCHED_DIR_$(CLASS))
```

```
475 USR_SOCK_DIR            = $(USR_SOCK_DIR_$(CLASS))
476 USR_STRMOD_DIR          = $(USR_STRMOD_DIR_$(CLASS))
477 USR_SYS_DIR             = $(USR_SYS_DIR_$(CLASS))
478 USR_MISC_DIR            = $(USR_MISC_DIR_$(CLASS))
479 USR_DACF_DIR            = $(USR_DACF_DIR_$(CLASS))
480 USR_PCBE_DIR            = $(USR_PCBE_DIR_$(CLASS))
481 USR_DTRACE_DIR          = $(USR_DTRACE_DIR_$(CLASS))
482 USR_BRAND_DIR           = $(USR_BRAND_DIR_$(CLASS))

484 USR_MOD_DIRS_32         = $(USR_DRV_DIR_32) $(USR_EXEC_DIR_32)
485 USR_MOD_DIRS_32         += $(USR_FS_DIR_32) $(USR_SCHED_DIR_32)
486 USR_MOD_DIRS_32         += $(USR_STRMOD_DIR_32) $(USR_SYS_DIR_32)
487 USR_MOD_DIRS_32         += $(USR_MISC_DIR_32) $(USR_DACF_DIR_32)
488 USR_MOD_DIRS_32         += $(USR_PCBE_DIR_32)
489 USR_MOD_DIRS_32         += $(USR_DTRACE_DIR_32) $(USR_BRAND_DIR_32)
490 USR_MOD_DIRS_32         += $(USR_SOCK_DIR_32)

492 #
493 #
494 #
495 include $(SRC)/Makefile.psm

497 #
498 #       The "-r" on the remove may be considered temporary, but is required
499 #       while the replacement of the SUNW,SPARCstation-10,SX directory by
500 #       a symbolic link is being propagated.
501 #
502 INS.slink1= $(RM) -r $@; $(SYMLINK) $(PLATFORM) $@
503 INS.slink2= $(RM) -r $@; $(SYMLINK) ../$(PLATFORM)/$(@F) $@
504 INS.slink3= $(RM) -r $@; $(SYMLINK) $(IMPLEMENTED_PLATFORM) $@
505 INS.slink4= $(RM) -r $@; $(SYMLINK) ../$(PLATFORM)/include $@
506 INS.slink5= $(RM) -r $@; $(SYMLINK) ../$(PLATFORM)/sbin $@
507 INS.slink6= $(RM) -r $@; $(SYMLINK) ../../$(PLATFORM)/lib/$(MODULE) $@
508 INS.slink7= $(RM) -r $@; $(SYMLINK) ../../$(PLATFORM)/sbin/$(@F) $@

510 ROOT_PLAT_LINKS          = $(PLAT_LINKS:%=$(ROOT_PLAT_DIR)/%)
511 ROOT_PLAT_LINKS_2        = $(PLAT_LINKS_2:%=$(ROOT_PLAT_DIR)/%)
512 USR_PLAT_LINKS           = $(PLAT_LINKS:%=$(USR_PLAT_DIR)/%)
513 USR_PLAT_LINKS_2         = $(PLAT_LINKS_2:%=$(USR_PLAT_DIR)/%)

515 #
516 # Collection of all relevant, delivered kernel modules.
517 #
518 # Note that we insist on building genunix first, because everything else
519 # uniquifies against it.  When doing a 'make' from usr/src/uts/, we'll enter
520 # the platform directories first.  These will cd into the corresponding genunix
521 # directory and build it.  So genunix /shouldn't/ get rebuilt when we get to
522 # building all the kernel modules.  However, due to an as-yet-unexplained
523 # problem with dependencies, sometimes it does get rebuilt, which then messes
524 # up the other modules.  So we always force the issue here rather than try to
525 # build genunix in parallel with everything else.
526 #
527 PARALLEL_KMODS = $(DRV_KMODS) $(EXEC_KMODS) $(FS_KMODS) $(SCHED_KMODS) \
528                  $(TOD_KMODS) $(STRMOD_KMODS) $(SYS_KMODS) $(MISC_KMODS) \
529                  $(NLMISC_KMODS) $(MACH_KMODS) $(CPU_KMODS) $(GSS_KMODS) \
530                  $(MMU_KMODS) $(DACF_KMODS) $(EXPORT_KMODS) $(IPP_KMODS) \
531                  $(CRYPTO_KMODS) $(PCBE_KMODS) \
532                  $(DRV_KMODS_$(CLASS)) $(MISC_KMODS_$(CLASS)) $(MAC_KMODS) \
533                  $(BRAND_KMODS) $(KICONV_KMODS) \
534                  $(SOCKET_KMODS)

536 KMODS = $(GENUNIX_KMODS) $(PARALLEL_KMODS)

538 $(PARALLEL_KMODS): $(GENUNIX_KMODS)

540 LINT_KMODS = $(DRV_KMODS) $(EXEC_KMODS) $(FS_KMODS) $(SCHED_KMODS) \
```



```
541           $(TOD_KMODS) $(STRMOD_KMODS) $(SYS_KMODS) $(MISC_KMODS) \
542           $(MACH_KMODS) $(GSS_KMODS) $(DACF_KMODS) $(IPP_KMODS) \
543           $(CRYPTO_KMODS) $(PCBE_KMODS) \
544           $(DRV_KMODS_$(CLASS)) $(MISC_KMODS_$(CLASS)) $(MAC_KMODS) \
545           $(BRAND_KMODS) $(KICONV_KMODS) $(SOCKET_KMODS)

547 #
548 #       Files to be compiled with -xa, to generate basic block execution
549 #       count data.
550 #
551 #       There are several ways to compile parts of the kernel for kcov:
552 #               1)  Add targets to BB_FILES here or in other Makefiles
553 #                       (they must in the form of $(OBJS_DIR)/target.o)
554 #               2)  setenv BB_FILES '$(XXX_OBJS:%=$(OBJS_DIR)/%)'
555 #               3)  setenv BB_FILES '$(OBJECTS)'
556 #
557 #       Do NOT setenv CFLAGS -xa, as that will cause infinite recursion
558 #       in unix_bb.o
559 #
560 BB_FILES =
561 $(BB_FILES)     := XAOPT = -xa

563 #
564 #       The idea here is for unix_bb.o to be in all kernels except the
565 #       kernel which actually gets shipped to customers.  In practice,
566 #       $(RELEASE_BUILD) is on for a number of the late beta and fcs builds.
567 #
568 $(NOT_RELEASE_BUILD)$(OBJS_DIR)/unix_bb.o   := CPPFLAGS     += -DKCOV
569 $(NOT_RELEASE_BUILD)$(OBJS_DIR)/unix_bb.ln  := CPPFLAGS     += -DKCOV

571 #
572 #       Do not let unix_bb.o get compiled with -xa!
573 #
574 $(OBJS_DIR)/unix_bb.o   := XAOPT =

576 #
577 # Privilege files
578 #
579 PRIVS_AWK = $(SRC)/uts/common/os/privs.awk
580 PRIVS_DEF = $(SRC)/uts/common/os/priv_defs

582 #
583 # USB device data
584 #
585 USBDEVS_AWK =   $(SRC)/uts/common/io/usb/usbdevs2h.awk
586 USBDEVS_DATA =  $(SRC)/uts/common/io/usb/usbdevs
```

```
**********************************************************
    2422 Fri Apr 17 12:29:31 2015
new/usr/src/uts/common/ctf/mapfile
1730 DTrace should ignore type information from modules with cth_parlabel mismat
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
**********************************************************
#
# Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

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

$mapfile_version 2

SYMBOL_SCOPE {
        global:
                ctf_add_array;
                ctf_add_const;
                ctf_add_enum;
                ctf_add_enumerator;
                ctf_add_float;
                ctf_add_forward;
                ctf_add_function;
                ctf_add_integer;
                ctf_add_member;
                ctf_add_pointer;
                ctf_add_restrict;
                ctf_add_struct;
                ctf_add_type;
                ctf_add_typedef;
                ctf_add_union;
                ctf_add_volatile;
                ctf_array_info;
                ctf_bufopen;
```



```
                ctf_close;
                ctf_create;
                ctf_discard;
                ctf_enum_iter;
                ctf_enum_name;
                ctf_enum_value;
                ctf_errmsg;
                ctf_errno;
                ctf_fdopen;
                ctf_func_args;
                ctf_func_info;
                ctf_getmodel;
                ctf_getspecific;
                ctf_import;
                ctf_label_info;
                ctf_label_iter;
                ctf_label_topmost;
                ctf_lookup_by_name;
                ctf_lookup_by_symbol;
                ctf_member_info;
                ctf_member_iter;
                ctf_modopen;
                ctf_open;
                ctf_parent_file;
                ctf_parent_name;
                ctf_parent_label;
#endif /* ! codereview */
                ctf_setmodel;
                ctf_setspecific;
                ctf_set_array;
                ctf_type_align;
                ctf_type_cmp;
                ctf_type_compat;
                ctf_type_encoding;
                ctf_type_iter;
                ctf_type_kind;
                ctf_type_lname;
                ctf_type_name;
                ctf_type_pointer;
                ctf_type_reference;
                ctf_type_resolve;
                ctf_type_size;
                ctf_type_visit;
                ctf_update;
                ctf_version;
                ctf_write;
                _info;
                _init;
                _fini;
        local:
                *;
};
```

```
**********************************************************
    9627 Fri Apr 17 12:29:31 2015
new/usr/src/uts/common/sys/ctf_api.h
1730 DTrace should ignore type information from modules with cth_parlabel mismat
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
**********************************************************
______unchanged_portion_omitted_

 110 #define CTF_FUNC_VARARG 0x1     /* function arguments end with varargs */

 112 /*
 113  * Functions that return integer status or a ctf_id_t use the following value
 114  * to indicate failure.  ctf_errno() can be used to obtain an error code.
 115  */
 116 #define CTF_ERR (-1L)

 118 /*
 119  * The CTF data model is inferred to be the caller's data model or the data
 120  * model of the given object, unless ctf_setmodel() is explicitly called.
 121  */
 122 #define CTF_MODEL_ILP32 1       /* object data model is ILP32 */
 123 #define CTF_MODEL_LP64  2       /* object data model is LP64 */
 124 #ifdef _LP64
 125 #define CTF_MODEL_NATIVE        CTF_MODEL_LP64
 126 #else
 127 #define CTF_MODEL_NATIVE        CTF_MODEL_ILP32
 128 #endif

 130 /*
 131  * Dynamic CTF containers can be created using ctf_create().  The ctf_add_*
 132  * routines can be used to add new definitions to the dynamic container.
 133  * New types are labeled as root or non-root to determine whether they are
 134  * visible at the top-level program scope when subsequently doing a lookup.
 135  */
 136 #define CTF_ADD_NONROOT 0       /* type only visible in nested scope */
 137 #define CTF_ADD_ROOT    1       /* type visible at top-level scope */

 139 /*
 140  * These typedefs are used to define the signature for callback functions
 141  * that can be used with the iteration and visit functions below:
 142  */
 143 typedef int ctf_visit_f(const char *, ctf_id_t, ulong_t, int, void *);
 144 typedef int ctf_member_f(const char *, ctf_id_t, ulong_t, void *);
 145 typedef int ctf_enum_f(const char *, int, void *);
 146 typedef int ctf_type_f(ctf_id_t, void *);
 147 typedef int ctf_label_f(const char *, const ctf_lblinfo_t *, void *);

 149 extern ctf_file_t *ctf_bufopen(const ctf_sect_t *, const ctf_sect_t *,
 150     const ctf_sect_t *, int *);
 151 extern ctf_file_t *ctf_fdopen(int, int *);
 152 extern ctf_file_t *ctf_open(const char *, int *);
 153 extern ctf_file_t *ctf_create(int *);
 154 extern ctf_file_t *ctf_dup(ctf_file_t *);
 155 extern void ctf_close(ctf_file_t *);

 157 extern ctf_file_t *ctf_parent_file(ctf_file_t *);
 158 extern const char *ctf_parent_name(ctf_file_t *);
 159 extern const char *ctf_parent_label(ctf_file_t *);
 160 #endif /* ! codereview */

 162 extern int ctf_import(ctf_file_t *, ctf_file_t *);
 163 extern int ctf_setmodel(ctf_file_t *, int);
 164 extern int ctf_getmodel(ctf_file_t *);

 166 extern void ctf_setspecific(ctf_file_t *, void *);
```



```
 167 extern void *ctf_getspecific(ctf_file_t *);

 169 extern int ctf_errno(ctf_file_t *);
 170 extern const char *ctf_errmsg(int);
 171 extern int ctf_version(int);

 173 extern int ctf_func_info(ctf_file_t *, ulong_t, ctf_funcinfo_t *);
 174 extern int ctf_func_args(ctf_file_t *, ulong_t, uint_t, ctf_id_t *);

 176 extern ctf_id_t ctf_lookup_by_name(ctf_file_t *, const char *);
 177 extern ctf_id_t ctf_lookup_by_symbol(ctf_file_t *, ulong_t);

 179 extern ctf_id_t ctf_type_resolve(ctf_file_t *, ctf_id_t);
 180 extern ssize_t ctf_type_lname(ctf_file_t *, ctf_id_t, char *, size_t);
 181 extern char *ctf_type_name(ctf_file_t *, ctf_id_t, char *, size_t);
 182 extern char *ctf_type_qname(ctf_file_t *, ctf_id_t, char *, size_t,
 183     const char *);
 184 extern ssize_t ctf_type_size(ctf_file_t *, ctf_id_t);
 185 extern ssize_t ctf_type_align(ctf_file_t *, ctf_id_t);
 186 extern int ctf_type_kind(ctf_file_t *, ctf_id_t);
 187 extern ctf_id_t ctf_type_reference(ctf_file_t *, ctf_id_t);
 188 extern ctf_id_t ctf_type_pointer(ctf_file_t *, ctf_id_t);
 189 extern int ctf_type_encoding(ctf_file_t *, ctf_id_t, ctf_encoding_t *);
 190 extern int ctf_type_visit(ctf_file_t *, ctf_id_t, ctf_visit_f *, void *);
 191 extern int ctf_type_cmp(ctf_file_t *, ctf_id_t, ctf_file_t *, ctf_id_t);
 192 extern int ctf_type_compat(ctf_file_t *, ctf_id_t, ctf_file_t *, ctf_id_t);

 194 extern int ctf_member_info(ctf_file_t *, ctf_id_t, const char *,
 195     ctf_membinfo_t *);
 196 extern int ctf_array_info(ctf_file_t *, ctf_id_t, ctf_arinfo_t *);

 198 extern const char *ctf_enum_name(ctf_file_t *, ctf_id_t, int);
 199 extern int ctf_enum_value(ctf_file_t *, ctf_id_t, const char *, int *);

 201 extern const char *ctf_label_topmost(ctf_file_t *);
 202 extern int ctf_label_info(ctf_file_t *, const char *, ctf_lblinfo_t *);

 204 extern int ctf_member_iter(ctf_file_t *, ctf_id_t, ctf_member_f *, void *);
 205 extern int ctf_enum_iter(ctf_file_t *, ctf_id_t, ctf_enum_f *, void *);
 206 extern int ctf_type_iter(ctf_file_t *, ctf_type_f *, void *);
 207 extern int ctf_label_iter(ctf_file_t *, ctf_label_f *, void *);

 209 extern ctf_id_t ctf_add_array(ctf_file_t *, uint_t, const ctf_arinfo_t *);
 210 extern ctf_id_t ctf_add_const(ctf_file_t *, uint_t, ctf_id_t);
 211 extern ctf_id_t ctf_add_enum(ctf_file_t *, uint_t, const char *);
 212 extern ctf_id_t ctf_add_float(ctf_file_t *, uint_t,
 213     const char *, const ctf_encoding_t *);
 214 extern ctf_id_t ctf_add_forward(ctf_file_t *, uint_t, const char *, uint_t);
 215 extern ctf_id_t ctf_add_function(ctf_file_t *, uint_t,
 216     const ctf_funcinfo_t *, const ctf_id_t *);
 217 extern ctf_id_t ctf_add_integer(ctf_file_t *, uint_t,
 218     const char *, const ctf_encoding_t *);
 219 extern ctf_id_t ctf_add_pointer(ctf_file_t *, uint_t, ctf_id_t);
 220 extern ctf_id_t ctf_add_type(ctf_file_t *, ctf_file_t *, ctf_id_t);
 221 extern ctf_id_t ctf_add_typedef(ctf_file_t *, uint_t, const char *, ctf_id_t);
 222 extern ctf_id_t ctf_add_restrict(ctf_file_t *, uint_t, ctf_id_t);
 223 extern ctf_id_t ctf_add_struct(ctf_file_t *, uint_t, const char *);
 224 extern ctf_id_t ctf_add_union(ctf_file_t *, uint_t, const char *);
 225 extern ctf_id_t ctf_add_volatile(ctf_file_t *, uint_t, ctf_id_t);

 227 extern int ctf_add_enumerator(ctf_file_t *, ctf_id_t, const char *, int);
 228 extern int ctf_add_member(ctf_file_t *, ctf_id_t, const char *, ctf_id_t);

 230 extern int ctf_set_array(ctf_file_t *, ctf_id_t, const ctf_arinfo_t *);

 232 extern int ctf_delete_type(ctf_file_t *, ctf_id_t);
```

```
234 extern int ctf_update(ctf_file_t *);
235 extern int ctf_discard(ctf_file_t *);
236 extern int ctf_write(ctf_file_t *, int);

238 #ifdef _KERNEL

240 struct module;
241 extern ctf_file_t *ctf_modopen(struct module *, int *);

243 #endif

245 #ifdef  __cplusplus
246 }
247 #endif

249 #endif  /* _CTF_API_H */
```

```
**********************************************************
    6563 Fri Apr 17 12:29:32 2015
new/usr/src/uts/i86pc/unix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#       This makefile drives the production of unix (and unix.o).
#
#       i86pc implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
UNIX            = unix
DBOOT           = dboot
MULTIBOOT       = multiboot

OBJECTS         = $(SPECIAL_OBJS:%=$(OBJS_DIR)/%) \
                  $(CORE_OBJS:%=$(OBJS_DIR)/%) \
                  $(KRTLD_OBJS:%=$(OBJS_DIR)/%) \
                  $(MACH_NOT_YET_KMODS:%=$(OBJS_DIR)/%)

LINTS           = $(SPECIAL_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(CORE_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(KRTLD_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(MACH_NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln) \
                  $(LINTS_DIR)/vers.ln \
                  $(LINTS_DIR)/modstubs.ln

ROOTMODULE      = $(ROOT_PSM_KERN_DIR)/$(UNIX)
ROOT_MULTIBOOT  = $(ROOT_PSM_DIR)/$(MULTIBOOT)

UNIX_BIN        = $(OBJS_DIR)/$(UNIX)
```



```
LIBS            = $(GENLIB)

GENUNIX         = genunix
GENUNIX_DIR     = ../../intel/$(GENUNIX)

LIBOPTS         = -L $(GENUNIX_DIR)/$(OBJS_DIR) -l $(GENUNIX)

CTFEXTRAOBJS    = $(OBJS_DIR)/vers.o

DBOOT_OBJS_DIR  = dboot/$(OBJS_DIR)
DBOOT_OBJECTS   = $(DBOOT_OBJS:%=$(DBOOT_OBJS_DIR)/%)
DBOOT_BIN       = $(DBOOT_OBJS_DIR)/$(DBOOT)
DBOOT_O         = $(OBJS_DIR)/$(DBOOT).o
DBOOT_S         = $(DBOOT_O:%.o=%.s)
DBOOT_LINTS     = $(DBOOT_OBJS:%.o=$(DBOOT_OBJS_DIR)/%.ln)
DBOOT_LINT      = $(i386_LINT)
DBOOT_LINTTAGS  = -erroff=E_STATIC_UNUSED

#
#       Include common rules.
#
include $(UTSBASE)/i86pc/Makefile.i86pc

#
#       Define targets
#
ALL_TARGET      = $(UNIX_BIN) $(MULTIBOOT)
LINT_TARGET     = $(LINT_LIB) $(DBOOT_LINT_LIB)
INSTALL_TARGET  = $(UNIX_BIN) $(MULTIBOOT) $(ROOTMODULE) $(ROOT_MULTIBOOT)

#
#       This is UNIX_DIR. Use a short path.
#
UNIX_DIR        = .

#
#       Overrides
#
CLEANFILES      +=              \
        $(UNIX_O) $(MODSTUBS_O) \
        $(OBJS_DIR)/vers.c $(OBJS_DIR)/vers.o \
        $(OBJS_DIR)/dtracestubs.s \
        $(DTRACESTUBS_O) $(DTRACESTUBS)

CLEANFILES      +=              \
        $(DBOOT_O) $(DBOOT_S)   \
        $(DBOOT_OBJECTS)        \
        $(OBJS_DIR)/bios_call_src.o \
        $(OBJS_DIR)/bios_call_src \
        $(OBJS_DIR)/bios_call.s \
        $(DBOOT_BIN)

CLEANFILES      +=              \
        $(OBJS_DIR)/fb_swtch_src.o \
        $(OBJS_DIR)/fb_swtch_src \
        $(OBJS_DIR)/fb_swtch.s

CLEANFILES      += \
        $(ZLIB_OBJS:%.o=$(OBJS_DIR)/%.o) \
        $(ZLIB_OBJS:%.o=$(OBJS_DIR)/%.ln)

CLOBBERFILES    = $(CLEANFILES) $(UNIX_BIN) $(MULTIBOOT)
CLEANLINTFILES  += $(LINT_LIB) $(DBOOT_LINT_LIB) $(DBOOT_LINTS)

# instr_size needs a special header
$(OBJS_DIR)/instr_size.o :=     EXTRA_OPTIONS   = -I$(SRC)/common/dis/i386
```

```
126 $(OBJS_DIR)/instr_size.ln :=    EXTRA_OPTIONS   = -I$(SRC)/common/dis/i386

128 CFLAGS += -DDIS_MEM

130 #
131 # For now, disable these lint checks; maintainers should endeavor
132 # to investigate and remove these for maximum lint coverage.
133 # Please do not carry these forward to new Makefiles.
134 #
135 LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
136 LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
137 LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
138 LINTTAGS        += -erroff=E_STATIC_UNUSED
139 LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
140 LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

142 CERRWARN        += -_gcc=-Wno-parentheses
143 CERRWARN        += -_gcc=-Wno-uninitialized
144 CERRWARN        += -_gcc=-Wno-char-subscripts
145 CERRWARN        += -_gcc=-Wno-unused-variable
146 CERRWARN        += -_gcc=-Wno-unused-function
147 CERRWARN        += -_gcc=-Wno-unused-label
148 CERRWARN        += -_gcc=-Wno-type-limits
149 CERRWARN        += -_gcc=-Wno-clobbered
150 CERRWARN        += -_gcc=-Wno-empty-body
151 CERRWARN        += -_gcc=-Wno-unused-value

153 # Ensure that lint sees 'struct cpu' containing a fully declared
154 # embedded 'struct machcpu'
155 #
156 LINTFLAGS       += -D_MACHDEP -I../../i86pc

158 #
159 #       Default build targets.
160 #
161 .KEEP_STATE:

163 def: $(DEF_DEPS)

165 all: $(ALL_DEPS)

167 clean: $(CLEAN_DEPS)

169 clobber: $(CLOBBER_DEPS)

171 lint: $(LINT_DEPS)

173 clean.lint: $(CLEAN_LINT_DEPS)

175 install: $(INSTALL_DEPS)

177 MAPFILE_32 = $(MAPFILE)
178 MAPFILE_64 = $(MAPFILE).amd64

180 MAPFILE_NAME = $(MAPFILE_$(CLASS))

182 $(UNIX_BIN):    $(UNIX_O) $(MODSTUBS_O) $(MAPFILE_NAME) \
183                 $(GENLIB) $(DTRACESTUBS) $(DBOOT_O)
184         $(LD) -dy -b -o $@ -e dboot_image -znointerp -M $(MAPFILE_NAME) \
185             $(UNIX_O) $(DBOOT_O) $(MODSTUBS_O) $(LIBOPTS) \
186             $(DTRACESTUBS)
187         $(MBH_PATCH) $(UNIX_BIN)
188         $(CTFMERGE_MODULE)
188         $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
189         $(POST_PROCESS)
```



```
191 $(UNIX_O):      $(OBJECTS) $(OBJS_DIR)/vers.o
192         $(LD) -r -o $@ $(OBJECTS) $(OBJS_DIR)/vers.o

194 $(DBOOT_BIN):   $(DBOOT_OBJS_DIR) $(DBOOT_OBJECTS) dboot/Mapfile.dboot
195         $(LD) -dn -e _start -M dboot/Mapfile.dboot \
196                 -o $(DBOOT_BIN) $(DBOOT_OBJECTS)

198 $(DBOOT_O):     $(DBOOT_BIN)
199         @echo " .data"                                  > $(DBOOT_S)
200         @echo " .globl  dboot_image"                    >> $(DBOOT_S)
201         @echo "dboot_image:"                            >> $(DBOOT_S)
202         $(ELFEXTRACT) $(DBOOT_BIN)                      >> $(DBOOT_S)
203         $(COMPILE.s) -o $(DBOOT_O) $(DBOOT_S)

205 $(DBOOT_OBJS_DIR):
206         -@mkdir -p $@ 2> /dev/null

208 #
209 # dboot is built as an intermediate target in dboot.o, so just make
210 # dboot.o the dependency here.
211 #
212 $(MULTIBOOT):   $(DBOOT_O)
213         $(CP)   $(DBOOT_BIN) $(MULTIBOOT)
214         $(POST_PROCESS)

216 #
217 #       Special rules for generating assym.h for inclusion in assembly files.
218 #
219 $(DSF_DIR)/$(OBJS_DIR)/assym.h $(DSF_DIR)/$(OBJS_DIR)/kdi_assym.h:      FRC
220         @cd $(DSF_DIR); $(MAKE) all.targ

222 #
223 #       The global lint target builds the kernel lint library (llib-lunix.ln)
224 #       which is equivalent to a lint of /unix.o. Then all kernel modules for
225 #       this architecture are linted against the kernel lint library.
226 #
227 #       Note:   lint errors in the kernel lint library will be repeated for
228 #               each module. It is important that the kernel lint library
229 #               be clean to keep the textual output to a reasonable level.
230 #

232 $(LINT_LIB):    $(LINT_LIB_DIR) $(LINTS)
233         @pwd
234         @-$(ECHO) "\n$(UNIX): (library construction):"
235         @$(LINT) -o$(UNIX) $(LINTFLAGS) $(LINTS)
236         @$(MV) $(@F) $@

238 $(DBOOT_LINT_LIB):      $(LINT_LIB_DIR) $(DBOOT_LINTS)
239         @pwd
240         @-$(ECHO) "\n$(DBOOT): (library construction):"
241         @$(LINT) -o$(DBOOT) $(DBOOT_LINTFLAGS) $(DBOOT_LINTS)
242         @$(MV) $(@F) $@

244 lintlib:        $(LINT_DEPS)

246 #
247 #       Include common targets.
248 #
249 include $(UTSBASE)/i86pc/Makefile.targ
```

```
**********************************************************
    6121 Fri Apr 17 12:29:32 2015
new/usr/src/uts/i86xpv/unix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#       This makefile drives the production of unix (and unix.o).
#
#       i86xpv implementation architecture dependent
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
UNIX            = unix
DBOOT           = dboot

OBJECTS         = $(SPECIAL_OBJS:%=$(OBJS_DIR)/%) \
                  $(CORE_OBJS:%=$(OBJS_DIR)/%) \
                  $(KRTLD_OBJS:%=$(OBJS_DIR)/%) \
                  $(MACH_NOT_YET_KMODS:%=$(OBJS_DIR)/%)

LINTS           = $(SPECIAL_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(CORE_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(KRTLD_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(MACH_NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln) \
                  $(LINTS_DIR)/vers.ln \
                  $(LINTS_DIR)/modstubs.ln

ROOTMODULE      = $(ROOT_PSM_KERN_DIR)/$(UNIX)

UNIX_BIN        = $(OBJS_DIR)/$(UNIX)

LIBS            = $(GENLIB)
```



```
GENUNIX         = genunix
GENUNIX_DIR     = ../../intel/$(GENUNIX)

LIBOPTS         = -L $(GENUNIX_DIR)/$(OBJS_DIR) -l $(GENUNIX)

CTFEXTRAOBJS    = $(OBJS_DIR)/vers.o

DBOOT_OBJS_DIR  = dboot/$(OBJS_DIR)
DBOOT_OBJECTS   = $(DBOOT_OBJS:%=$(DBOOT_OBJS_DIR)/%)
DBOOT_BIN       = $(DBOOT_OBJS_DIR)/$(DBOOT)
DBOOT_O         = $(OBJS_DIR)/$(DBOOT).o
DBOOT_S         = $(DBOOT_O:%.o=%.s)
DBOOT_LINTS     = $(DBOOT_OBJS:%.o=$(DBOOT_OBJS_DIR)/%.ln)
DBOOT_LINT      = $(LINT_$(MACH)_$(CLASS))

#
#       Include common rules.
#
include $(UTSBASE)/i86xpv/Makefile.i86xpv

#
#       Define targets
#
ALL_TARGET      = $(UNIX_BIN)
LINT_TARGET     = $(LINT_LIB) $(DBOOT_LINT_LIB)
INSTALL_TARGET  = $(UNIX_BIN) $(ROOTMODULE)

#
#       This is UNIX_DIR. Use a short path.
#
UNIX_DIR        = .

#
#       Overrides
#
CLEANFILES      +=              \
        $(UNIX_O) $(MODSTUBS_O) \
        $(OBJS_DIR)/vers.c $(OBJS_DIR)/vers.o \
        $(OBJS_DIR)/dtracestubs.s \
        $(DTRACESTUBS_O) $(DTRACESTUBS)

CLEANFILES      +=              \
        $(DBOOT_O) $(DBOOT_S)   \
        $(DBOOT_OBJECTS)        \
        $(DBOOT_BIN)

CLEANFILES      +=              \
        $(OBJS_DIR)/fb_swtch_src.o \
        $(OBJS_DIR)/fb_swtch_src \
        $(OBJS_DIR)/fb_swtch.s

CLEANFILES      += \
        $(ZLIB_OBJS:%.o=$(OBJS_DIR)/%.o) \
        $(ZLIB_OBJS:%.o=$(OBJS_DIR)/%.ln)

CLOBBERFILES    = $(CLEANFILES) $(UNIX_BIN)
CLEANLINTFILES  += $(LINT_LIB) $(DBOOT_LINT_LIB) $(DBOOT_LINTS)

# instr_size needs a special header
$(OBJS_DIR)/instr_size.o :=     EXTRA_OPTIONS   = -I$(SRC)/common/dis/i386
$(OBJS_DIR)/instr_size.ln :=    EXTRA_OPTIONS   = -I$(SRC)/common/dis/i386

CFLAGS += -DDIS_MEM

#
# For now, disable these lint checks; maintainers should endeavor
```

```
126 # to investigate and remove these for maximum lint coverage.
127 # Please do not carry these forward to new Makefiles.
128 #
129 LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
130 LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
131 LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
132 LINTTAGS        += -erroff=E_STATIC_UNUSED
133 LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
134 LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

136 CERRWARN        += -_gcc=-Wno-parentheses
137 CERRWARN        += -_gcc=-Wno-uninitialized
138 CERRWARN        += -_gcc=-Wno-char-subscripts
139 CERRWARN        += -_gcc=-Wno-unused-variable
140 CERRWARN        += -_gcc=-Wno-unused-function
141 CERRWARN        += -_gcc=-Wno-unused-label
142 CERRWARN        += -_gcc=-Wno-type-limits
143 CERRWARN        += -_gcc=-Wno-clobbered
144 CERRWARN        += -_gcc=-Wno-unused-value
145 CERRWARN        += -_gcc=-Wno-empty-body

147 # Ensure that lint sees 'struct cpu' containing a fully declared
148 # embedded 'struct machcpu'
149 #
150 LINTFLAGS       += -D_MACHDEP -I../../i86pc

152 #
153 #       Default build targets.
154 #
155 .KEEP_STATE:

157 def: $(DEF_DEPS)

159 all: $(ALL_DEPS)

161 clean: $(CLEAN_DEPS)

163 clobber: $(CLOBBER_DEPS)

165 lint: $(LINT_DEPS)

167 clean.lint: $(CLEAN_LINT_DEPS)

169 install: $(INSTALL_DEPS)

171 MAPFILE_32 = $(MAPFILE)
172 MAPFILE_64 = $(MAPFILE).amd64

174 MAPFILE_NAME = $(MAPFILE_$(CLASS))

176 $(UNIX_BIN):    $(UNIX_O) $(MODSTUBS_O) $(MAPFILE_NAME) \
177                 $(GENLIB) $(DTRACESTUBS) $(DBOOT_O)
178         $(LD) -dy -b -o $@ -e dboot_image -znointerp -M $(MAPFILE_NAME) \
179             $(UNIX_O) $(DBOOT_O) $(MODSTUBS_O) $(LIBOPTS) \
180             $(DTRACESTUBS)
181         $(CTFMERGE_MODULE)
181         $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
182         $(POST_PROCESS)

184 $(UNIX_O):      $(OBJECTS) $(OBJS_DIR)/vers.o
185         $(LD) -r -o $@ $(OBJECTS) $(OBJS_DIR)/vers.o

187 $(DBOOT_BIN):   $(DBOOT_OBJS_DIR) $(DBOOT_OBJECTS) dboot/Mapfile.dboot
188         $(LD) -dn -e _start -M dboot/Mapfile.dboot \
189                 -o $(DBOOT_BIN) $(DBOOT_OBJECTS)
```



```
191 $(DBOOT_O):     $(DBOOT_BIN)
192         @echo " .data"                                  > $(DBOOT_S)
193         @echo " .globl  dboot_image"                    >> $(DBOOT_S)
194         @echo "dboot_image:"                           >> $(DBOOT_S)
195         $(ELFEXTRACT) $(DBOOT_BIN)                      >> $(DBOOT_S)
196         $(COMPILE.s) -o $(DBOOT_O) $(DBOOT_S)

198 $(DBOOT_OBJS_DIR):
199         -@mkdir -p $@ 2> /dev/null

201 #
202 #       Special rules for generating assym.h for inclusion in assembly files.
203 #
204 $(DSF_DIR)/$(OBJS_DIR)/assym.h $(DSF_DIR)/$(OBJS_DIR)/kdi_assym.h:      FRC
205         @cd $(DSF_DIR); $(MAKE) all.targ

207 #
208 #       The global lint target builds the kernel lint library (llib-lunix.ln)
209 #       which is equivalent to a lint of /unix.o. Then all kernel modules for
210 #       this architecture are linted against the kernel lint library.
211 #
212 #       Note:   lint errors in the kernel lint library will be repeated for
213 #               each module. It is important that the kernel lint library
214 #               be clean to keep the textual output to a reasonable level.
215 #

217 $(LINT_LIB):    $(LINT_LIB_DIR) $(LINTS)
218         @pwd
219         @-$(ECHO) "\n$(UNIX): (library construction):"
220         @$(LINT) -o$(UNIX) $(LINTFLAGS) $(LINTS)
221         @$(MV) $(@F) $@

223 $(DBOOT_LINT_LIB):      $(LINT_LIB_DIR) $(DBOOT_LINTS)
224         @pwd
225         @-$(ECHO) "\n$(DBOOT): (library construction):"
226         @$(LINT) -o$(DBOOT) $(DBOOT_LINTFLAGS) $(DBOOT_LINTS)
227         @$(MV) $(@F) $@

229 lintlib:        $(LINT_DEPS)

231 #
232 #       Include common targets.
233 #
234 include $(UTSBASE)/i86xpv/Makefile.targ
```

```
**********************************************************
   20222 Fri Apr 17 12:29:33 2015
new/usr/src/uts/intel/dtrace/fbt.c
1730 DTrace should ignore type information from modules with cth_parlabel mismat
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
**********************************************************
______unchanged_portion_omitted_

 577 /*ARGSUSED*/
 578 static void
 579 fbt_getargdesc(void *arg, dtrace_id_t id, void *parg, dtrace_argdesc_t *desc)
 580 {
 581         fbt_probe_t *fbt = parg;
 582         struct modctl *ctl = fbt->fbtp_ctl;
 583         struct module *mp = ctl->mod_mp;
 584         ctf_file_t *fp = NULL, *pfp;
 585         ctf_funcinfo_t f;
 586         int error;
 587         ctf_id_t argv[32], type;
 588         int argc = sizeof (argv) / sizeof (ctf_id_t);
 589         const char *parent;

 591         if (!ctl->mod_loaded || (ctl->mod_loadcnt != fbt->fbtp_loadcnt))
 592                 goto err;

 594         if (fbt->fbtp_roffset != 0 && desc->dtargd_ndx == 0) {
 595                 (void) strcpy(desc->dtargd_native, "int");
 596                 return;
 597         }

 599         if ((fp = ctf_modopen(mp, &error)) == NULL) {
 600                 /*
 601                  * We have no CTF information for this module -- and therefore
 602                  * no args[] information.
 603                  */
 604                 goto err;
 605         }

 607         /*
 608          * If we have a parent container, we must manually import it.
 609          */
 610         if ((parent = ctf_parent_name(fp)) != NULL) {
 611                 struct modctl *mp = &modules;
 612                 struct modctl *mod = NULL;

 614                 /*
 615                  * We must iterate over all modules to find the module that
 616                  * is our parent.
 617                  */
 618                 do {
 619                         if (strcmp(mp->mod_modname, parent) == 0) {
 620                                 mod = mp;
 621                                 break;
 622                         }
 623                 } while ((mp = mp->mod_next) != &modules);

 625                 if (mod == NULL)
 626                         goto err;

 628                 if ((pfp = ctf_modopen(mod->mod_mp, &error)) == NULL) {
 629                         goto err;
 630                 }

 632                 /*
 633                  * If the parent module does not have the label we expect,
```



```
 634                  * ignore it and fail to avoid presenting non-sensical data.
 635                  */
 636                 if (ctf_label_info(pfp, ctf_parent_label(fp),
 637                     NULL) == CTF_ERR) {
 638                         ctf_close(pfp);
 639                         goto err;
 640                 }

 642 #endif /* ! codereview */
 643                 if (ctf_import(fp, pfp) != 0) {
 644                         ctf_close(pfp);
 645                         goto err;
 646                 }

 648                 ctf_close(pfp);
 649         }

 651         if (ctf_func_info(fp, fbt->fbtp_symndx, &f) == CTF_ERR)
 652                 goto err;

 654         if (fbt->fbtp_roffset != 0) {
 655                 if (desc->dtargd_ndx > 1)
 656                         goto err;

 658                 ASSERT(desc->dtargd_ndx == 1);
 659                 type = f.ctc_return;
 660         } else {
 661                 if (desc->dtargd_ndx + 1 > f.ctc_argc)
 662                         goto err;

 664                 if (ctf_func_args(fp, fbt->fbtp_symndx, argc, argv) == CTF_ERR)
 665                         goto err;

 667                 type = argv[desc->dtargd_ndx];
 668         }

 670         if (ctf_type_name(fp, type, desc->dtargd_native,
 671             DTRACE_ARGTYPELEN) != NULL) {
 672                 ctf_close(fp);
 673                 return;
 674         }
 675 err:
 676         if (fp != NULL)
 677                 ctf_close(fp);

 679         desc->dtargd_ndx = DTRACE_ARGNONE;
 680 }

 682 static dtrace_pattr_t fbt_attr = {
 683 { DTRACE_STABILITY_EVOLVING, DTRACE_STABILITY_EVOLVING, DTRACE_CLASS_ISA },
 684 { DTRACE_STABILITY_PRIVATE, DTRACE_STABILITY_PRIVATE, DTRACE_CLASS_UNKNOWN },
 685 { DTRACE_STABILITY_PRIVATE, DTRACE_STABILITY_PRIVATE, DTRACE_CLASS_UNKNOWN },
 686 { DTRACE_STABILITY_EVOLVING, DTRACE_STABILITY_EVOLVING, DTRACE_CLASS_ISA },
 687 { DTRACE_STABILITY_PRIVATE, DTRACE_STABILITY_PRIVATE, DTRACE_CLASS_ISA },
 688 };

 690 static dtrace_pops_t fbt_pops = {
 691         NULL,
 692         fbt_provide_module,
 693         fbt_enable,
 694         fbt_disable,
 695         fbt_suspend,
 696         fbt_resume,
 697         fbt_getargdesc,
 698         NULL,
 699         NULL,
```

```
700         fbt_destroy
701 };

703 static void
704 fbt_cleanup(dev_info_t *devi)
705 {
706         dtrace_invop_remove(fbt_invop);
707         ddi_remove_minor_node(devi, NULL);
708         kmem_free(fbt_probetab, fbt_probetab_size * sizeof (fbt_probe_t *));
709         fbt_probetab = NULL;
710         fbt_probetab_mask = 0;
711 }

713 static int
714 fbt_attach(dev_info_t *devi, ddi_attach_cmd_t cmd)
715 {
716         switch (cmd) {
717         case DDI_ATTACH:
718                 break;
719         case DDI_RESUME:
720                 return (DDI_SUCCESS);
721         default:
722                 return (DDI_FAILURE);
723         }

725         if (fbt_probetab_size == 0)
726                 fbt_probetab_size = FBT_PROBETAB_SIZE;

728         fbt_probetab_mask = fbt_probetab_size - 1;
729         fbt_probetab =
730             kmem_zalloc(fbt_probetab_size * sizeof (fbt_probe_t *), KM_SLEEP);

732         dtrace_invop_add(fbt_invop);

734         if (ddi_create_minor_node(devi, "fbt", S_IFCHR, 0,
735             DDI_PSEUDO, NULL) == DDI_FAILURE ||
736             dtrace_register("fbt", &fbt_attr, DTRACE_PRIV_KERNEL, NULL,
737             &fbt_pops, NULL, &fbt_id) != 0) {
738                 fbt_cleanup(devi);
739                 return (DDI_FAILURE);
740         }

742         ddi_report_dev(devi);
743         fbt_devi = devi;

745         return (DDI_SUCCESS);
746 }

748 static int
749 fbt_detach(dev_info_t *devi, ddi_detach_cmd_t cmd)
750 {
751         switch (cmd) {
752         case DDI_DETACH:
753                 break;
754         case DDI_SUSPEND:
755                 return (DDI_SUCCESS);
756         default:
757                 return (DDI_FAILURE);
758         }

760         if (dtrace_unregister(fbt_id) != 0)
761                 return (DDI_FAILURE);

763         fbt_cleanup(devi);

765         return (DDI_SUCCESS);
```



```
766 }

768 /*ARGSUSED*/
769 static int
770 fbt_info(dev_info_t *dip, ddi_info_cmd_t infocmd, void *arg, void **result)
771 {
772         int error;

774         switch (infocmd) {
775         case DDI_INFO_DEVT2DEVINFO:
776                 *result = (void *)fbt_devi;
777                 error = DDI_SUCCESS;
778                 break;
779         case DDI_INFO_DEVT2INSTANCE:
780                 *result = (void *)0;
781                 error = DDI_SUCCESS;
782                 break;
783         default:
784                 error = DDI_FAILURE;
785         }
786         return (error);
787 }

789 /*ARGSUSED*/
790 static int
791 fbt_open(dev_t *devp, int flag, int otyp, cred_t *cred_p)
792 {
793         return (0);
794 }

796 static struct cb_ops fbt_cb_ops = {
797         fbt_open,               /* open */
798         nodev,                  /* close */
799         nulldev,                /* strategy */
800         nulldev,                /* print */
801         nodev,                  /* dump */
802         nodev,                  /* read */
803         nodev,                  /* write */
804         nodev,                  /* ioctl */
805         nodev,                  /* devmap */
806         nodev,                  /* mmap */
807         nodev,                  /* segmap */
808         nochpoll,               /* poll */
809         ddi_prop_op,            /* cb_prop_op */
810         0,                      /* streamtab  */
811         D_NEW | D_MP            /* Driver compatibility flag */
812 };

814 static struct dev_ops fbt_ops = {
815         DEVO_REV,               /* devo_rev */
816         0,                      /* refcnt */
817         fbt_info,               /* get_dev_info */
818         nulldev,                /* identify */
819         nulldev,                /* probe */
820         fbt_attach,             /* attach */
821         fbt_detach,             /* detach */
822         nodev,                  /* reset */
823         &fbt_cb_ops,            /* driver operations */
824         NULL,                   /* bus operations */
825         nodev,                  /* dev power */
826         ddi_quiesce_not_needed,         /* quiesce */
827 };

829 /*
830  * Module linkage information for the kernel.
831  */
```

```
832 static struct modldrv modldrv = {
833         &mod_driverops,         /* module type (this is a pseudo driver) */
834         "Function Boundary Tracing",    /* name of module */
835         &fbt_ops,               /* driver ops */
836 };

838 static struct modlinkage modlinkage = {
839         MODREV_1,
840         (void *)&modldrv,
841         NULL
842 };

844 int
845 _init(void)
846 {
847         return (mod_install(&modlinkage));
848 }

850 int
851 _info(struct modinfo *modinfop)
852 {
853         return (mod_info(&modlinkage, modinfop));
854 }

856 int
857 _fini(void)
858 {
859         return (mod_remove(&modlinkage));
860 }
```

```
**********************************************************
    3479 Fri Apr 17 12:29:33 2015
new/usr/src/uts/intel/genunix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
```

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

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
#       This makefile drives the production of the generic
#       unix kernel module.
#
#       x86 implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = genunix
GENUNIX         = $(OBJS_DIR)/$(MODULE)

OBJECTS         = $(GENUNIX_OBJS:%=$(OBJS_DIR)/%) \
                  $(NOT_YET_KMODS:%=$(OBJS_DIR)/%)

LINTS           = $(GENUNIX_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln)

ROOTMODULE      = $(ROOT_KERN_DIR)/$(MODULE)

LIBGEN          = $(OBJS_DIR)/libgenunix.so
LIBSTUBS        = $(GENSTUBS_OBJS:%=$(OBJS_DIR)/%)

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel
```



```
#
#       Define targets
#
ALL_TARGET      = $(LIBGEN)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(GENUNIX) $(ROOTMODULE)

#
#       Overrides
#
CLOBBERFILES    += $(GENUNIX)
CLEANFILES      += $(LIBSTUBS) $(LIBGEN)
BINARY          =

#
# Non-patch genunix builds merge a version of the ip module called ipctf.  This
# is to ensure that the common network-related types are included in genunix and
# can thus be uniquified out of other modules.  We don't want to do this for
# patch builds, since we can't guarantee that ip and genunix will be in the same
# patch.
#
IPCTF_TARGET    = $(IPCTF)
$(PATCH_BUILD)IPCTF_TARGET =

CPPFLAGS        += -I$(SRC)/common
CPPFLAGS        += -I$(SRC)/uts/common/fs/zfs

CPPFLAGS        += -I$(UTSBASE)/i86pc

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-value
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-switch
CERRWARN        += -_gcc=-Wno-type-limits
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-clobbered
CERRWARN        += -_gcc=-Wno-empty-body

#
# Ensure that lint sees 'struct cpu' containing a fully declared
# embedded 'struct machcpu'
#
LINTFLAGS       += -D_MACHDEP -I../../i86pc

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)
```

```
116 all:            $(ALL_DEPS)

118 clean:          $(CLEAN_DEPS)

120 clobber:        $(CLOBBER_DEPS)

122 lint:           $(LINT_DEPS)

124 modlintlib:     $(MODLINTLIB_DEPS)

126 clean.lint:     $(CLEAN_LINT_DEPS)

128 install:        $(INSTALL_DEPS)

130 $(LIBGEN):      $(GENUNIX) $(LIBSTUBS)
131         $(BUILD.SO) $(GENUNIX) $(LIBSTUBS)

133 $(GENUNIX): $(OBJECTS)
143 $(IPCTF_TARGET) ipctf_target: FRC
144         @cd $(IPDRV_DIR); pwd; $(MAKE) ipctf.$(OBJS_DIR)
145         @pwd

147 $(GENUNIX): $(IPCTF_TARGET) $(OBJECTS)
134         $(LD) -r $(LDFLAGS) -o $@ $(OBJECTS)
135         $(CTFMERGE_MODULE)
149         $(CTFMERGE_GENUNIX_MERGE)
136         $(POST_PROCESS)

138 #
139 #       Include common targets.
140 #
141 include $(UTSBASE)/intel/Makefile.targ

143 #
144 #       Software workarounds for hardware "features".
145 #
146 include $(UTSBASE)/i86pc/Makefile.workarounds

148 ALL_DEFS += $(WORKAROUND_DEFS)

150 #
151 # Override.
152 #
153 $(MODULE).lint := GEN_LINT_LIB =
```

```
**********************************************************
    3291 Fri Apr 17 12:29:34 2015
new/usr/src/uts/intel/ip/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#
#       This makefile drives the production of the ip driver
#       kernel module.
#
#       intel implementation architecture dependent
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = ip
OBJECTS         = $(IP_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(IP_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
ROOTLINK        = $(ROOT_STRMOD_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/inet/ip

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY) $(SRC_CONFILE)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOTLINK) $(ROOT_CONFFILE)

CINLINEFLAGS = -xinline=tcp_set_ws_value
```



```
CFLAGS          += $(CINLINEFLAGS)

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

#
# To get the BPF header files included by ipnet.h
#
INC_PATH        += -I$(UTSBASE)/common/io/bpf

#
# Depends on md5 and swrand (for SCTP). SCTP needs to depend on
# swrand as it needs random numbers early on during boot before
# kCF subsystem can load swrand.
#
LDFLAGS         += -dy -Nmisc/md5 -Ncrypto/swrand -Nmisc/hook -Nmisc/neti

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS) $(SISCHECK_DEPS)

clean:          $(CLEAN_DEPS) $(SISCLEAN_DEPS)

# Need to clobber all build types due to ipctf.a
clobber:        $(CLOBBER_DEPS) $(SISCLEAN_DEPS) \
                clobber.obj32 clobber.obj64 \
                clobber.debug32 clobber.debug64

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS) $(SISCHECK_DEPS)

$(ROOTLINK):    $(ROOT_STRMOD_DIR) $(ROOTMODULE)
        -$(RM) $@; ln $(ROOTMODULE) $@

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ
```

```
#
# The ip CTF data is merged into the genunix module because these types are
# complex and heavily shared.  The genunix build will execute one of the
# rules below to create an archive, ipctf.a, containing the ip objects.  The
# real ip will be uniquified against genunix later in the build, and will
# emerge containing very few types.
#
$(OBJS_DIR)/ipctf.a: $(OBJECTS)
        -$(RM) $@
        $(AR) -r $@ $(OBJECTS)

$(OBJECTS): $(OBJS_DIR)

CLOBBERFILES += $(OBJS_DIR)/ipctf.a

ipctf.obj32: FRC
        @BUILD_TYPE=OBJ32 VERSION='$(VERSION)' $(MAKE) obj32/ipctf.a

ipctf.debug32: FRC
        @BUILD_TYPE=DBG32 VERSION='$(VERSION)' $(MAKE) debug32/ipctf.a

ipctf.obj64: FRC
        @BUILD_TYPE=OBJ64 VERSION='$(VERSION)' $(MAKE) obj64/ipctf.a

ipctf.debug64: FRC
        @BUILD_TYPE=DBG64 VERSION='$(VERSION)' $(MAKE) debug64/ipctf.a
```

```
**********************************************************
   50310 Fri Apr 17 12:29:35 2015
new/usr/src/uts/sparc/dtrace/fbt.c
1730 DTrace should ignore type information from modules with cth_parlabel mismat
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
**********************************************************
______unchanged_portion_omitted_

1606 /*ARGSUSED*/
1607 static void
1608 fbt_getargdesc(void *arg, dtrace_id_t id, void *parg, dtrace_argdesc_t *desc)
1609 {
1610         fbt_probe_t *fbt = parg;
1611         struct modctl *ctl = fbt->fbtp_ctl;
1612         struct module *mp = ctl->mod_mp;
1613         ctf_file_t *fp = NULL, *pfp;
1614         ctf_funcinfo_t f;
1615         int error;
1616         ctf_id_t argv[32], type;
1617         int argc = sizeof (argv) / sizeof (ctf_id_t);
1618         const char *parent;

1620         if (!ctl->mod_loaded || (ctl->mod_loadcnt != fbt->fbtp_loadcnt))
1621                 goto err;

1623         if (fbt->fbtp_return && desc->dtargd_ndx == 0) {
1624                 (void) strcpy(desc->dtargd_native, "int");
1625                 return;
1626         }

1628         if ((fp = ctf_modopen(mp, &error)) == NULL) {
1629                 /*
1630                  * We have no CTF information for this module -- and therefore
1631                  * no args[] information.
1632                  */
1633                 goto err;
1634         }

1636         /*
1637          * If we have a parent container, we must manually import it.
1638          */
1639         if ((parent = ctf_parent_name(fp)) != NULL) {
1640                 struct modctl *mp = &modules;
1641                 struct modctl *mod = NULL;

1643                 /*
1644                  * We must iterate over all modules to find the module that
1645                  * is our parent.
1646                  */
1647                 do {
1648                         if (strcmp(mp->mod_modname, parent) == 0) {
1649                                 mod = mp;
1650                                 break;
1651                         }
1652                 } while ((mp = mp->mod_next) != &modules);

1654                 if (mod == NULL)
1655                         goto err;

1657                 if ((pfp = ctf_modopen(mod->mod_mp, &error)) == NULL)
1658                         goto err;

1660                 /*
1661                  * If the parent module does not have the label we expect,
1662                  * ignore it and fail to avoid presenting non-sensical data.
```



```
1663                  */
1664                 if (ctf_label_info(pfp, ctf_parent_label(fp),
1665                     NULL) == CTF_ERR) {
1666                         ctf_close(pfp);
1667                         goto err;
1668                 }

1670 #endif /* ! codereview */
1671                 if (ctf_import(fp, pfp) != 0) {
1672                         ctf_close(pfp);
1673                         goto err;
1674                 }

1676                 ctf_close(pfp);
1677         }

1679         if (ctf_func_info(fp, fbt->fbtp_symndx, &f) == CTF_ERR)
1680                 goto err;

1682         if (fbt->fbtp_return) {
1683                 if (desc->dtargd_ndx > 1)
1684                         goto err;

1686                 ASSERT(desc->dtargd_ndx == 1);
1687                 type = f.ctc_return;
1688         } else {
1689                 if (desc->dtargd_ndx + 1 > f.ctc_argc)
1690                         goto err;

1692                 if (ctf_func_args(fp, fbt->fbtp_symndx, argc, argv) == CTF_ERR)
1693                         goto err;

1695                 type = argv[desc->dtargd_ndx];
1696         }

1698         if (ctf_type_name(fp, type, desc->dtargd_native,
1699             DTRACE_ARGTYPELEN) != NULL) {
1700                 ctf_close(fp);
1701                 return;
1702         }
1703 err:
1704         if (fp != NULL)
1705                 ctf_close(fp);

1707         desc->dtargd_ndx = DTRACE_ARGNONE;
1708 }

1710 static dtrace_pattr_t fbt_attr = {
1711 { DTRACE_STABILITY_EVOLVING, DTRACE_STABILITY_EVOLVING, DTRACE_CLASS_ISA },
1712 { DTRACE_STABILITY_PRIVATE, DTRACE_STABILITY_PRIVATE, DTRACE_CLASS_UNKNOWN },
1713 { DTRACE_STABILITY_PRIVATE, DTRACE_STABILITY_PRIVATE, DTRACE_CLASS_UNKNOWN },
1714 { DTRACE_STABILITY_EVOLVING, DTRACE_STABILITY_EVOLVING, DTRACE_CLASS_ISA },
1715 { DTRACE_STABILITY_PRIVATE, DTRACE_STABILITY_PRIVATE, DTRACE_CLASS_ISA },
1716 };

1718 static dtrace_pops_t fbt_pops = {
1719         NULL,
1720         fbt_provide_module,
1721         fbt_enable,
1722         fbt_disable,
1723         fbt_suspend,
1724         fbt_resume,
1725         fbt_getargdesc,
1726         NULL,
1727         NULL,
1728         fbt_destroy
```

```
1729 };

1731 static int
1732 fbt_attach(dev_info_t *devi, ddi_attach_cmd_t cmd)
1733 {
1734         switch (cmd) {
1735         case DDI_ATTACH:
1736                 break;
1737         case DDI_RESUME:
1738                 return (DDI_SUCCESS);
1739         default:
1740                 return (DDI_FAILURE);
1741         }

1743         if (ddi_create_minor_node(devi, "fbt", S_IFCHR, 0,
1744             DDI_PSEUDO, NULL) == DDI_FAILURE ||
1745             dtrace_register("fbt", &fbt_attr, DTRACE_PRIV_KERNEL, NULL,
1746             &fbt_pops, NULL, &fbt_id) != 0) {
1747                 ddi_remove_minor_node(devi, NULL);
1748                 return (DDI_FAILURE);
1749         }

1751         ddi_report_dev(devi);
1752         fbt_devi = devi;
1753         return (DDI_SUCCESS);
1754 }

1756 static int
1757 fbt_detach(dev_info_t *devi, ddi_detach_cmd_t cmd)
1758 {
1759         switch (cmd) {
1760         case DDI_DETACH:
1761                 break;
1762         case DDI_SUSPEND:
1763                 return (DDI_SUCCESS);
1764         default:
1765                 return (DDI_FAILURE);
1766         }

1768         if (dtrace_unregister(fbt_id) != 0)
1769                 return (DDI_FAILURE);

1771         ddi_remove_minor_node(devi, NULL);
1772         return (DDI_SUCCESS);
1773 }

1775 /*ARGSUSED*/
1776 static int
1777 fbt_info(dev_info_t *dip, ddi_info_cmd_t infocmd, void *arg, void **result)
1778 {
1779         int error;

1781         switch (infocmd) {
1782         case DDI_INFO_DEVT2DEVINFO:
1783                 *result = (void *)fbt_devi;
1784                 error = DDI_SUCCESS;
1785                 break;
1786         case DDI_INFO_DEVT2INSTANCE:
1787                 *result = (void *)0;
1788                 error = DDI_SUCCESS;
1789                 break;
1790         default:
1791                 error = DDI_FAILURE;
1792         }
1793         return (error);
1794 }
```



```
1796 /*ARGSUSED*/
1797 static int
1798 fbt_open(dev_t *devp, int flag, int otyp, cred_t *cred_p)
1799 {
1800         return (0);
1801 }

1803 static struct cb_ops fbt_cb_ops = {
1804         fbt_open,               /* open */
1805         nodev,                  /* close */
1806         nulldev,                /* strategy */
1807         nulldev,                /* print */
1808         nodev,                  /* dump */
1809         nodev,                  /* read */
1810         nodev,                  /* write */
1811         nodev,                  /* ioctl */
1812         nodev,                  /* devmap */
1813         nodev,                  /* mmap */
1814         nodev,                  /* segmap */
1815         nochpoll,               /* poll */
1816         ddi_prop_op,            /* cb_prop_op */
1817         0,                      /* streamtab  */
1818         D_NEW | D_MP            /* Driver compatibility flag */
1819 };

1821 static struct dev_ops fbt_ops = {
1822         DEVO_REV,               /* devo_rev */
1823         0,                      /* refcnt */
1824         fbt_info,               /* get_dev_info */
1825         nulldev,                /* identify */
1826         nulldev,                /* probe */
1827         fbt_attach,             /* attach */
1828         fbt_detach,             /* detach */
1829         nodev,                  /* reset */
1830         &fbt_cb_ops,            /* driver operations */
1831         NULL,                   /* bus operations */
1832         nodev,                  /* dev power */
1833         ddi_quiesce_not_needed,         /* quiesce */
1834 };

1836 /*
1837  * Module linkage information for the kernel.
1838  */
1839 static struct modldrv modldrv = {
1840         &mod_driverops,         /* module type (this is a pseudo driver) */
1841         "Function Boundary Tracing",    /* name of module */
1842         &fbt_ops,               /* driver ops */
1843 };

1845 static struct modlinkage modlinkage = {
1846         MODREV_1,
1847         (void *)&modldrv,
1848         NULL
1849 };

1851 int
1852 _init(void)
1853 {
1854         return (mod_install(&modlinkage));
1855 }

1857 int
1858 _info(struct modinfo *modinfop)
1859 {
1860         return (mod_info(&modlinkage, modinfop));
```

```
1861 }

1863 int
1864 _fini(void)
1865 {
1866         return (mod_remove(&modlinkage));
1867 }
```

```
**********************************************************
    3182 Fri Apr 17 12:29:35 2015
new/usr/src/uts/sparc/ip/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#
#       This makefile drives the production of the ip driver
#       kernel module.
#
#       sparc architecture dependent
#
#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = ip
OBJECTS         = $(IP_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(IP_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
ROOTLINK        = $(ROOT_STRMOD_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/inet/ip

#
#       Include common rules.
#
include $(UTSBASE)/sparc/Makefile.sparc

#
#       Define targets
#
ALL_TARGET      = $(BINARY) $(SRC_CONFFILE)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOTLINK) $(ROOT_CONFFILE)

#
# lint pass one enforcement
```



```
#
CFLAGS          += $(CCVERBOSE)
CFLAGS          += -xinline=tcp_set_ws_value
#
# To get the BPF header files included by ipnet.h
#
INC_PATH        += -I$(UTSBASE)/common/io/bpf

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

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

#
# Depends on md5 and swrand (for SCTP). SCTP needs to depend on
# swrand as it needs random numbers early on during boot before
# kCF subsystem can load swrand.
#
LDFLAGS         += -dy -Nmisc/md5 -Ncrypto/swrand -Nmisc/hook -Nmisc/neti

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS) $(SISCHECK_DEPS)

clean:          $(CLEAN_DEPS) $(SISCLEAN_DEPS)

clobber:        $(CLOBBER_DEPS) $(SISCLEAN_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS) $(SISCHECK_DEPS)

$(ROOTLINK):    $(ROOT_STRMOD_DIR) $(ROOTMODULE)
        -$(RM) $@; ln $(ROOTMODULE) $@

#
#       Include common targets.
#
include $(UTSBASE)/sparc/Makefile.targ

#
# The ip CTF data is merged into the genunix module because these types are
```

```
126 # complex and heavily shared.  The genunix build will execute one of the
127 # rules below to create an archive, ipctf.a, containing the ip objects.  The
128 # real ip will be uniquified against genunix later in the build, and will
129 # emerge containing very few types.
130 #
131 $(OBJS_DIR)/ipctf.a: $(OBJECTS)
132         -$(RM) $@
133         $(AR) -r $@ $(OBJECTS)

124 $(OBJECTS): $(OBJS_DIR)

137 CLOBBERFILES += $(OBJS_DIR)/ipctf.a

139 ipctf.obj64: FRC
140         @BUILD_TYPE=OBJ64 VERSION='$(VERSION)' $(MAKE) obj64/ipctf.a

142 ipctf.debug64: FRC
143         @BUILD_TYPE=DBG64 VERSION='$(VERSION)' $(MAKE) debug64/ipctf.a
```

```
**********************************************************
    3713 Fri Apr 17 12:29:36 2015
new/usr/src/uts/sun4u/genunix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
#       This makefile drives the production of the generic
#       unix kernel module.
#
#       sparc implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = genunix
GENUNIX         = $(OBJS_DIR)/$(MODULE)

OBJECTS         = $(GENUNIX_OBJS:%=$(OBJS_DIR)/%) \
                  $(NOT_YET_KMODS:%=$(OBJS_DIR)/%)

LINTS           = $(GENUNIX_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln)

ROOTMODULE      = $(ROOT_PSM_KERN_DIR)/$(MODULE)

PLATFORM        = sun4u
LIBGEN          = $(OBJS_DIR)/libgenunix.so
LIBSTUBS        = $(GENSTUBS_OBJS:%=$(OBJS_DIR)/%)

#
#       Include common rules.
#
```



```
include $(UTSBASE)/sparc/Makefile.sparc

#
#       Define targets
#
ALL_TARGET      = $(LIBGEN)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(GENUNIX) $(ROOTMODULE)

#
#       Override defaults
#
CLEANFILES      += $(LIBSTUBS) $(LIBGEN)

LINT_LIB_DIR    = $(UTSBASE)/$(PLATFORM)/lint-libs/$(OBJS_DIR)
LINT_LIB        = $(LINT_LIB_DIR)/llib-lunix.ln
GEN_LINT_LIB    =

BINARY          =

CLOBBERFILES    += $(GENUNIX)

#
# Non-patch genunix builds merge a version of the ip module called ipctf.  This
# is to ensure that the common network-related types are included in genunix and
# can thus be uniquified out of other modules.  We don't want to do this for
# patch builds, since we can't guarantee that ip and genunix will be in the same
# patch.
#
IPCTF_TARGET    = $(IPCTF)
$(PATCH_BUILD)IPCTF_TARGET =

#
# lint pass one enforcement
#
CFLAGS += $(CCVERBOSE)
CPPFLAGS += -I$(SRC)/common
CPPFLAGS += -I$(SRC)/uts/common/fs/zfs

INC_PATH +=  -I$(UTSBASE)/sun4

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-value
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-switch
CERRWARN        += -_gcc=-Wno-type-limits
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-clobbered
CERRWARN        += -_gcc=-Wno-empty-body

#
# Ensure that lint sees 'struct cpu' containing a fully declared
```

```
116 # embedded 'struct machcpu'
117 #
118 LINTFLAGS       += -D_MACHDEP -I../../sun4 -I../../$(PLATFORM) -I../../sfmmu

120 #       Default build targets.
121 #
122 .KEEP_STATE:

124 .PARALLEL:      $(LIBSTUBS)

126 def:            $(DEF_DEPS)

128 all:            $(ALL_DEPS)

130 clean:          $(CLEAN_DEPS)

132 clobber:        $(CLOBBER_DEPS)

134 lint:           $(LINT_DEPS)

136 modlintlib:     $(MODLINTLIB_DEPS)

138 clean.lint:     $(CLEAN_LINT_DEPS)

140 install:        $(INSTALL_DEPS)

142 install_h:


145 $(LIBGEN):      $(GENUNIX) $(LIBSTUBS)
146         $(BUILD.SO) $(GENUNIX) $(LIBSTUBS)

148 $(GENUNIX): $(OBJECTS)
158 $(IPCTF_TARGET) ipctf_target: FRC
159         @cd $(IPDRV_DIR); pwd; $(MAKE) ipctf.$(OBJS_DIR)
160         @pwd

162 $(GENUNIX): $(IPCTF_TARGET) $(OBJECTS)
149         $(LD) -r $(LDFLAGS) -o $@ $(OBJECTS)
150         $(CTFMERGE_MODULE)
164         $(CTFMERGE_GENUNIX_MERGE)
151         $(POST_PROCESS)

153 $(OBJECTS): $(OBJS_DIR)

155 #
156 #       Include common targets.
157 #
158 include $(UTSBASE)/sparc/Makefile.targ

160 #
161 #       Include sun4u workarounds.
162 #
163 include $(UTSBASE)/sun4u/Makefile.workarounds

165 ALL_DEFS +=     $(WORKAROUND_DEFS)
```

```
**********************************************************
    5046 Fri Apr 17 12:29:36 2015
new/usr/src/uts/sun4u/opl/unix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
#       This makefile drives the production of unix (and unix.o).
#
#       sun4u opl implementation architecture dependent
#
# uts/sun4u/opl/unix/Makefile
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../../..

#
#       Define the module and object file sets.
#
UNIX            = unix
OBJECTS         = $(SPECIAL_OBJS:%=$(OBJS_DIR)/%) \
                  $(CORE_OBJS:%=$(OBJS_DIR)/%) \
                  $(MACH_NOT_YET_KMODS:%=$(OBJS_DIR)/%)
LINTS           = $(SPECIAL_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(CORE_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(MACH_NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln) \
                  $(LINTS_DIR)/vers.ln \
                  $(LINTS_DIR)/modstubs.ln

KRTLD_MAPFILE   = $(UTSBASE)/sparc/krtld/mapfile
KRTLD_OBJECTS   = $(KRTLD_OBJS:%=$(OBJS_DIR)/%)
KRTLD_O         = $(OBJS_DIR)/krtld.o

ROOTMODULE      = $(ROOT_OPL_KERN_DIR)/$(UNIX)
UNIX_BIN        = $(OBJS_DIR)/$(UNIX)

LIBS            = $(GENLIB) $(PLATLIB) $(CPULIB)
```



```
GENUNIX         = genunix
GENUNIX_DIR     = ../../$(GENUNIX)
GENOPTS         = -L $(GENUNIX_DIR)/$(OBJS_DIR) -l $(GENUNIX)

CPU_DIR         = .
CPUOPTS         = -L $(CPU_DIR)/$(OBJS_DIR) -l $(CPUNAME)

PLAT_DIR        = ../../platmod
PLATOPTS        = -L $(PLAT_DIR)/$(OBJS_DIR) -l $(PLATMOD)

LIBOPTS         = $(GENOPTS) $(PLATOPTS) $(CPUOPTS)

CTFEXTRAOBJS    = $(OBJS_DIR)/vers.o

#
#       Include common rules.
#
include $(UTSBASE)/sun4u/opl/Makefile.opl

#
#       Define targets
#
ALL_TARGET      = $(UNIX_BIN)
LINT_TARGET     = $(LINT_LIB)
INSTALL_TARGET  = $(UNIX_BIN) $(ROOTMODULE)

#
#       Overrides
#
ALL_BUILDS      = $(ALL_BUILDSONLY64)
DEF_BUILDS      = $(DEF_BUILDSONLY64)
SYM_BUILDS      = $(DEF_BUILDSONLY64)
CLEANLINTFILES  += $(LINT32_FILES)

#
#       This is UNIX_DIR. Use a short path.
#
UNIX_DIR        = .

#
#       Overrides
#
CLEANFILES      += $(UNIX_O) $(MODSTUBS_O) $(KRTLD_O) $(KRTLD_OBJECTS) \
                   $(OBJS_DIR)/vers.c $(OBJS_DIR)/vers.o \
                   $(CPU_OBJ) $(CPULIB) \
                   $(DTRACESTUBS_O) $(DTRACESTUBS)

CLOBBERFILES    = $(CLEANFILES) $(UNIX_BIN)
CLEANLINTFILES  += $(LINT_LIB)

#
# lint pass one enforcement
# Turn on doubleword alignment for 64 bit counter timer registers
#
CFLAGS += $(CCVERBOSE) -dalign

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-char-subscripts
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-type-limits
CERRWARN        += -_gcc=-Wno-clobbered
CERRWARN        += -_gcc=-Wno-empty-body
```

```
126 CERRWARN        += -_gcc=-Wno-unused-value
127 CERRWARN        += -_gcc=-Wno-switch

129 #
130 #       Default build targets.
131 #
132 .KEEP_STATE:

134 all:            $(ALL_DEPS)

136 def:            $(DEF_DEPS)

138 clean:          $(CLEAN_DEPS)

140 clobber:        $(CLOBBER_DEPS)

142 lint:           $(LINT_DEPS)

144 clean.lint:     $(CLEAN_LINT_DEPS)

146 install:        $(INSTALL_DEPS)

148 symcheck:       $(SYM_DEPS)

150 $(UNIX_BIN):    $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(MAPFILE) $(LIBS) \
151                 $(DTRACESTUBS)
152         $(LD) -dy -b -o $@ -e _start -M $(MAPFILE) \
153             $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)
154         $(CTFMERGE_MODULE)
154         $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
155         $(POST_PROCESS)

157 symcheck.targ:  $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBS) $(DTRACESTUBS)
158         $(LD) -dy -b -o $(SYM_MOD) -M $(MAPFILE) \
159         $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)

161 $(UNIX_O):      $(OBJECTS) $(OBJS_DIR)/vers.o
162         $(LD) -r -o $@ $(OBJECTS) $(OBJS_DIR)/vers.o

164 $(KRTLD_O):     $(KRTLD_OBJECTS)
165         $(LD) -r -o $@ -M$(KRTLD_MAPFILE) $(KRTLD_OBJECTS)

167 #
168 #       CPU_OBJ now comprises of 2 object files which come from sun4 common
169 #       and from architecture dependent code.  OBJS_DIR is prepended where
170 #       CPU_OBJ is defined to allow for building multiple CPU_OBJ's
171 #
172 $(CPULIB):      $(CPU_OBJ)
173         $(LD) -o $@ -G -h 'cpu/$$CPU' $(CPU_OBJ)

175 #
176 #       The global lint target builds the kernel lint library (llib-lunix.ln)
177 #       which is equivalent to a lint of /unix.o. Then all kernel modules for
178 #       this architecture are linted against the kernel lint library.
179 #
180 #       Note:   lint errors in the kernel lint library will be repeated for
181 #               each module. It is important that the kernel lint library
182 #               be clean to keep the textual output to a reasonable level.
183 #

185 $(LINT_LIB):    $(LINT_LIB_DIR) $(LINTS)
186         @-$(ECHO) "\n$(UNIX): (library construction):"
187         @$(LINT) -o $(UNIX) $(LINTFLAGS) $(LINTS)
188         @$(MV) $(@F) $@

190 lintlib:        $(LINT_DEPS)
```



```
192 #
193 #       Include common targets.
194 #
195 include $(UTSBASE)/sun4u/opl/Makefile.targ
```

```
*********************************************************
    4986 Fri Apr 17 12:29:37 2015
new/usr/src/uts/sun4u/serengeti/unix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
*********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

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

#
#       This makefile drives the production of unix (and unix.o).
#
#       sun4u serengeti implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../../..

#
#       Define the module and object file sets.
#
UNIX            = unix
OBJECTS         = $(SPECIAL_OBJS:%=$(OBJS_DIR)/%) \
                  $(CORE_OBJS:%=$(OBJS_DIR)/%) \
                  $(MACH_NOT_YET_KMODS:%=$(OBJS_DIR)/%)
LINTS           = $(SPECIAL_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(CORE_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(MACH_NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln) \
                  $(LINTS_DIR)/vers.ln \
                  $(LINTS_DIR)/modstubs.ln

KRTLD_MAPFILE   = $(UTSBASE)/sparc/krtld/mapfile
KRTLD_OBJECTS   = $(KRTLD_OBJS:%=$(OBJS_DIR)/%)
KRTLD_O         = $(OBJS_DIR)/krtld.o

ROOTMODULE      = $(ROOT_SERENGETI_KERN_DIR)/$(UNIX)
UNIX_BIN        = $(OBJS_DIR)/$(UNIX)

LIBS            = $(GENLIB) $(PLATLIB) $(CPULIB)
```



```
GENUNIX         = genunix
GENUNIX_DIR     = ../../$(GENUNIX)
GENOPTS         = -L $(GENUNIX_DIR)/$(OBJS_DIR) -l $(GENUNIX)

CPU_DIR         = .
CPUOPTS         = -L $(CPU_DIR)/$(OBJS_DIR) -l $(CPUNAME)

PLAT_DIR        = ../../platmod
PLATOPTS        = -L $(PLAT_DIR)/$(OBJS_DIR) -l $(PLATMOD)

LIBOPTS         = $(GENOPTS) $(PLATOPTS) $(CPUOPTS)

CTFEXTRAOBJS    = $(OBJS_DIR)/vers.o

#
#       Include common rules.
#
include $(UTSBASE)/sun4u/serengeti/Makefile.serengeti

#
#       Define targets
#
ALL_TARGET      = $(UNIX_BIN)
LINT_TARGET     = $(LINT_LIB)
INSTALL_TARGET  = $(UNIX_BIN) $(ROOTMODULE)

#
#       Overrides
#
ALL_BUILDS      = $(ALL_BUILDSONLY64)
DEF_BUILDS      = $(DEF_BUILDSONLY64)
SYM_BUILDS      = $(DEF_BUILDSONLY64)
CLEANLINTFILES  += $(LINT32_FILES)

#
#       This is UNIX_DIR. Use a short path.
#
UNIX_DIR        = .

#
#       Overrides
#
CLEANFILES      += $(UNIX_O) $(MODSTUBS_O) $(KRTLD_O)  $(KRTLD_OBJECTS) \
                   $(OBJS_DIR)/vers.c  $(OBJS_DIR)/vers.o \
                   $(CPU_OBJ) $(CPULIB) \
                   $(DTRACESTUBS_O) $(DTRACESTUBS)

CLOBBERFILES    = $(CLEANFILES) $(UNIX_BIN)
CLEANLINTFILES  += $(LINT_LIB)

#
# lint pass one enforcement
# Turn on doubleword alignment for 64 bit counter timer registers
#
CFLAGS += $(CCVERBOSE) -dalign

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-char-subscripts
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-type-limits
CERRWARN        += -_gcc=-Wno-clobbered
CERRWARN        += -_gcc=-Wno-empty-body
CERRWARN        += -_gcc=-Wno-unused-value
```

```
126 CERRWARN        += -_gcc=-Wno-switch

128 #
129 #       Default build targets.
130 #
131 .KEEP_STATE:

133 def:            $(DEF_DEPS)

135 all:            $(ALL_DEPS)

137 clean:          $(CLEAN_DEPS)

139 clobber:        $(CLOBBER_DEPS)

141 lint:           $(LINT_DEPS)

143 clean.lint:     $(CLEAN_LINT_DEPS)

145 install:        $(INSTALL_DEPS)

147 symcheck:       $(SYM_DEPS)

149 $(UNIX_BIN):    $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(MAPFILE) $(LIBS) \
150                 $(DTRACESTUBS)
151         $(LD) -dy -b -o $@ -e _start -M $(MAPFILE) \
152             $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)
153         $(CTFMERGE_MODULE)
153         $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
154         $(POST_PROCESS)

156 symcheck.targ:  $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBS)
157         $(LD) -dy -b -o $(SYM_MOD) -M $(MAPFILE) \
158         $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)

160 $(UNIX_O):      $(OBJECTS) $(OBJS_DIR)/vers.o
161         $(LD) -r -o $@ $(OBJECTS) $(OBJS_DIR)/vers.o

163 $(KRTLD_O):     $(KRTLD_OBJECTS)
164         $(LD) -r -o $@ -M $(KRTLD_MAPFILE) $(KRTLD_OBJECTS)

166 #
167 #       CPU_OBJ now comprises of 2 object files which come from sun4 common
168 #       and from architecture dependent code.  OBJS_DIR is prepended where
169 #       CPU_OBJ is defined to allow for building multiple CPU_OBJ's
170 #
171 $(CPULIB):      $(CPU_OBJ)
172         $(BUILD.SO) $(CPU_OBJ)

174 #
175 #       The global lint target builds the kernel lint library (llib-lunix.ln)
176 #       which is equivalent to a lint of /unix.o. Then all kernel modules for
177 #       this architecture are linted against the kernel lint library.
178 #
179 #       Note:   lint errors in the kernel lint library will be repeated for
180 #               each module. It is important that the kernel lint library
181 #               be clean to keep the textual output to a reasonable level.
182 #

184 $(LINT_LIB):    $(LINT_LIB_DIR) $(LINTS)
185         @-$(ECHO) "\n$(UNIX): (library construction):"
186         @$(LINT) -o $(UNIX) $(LINTFLAGS) $(LINTS)
187         @$(MV) $(@F) $@

189 lintlib:        $(LINT_DEPS)
```



```
191 #
192 #       Include common targets.
193 #
194 include $(UTSBASE)/sun4u/serengeti/Makefile.targ
```

```
**********************************************************
    5033 Fri Apr 17 12:29:37 2015
new/usr/src/uts/sun4u/starcat/unix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
#       This makefile drives the production of unix (and unix.o).
#
#       sun4u starcat implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../../..

#
#       Define the module and object file sets.
#
UNIX            = unix
OBJECTS         = $(SPECIAL_OBJS:%=$(OBJS_DIR)/%) \
                  $(CORE_OBJS:%=$(OBJS_DIR)/%) \
                  $(MACH_NOT_YET_KMODS:%=$(OBJS_DIR)/%)
LINTS           = $(SPECIAL_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(CORE_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(MACH_NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln) \
                  $(LINTS_DIR)/vers.ln \
                  $(LINTS_DIR)/modstubs.ln

KRTLD_MAPFILE   = $(UTSBASE)/sparc/krtld/mapfile
KRTLD_OBJECTS   = $(KRTLD_OBJS:%=$(OBJS_DIR)/%)
KRTLD_O         = $(OBJS_DIR)/krtld.o

ROOTMODULE      = $(ROOT_STARCAT_KERN_DIR)/$(UNIX)
UNIX_BIN        = $(OBJS_DIR)/$(UNIX)

LIBS            = $(GENLIB) $(PLATLIB) $(CPULIB)
```



```
GENUNIX         = genunix
GENUNIX_DIR     = ../../$(GENUNIX)
GENOPTS         = -L $(GENUNIX_DIR)/$(OBJS_DIR) -l $(GENUNIX)

CPU_DIR         = .
CPUOPTS         = -L $(CPU_DIR)/$(OBJS_DIR) -l $(CPUNAME)

PLAT_DIR        = ../../platmod
PLATOPTS        = -L $(PLAT_DIR)/$(OBJS_DIR) -l $(PLATMOD)

LIBOPTS         = $(GENOPTS) $(PLATOPTS) $(CPUOPTS)

CTFEXTRAOBJS    = $(OBJS_DIR)/vers.o

#
#       Include common rules.
#
include $(UTSBASE)/sun4u/starcat/Makefile.starcat

#
#       Define targets
#
ALL_TARGET      = $(UNIX_BIN)
LINT_TARGET     = $(LINT_LIB)
INSTALL_TARGET  = $(UNIX_BIN) $(ROOTMODULE)

#
#       Overrides
#
ALL_BUILDS      = $(ALL_BUILDSONLY64)
DEF_BUILDS      = $(DEF_BUILDSONLY64)
SYM_BUILDS      = $(DEF_BUILDSONLY64)
CLEANLINTFILES  += $(LINT32_FILES)

#
#       This is UNIX_DIR. Use a short path.
#
UNIX_DIR        = .

#
#       Overrides
#
CLEANFILES      += $(UNIX_O) $(MODSTUBS_O) $(KRTLD_O) $(KRTLD_OBJECTS) \
                   $(OBJS_DIR)/vers.c $(OBJS_DIR)/vers.o \
                   $(CPU_OBJ) $(CPULIB) \
                   $(DTRACESTUBS_O) $(DTRACESTUBS)

CLOBBERFILES    = $(CLEANFILES) $(UNIX_BIN)
CLEANLINTFILES  += $(LINT_LIB)

#
# lint pass one enforcement
# Turn on doubleword alignment for 64 bit counter timer registers
#
CFLAGS += $(CCVERBOSE) -dalign

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-char-subscripts
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-type-limits
CERRWARN        += -_gcc=-Wno-clobbered
CERRWARN        += -_gcc=-Wno-empty-body
CERRWARN        += -_gcc=-Wno-unused-value
```

```
126 CERRWARN        += -_gcc=-Wno-switch

128 #
129 #       Default build targets.
130 #
131 .KEEP_STATE:

133 def:            $(DEF_DEPS)

135 all:            $(ALL_DEPS)

137 clean:          $(CLEAN_DEPS)

139 clobber:        $(CLOBBER_DEPS)

141 lint:           $(LINT_DEPS)

143 clean.lint:     $(CLEAN_LINT_DEPS)

145 install:        $(INSTALL_DEPS)

147 symcheck:       $(SYM_DEPS)

149 $(UNIX_BIN):    $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(MAPFILE) $(LIBS) \
150                 $(DTRACESTUBS)
151         $(LD) -dy -b -o $@ -e _start -M $(MAPFILE) \
152             $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)
153         $(CTFMERGE_MODULE)
154         $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
155         $(POST_PROCESS)
156         $(CHK4UBINARY)

157 symcheck.targ:  $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBS) $(DTRACESTUBS)
158         $(LD) -dy -b -o $(SYM_MOD) -M $(MAPFILE) \
159         $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)

161 $(UNIX_O):      $(OBJECTS) $(OBJS_DIR)/vers.o
162         $(LD) -r -o $@ $(OBJECTS) $(OBJS_DIR)/vers.o

164 $(KRTLD_O):     $(KRTLD_OBJECTS)
165         $(LD) -r -o $@ -M$(KRTLD_MAPFILE) $(KRTLD_OBJECTS)

167 #
168 #       CPU_OBJ now comprises of 2 object files which come from sun4 common
169 #       and from architecture dependent code.  OBJS_DIR is prepended where
170 #       CPU_DIR is defined to allow for building multiple CPU_OBJ's
171 #
172 $(CPULIB):      $(CPU_OBJ)
173         $(BUILD.SO) $(CPU_OBJ)

175 #
176 #       The global lint target builds the kernel lint library (llib-lunix.ln)
177 #       which is equivalent to a lint of /unix.o. Then all kernel modules for
178 #       this architecture are linted against the kernel lint library.
179 #
180 #       Note:   lint errors in the kernel lint library will be repeated for
181 #               each module. It is important that the kernel lint library
182 #               be clean to keep the textual output to a reasonable level.
183 #

185 $(LINT_LIB):    $(LINT_LIB_DIR) $(LINTS)
186         @-$(ECHO) "\n$(UNIX): (library construction):"
187         @$(LINT) -o $(UNIX) $(LINTFLAGS) $(LINTS)
188         @$(MV) $(@F) $@

190 lintlib:        $(LINT_DEPS)
```



```
192 #
193 #       Include common targets.
194 #
195 include $(UTSBASE)/sun4u/starcat/Makefile.targ
```

```
**********************************************************
    4977 Fri Apr 17 12:29:38 2015
new/usr/src/uts/sun4u/starfire/unix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
#       This makefile drives the production of unix (and unix.o).
#
#       sun4u starfire implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../../..

#
#       Define the module and object file sets.
#
UNIX            = unix
OBJECTS         = $(SPECIAL_OBJS:%=$(OBJS_DIR)/%) \
                  $(CORE_OBJS:%=$(OBJS_DIR)/%) \
                  $(MACH_NOT_YET_KMODS:%=$(OBJS_DIR)/%)
LINTS           = $(SPECIAL_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(CORE_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(MACH_NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln) \
                  $(LINTS_DIR)/vers.ln \
                  $(LINTS_DIR)/modstubs.ln

KRTLD_MAPFILE   = $(UTSBASE)/sparc/krtld/mapfile
KRTLD_OBJECTS   = $(KRTLD_OBJS:%=$(OBJS_DIR)/%)
KRTLD_O         = $(OBJS_DIR)/krtld.o

ROOTMODULE      = $(ROOT_STARFIRE_KERN_DIR)/$(UNIX)
UNIX32_LINK     = $(ROOT_STARFIRE_KERN_DIR_32)/$(UNIX)
UNIX_BIN        = $(OBJS_DIR)/$(UNIX)

LIBS            = $(GENLIB) $(PLATLIB) $(CPULIB)
```



```
GENUNIX         = genunix
GENUNIX_DIR     = ../../$(GENUNIX)
GENOPTS         = -L $(GENUNIX_DIR)/$(OBJS_DIR) -l $(GENUNIX)

CPU_DIR         = .
CPUOPTS         = -L $(CPU_DIR)/$(OBJS_DIR) -l $(CPUNAME)

PLAT_DIR        = ../../platmod
PLATOPTS        = -L $(PLAT_DIR)/$(OBJS_DIR) -l $(PLATMOD)

LIBOPTS         = $(GENOPTS) $(PLATOPTS) $(CPUOPTS)

CTFEXTRAOBJS    = $(OBJS_DIR)/vers.o

#
#       Include common rules.
#
include $(UTSBASE)/sun4u/starfire/Makefile.starfire

#
#       Define targets
#
ALL_TARGET      = $(UNIX_BIN)
LINT_TARGET     = $(LINT_LIB)
INSTALL_TARGET  = $(UNIX_BIN) $(ROOTMODULE) $(UNIX32_LINK)

#
#       This is UNIX_DIR. Use a short path.
#
UNIX_DIR        = .

#
#       Overrides
#
CLEANFILES      += $(UNIX_O) $(MODSTUBS_O) $(KRTLD_O) $(KRTLD_OBJECTS) \
                   $(OBJS_DIR)/vers.c  $(OBJS_DIR)/vers.o \
                   $(CPU_OBJ) $(CPULIB) \
                   $(DTRACESTUBS_O) $(DTRACESTUBS)

CLOBBERFILES    = $(CLEANFILES) $(UNIX_BIN)
CLEANLINTFILES  += $(LINT_LIB)

#
# lint pass one enforcement
# Turn on doubleword alignment for 64 bit counter timer registers
#
CFLAGS += $(CCVERBOSE) -dalign

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-char-subscripts
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-type-limits
CERRWARN        += -_gcc=-Wno-clobbered
CERRWARN        += -_gcc=-Wno-empty-body
CERRWARN        += -_gcc=-Wno-unused-value
CERRWARN        += -_gcc=-Wno-switch

#
#       Default build targets.
#
.KEEP_STATE:
```

```
126 def:            $(DEF_DEPS)

128 all:            $(ALL_DEPS)

130 clean:          $(CLEAN_DEPS)

132 clobber:        $(CLOBBER_DEPS)

134 lint:           $(LINT_DEPS)

136 clean.lint:     $(CLEAN_LINT_DEPS)

138 install:        $(INSTALL_DEPS)

140 $(UNIX_BIN):    $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(MAPFILE) $(LIBS) \
141                 $(DTRACESTUBS)
142         $(LD) -dy -b -o $@ -e _start -M $(MAPFILE) \
143             $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)
144         $(CTFMERGE_MODULE)
145         $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
146         $(POST_PROCESS)
147         $(CHK4UBINARY)

148 $(UNIX32_LINK): $(ROOT_PSM_KERN_DIR_32) $(UNIX_BIN)
149         -$(RM) $@; ln -s $(SUBDIR64)/$(UNIX) $@

151 symcheck:       $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBS)
152         $(LD) -dy -b -o $(SYM_MOD) -M $(MAPFILE) \
153         $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)

155 $(UNIX_O):      $(OBJECTS) $(OBJS_DIR)/vers.o
156         $(LD) -r -o $@ $(OBJECTS) $(OBJS_DIR)/vers.o

158 $(KRTLD_O):     $(KRTLD_OBJECTS)
159         $(LD) -r -o $@ -M$(KRTLD_MAPFILE) $(KRTLD_OBJECTS)

161 #
162 #       CPU_OBJ now comprises of 2 object files which come from sun4 common
163 #       and from architecture dependent code.  OBJS_DIR is prepended where
164 #       CPU_OBJ is defined to allow for building multiple CPU_OBJ's
165 #
166 $(CPULIB):      $(CPU_OBJ)
167         $(BUILD.SO) $(CPU_OBJ)

169 #
170 #       The global lint target builds the kernel lint library (llib-lunix.ln)
171 #       which is equivalent to a lint of /unix.o. Then all kernel modules for
172 #       this architecture are linted against the kernel lint library.
173 #
174 #       Note:   lint errors in the kernel lint library will be repeated for
175 #               each module. It is important that the kernel lint library
176 #               be clean to keep the textual output to a reasonable level.
177 #

179 $(LINT_LIB):    $(LINT_LIB_DIR) $(LINTS)
180         @-$(ECHO) "\n$(UNIX): (library construction):"
181         @$(LINT) -o $(UNIX) $(LINTFLAGS) $(LINTS)
182         @$(MV) $(@F) $@

184 lintlib:        $(LINT_DEPS)

186 #
187 #       Include common targets.
188 #
189 include $(UTSBASE)/sun4u/starfire/Makefile.targ
```

```
**********************************************************
    5512 Fri Apr 17 12:29:38 2015
new/usr/src/uts/sun4u/unix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
#       This makefile drives the production of unix (and unix.o).
#
#       sun4u implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
UNIX            = unix
OBJECTS         = $(SPECIAL_OBJS:%=$(OBJS_DIR)/%) \
                  $(CORE_OBJS:%=$(OBJS_DIR)/%) \
                  $(MACH_NOT_YET_KMODS:%=$(OBJS_DIR)/%)
LINTS           = $(SPECIAL_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(CORE_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(KRTLD_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(MACH_NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln) \
                  $(LINTS_DIR)/vers.ln \
                  $(LINTS_DIR)/modstubs.ln

KRTLD_MAPFILE   = $(UTSBASE)/sparc/krtld/mapfile
KRTLD_OBJECTS   = $(KRTLD_OBJS:%=$(OBJS_DIR)/%)
KRTLD_O         = $(OBJS_DIR)/krtld.o

ROOTMODULE      = $(ROOT_PSM_KERN_DIR)/$(UNIX)
UNIX32_LINK     = $(ROOT_PSM_KERN_DIR_32)/$(UNIX)
UNIX_BIN        = $(OBJS_DIR)/$(UNIX)
#UNIX_GLOM      = $(OBJS_DIR)/unix.glom
```



```
LIBS            = $(GENLIB) $(PLATLIB) $(CPULIB)

GENUNIX         = genunix
GENUNIX_DIR     = ../$(GENUNIX)
GENOPTS         = -L $(GENUNIX_DIR)/$(OBJS_DIR) -l $(GENUNIX)

CPU_DIR         = .
CPUOPTS         = -L $(CPU_DIR)/$(OBJS_DIR) -l $(CPUNAME)

PLAT_DIR        = ../platmod
PLATOPTS        = -L $(PLAT_DIR)/$(OBJS_DIR) -l $(PLATMOD)

LIBOPTS         = $(GENOPTS) $(PLATOPTS) $(CPUOPTS)

CTFEXTRAOBJS    = $(OBJS_DIR)/vers.o

#
#       Include common rules.
#
include $(UTSBASE)/sun4u/Makefile.sun4u

#
#       Define targets
#
ALL_TARGET      = $(UNIX_BIN)
LINT_TARGET     = $(LINT_LIB)
INSTALL_TARGET  = $(UNIX_BIN) $(ROOTMODULE) $(UNIX32_LINK)

#
#       This is UNIX_DIR. Use a short path.
#
UNIX_DIR        = .

#
#       Overrides
#
CLEANFILES      += $(UNIX_O) $(MODSTUBS_O) $(KRTLD_O) $(KRTLD_OBJECTS) \
                   $(OBJS_DIR)/vers.c $(OBJS_DIR)/vers.o \
                   $(CPU_OBJ) $(CPULIB) \
                   $(DTRACESTUBS_O) $(DTRACESTUBS)

CLOBBERFILES    = $(CLEANFILES) $(UNIX_BIN)
CLEANLINTFILES  += $(LINT_LIB)

#
# lint pass one enforcement
# Turn on doubleword alignment for 64 bit counter timer registers
#
CFLAGS += $(CCVERBOSE) -dalign

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-char-subscripts
```

```
126 CERRWARN        += -_gcc=-Wno-unused-variable
127 CERRWARN        += -_gcc=-Wno-unused-function
128 CERRWARN        += -_gcc=-Wno-unused-label
129 CERRWARN        += -_gcc=-Wno-type-limits
130 CERRWARN        += -_gcc=-Wno-clobbered
131 CERRWARN        += -_gcc=-Wno-empty-body
132 CERRWARN        += -_gcc=-Wno-unused-value
133 CERRWARN        += -_gcc=-Wno-switch

135 #
136 #       Default build targets.
137 #
138 .KEEP_STATE:

140 def:            $(DEF_DEPS)

142 all:            $(ALL_DEPS)

144 clean:          $(CLEAN_DEPS)

146 clobber:        $(CLOBBER_DEPS)

148 lint:           $(LINT_DEPS)

150 clean.lint:     $(CLEAN_LINT_DEPS)

152 install:        $(INSTALL_DEPS)


155 $(UNIX_BIN):    $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(MAPFILE) $(LIBS) \
156                 $(DTRACESTUBS)
157         $(LD) -dy -b -o $@ -e _start -M $(MAPFILE) \
158             $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)
159         $(CTFMERGE_MODULE)
159         $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
160         $(POST_PROCESS)
161         $(CHK4UBINARY)

163 $(UNIX32_LINK): $(ROOT_PSM_KERN_DIR_32) $(UNIX_BIN)
164         -$(RM) $@; ln -s $(SUBDIR64)/$(UNIX) $@

166 symcheck:       $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBS)
167         $(LD) -dy -b -o $(SYM_MOD) -M $(MAPFILE) \
168         $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)

170 $(UNIX_O):      $(OBJECTS) $(OBJS_DIR)/vers.o
171         $(LD) -r -o $@ $(OBJECTS) $(OBJS_DIR)/vers.o

173 $(KRTLD_O):     $(KRTLD_OBJECTS)
174         $(LD) -r -o $@ -M$(KRTLD_MAPFILE) $(KRTLD_OBJECTS)

176 #
177 #       CPU_OBJ now comprises of 2 object files which come from sun4 common
178 #       and from architecture dependent code.  OBJS_DIR is prepended where
179 #       CPU_OBJ is defined to allow for building multiple CPU_OBJ's
180 #
181 $(CPULIB):      $(CPU_OBJ)
182         $(BUILD.SO) $(CPU_OBJ)

184 $(PLATLIB):
185         ?@(cd $(PLAT_DIR); pwd; $(MAKE) all.targ)
186         ?@pwd

188 #
189 #       The global lint target builds the kernel lint library (llib-lunix.ln)
190 #       which is equivalent to a lint of /unix.o. Then all kernel modules for
```



```
191 #       this architecture are linted against the kernel lint library.
192 #
193 #       Note:   lint errors in the kernel lint library will be repeated for
194 #               each module. It is important that the kernel lint library
195 #               be clean to keep the textual output to a reasonable level.
196 #

198 $(LINT_LIB):    $(LINT_LIB_DIR) $(LINTS)
199         @pwd
200         @-$(ECHO) "\n$(UNIX): (library construction):"
201         @$(LINT) -o $(UNIX) $(LINTFLAGS) $(LINTS)
202         @$(MV) $(@F) $@

204 lintlib:        $(LINT_DEPS)

206 #
207 #       Include common targets.
208 #
209 include $(UTSBASE)/sun4u/Makefile.targ
```

```
*********************************************************
   12097 Fri Apr 17 12:29:39 2015
new/usr/src/uts/sun4v/Makefile.sun4v
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
*********************************************************
```

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

#
# Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
# Copyright (c) 2013 Andrew Stormont.  All rights reserved.
#
#       This makefile contains the common definitions for the sun4v unix
#       and all sun4v implementation architecture dependent modules.
#

#
#       Machine type (implementation architecture):
#
PLATFORM         = sun4v
LINKED_PLATFORMS += SUNW,Sun-Fire-T1000
LINKED_PLATFORMS += SUNW,SPARC-Enterprise-T5120
LINKED_PLATFORMS += SUNW,SPARC-Enterprise-T5220
LINKED_PLATFORMS += SUNW,T5140
LINKED_PLATFORMS += SUNW,T5240
LINKED_PLATFORMS += SUNW,T5440
LINKED_PLATFORMS += SUNW,SPARC-Enterprise-T1000
LINKED_PLATFORMS += SUNW,Sun-Blade-T6300
LINKED_PLATFORMS += SUNW,Sun-Blade-T6320
LINKED_PLATFORMS += SUNW,Netra-CP3260
LINKED_PLATFORMS += SUNW,Netra-T5220
LINKED_PLATFORMS += SUNW,USBRDT-5240
LINKED_PLATFORMS += SUNW,Netra-T5440
LINKED_PLATFORMS += SUNW,Sun-Blade-T6340
PROMIF           = ieee1275
PSMBASE          = $(UTSBASE)/../psm

#
#       uname -m value
#
UNAME_M         = $(PLATFORM)

#
# Definitions for the platform-specific /platform directories.
#
```



```
# PLATFORMS designates those sun4v machines which have no platform
# specific code.
#
# IMPLEMENTATIONS is used to designate sun4v machines which have
# platform specific modules.  All code specific to a given implementation
# resides in the appropriately named subdirectory.   This requires
# these platforms to have their own Makefiles to define ROOT_PLAT_DIRS,
# USR_PLAT_DIRS, etc.
# The number of IMPLEMENTATIONS should not grow!
#
# So if we had an implementation named 'foo', we would need the following
# Makefiles in the foo subdirectory:
#
#       sun4v/foo/Makefile
#       sun4v/foo/Makefile.foo
#       sun4v/foo/Makefile.targ
#

#
# all PLATFORMS that do not belong in the $(IMPLEMENTATIONS) list.
# This list should be empty.  A platform without platform modules
# is a plain, generic sun4v platform.
#
#IMPLEMENTED_PLATFORM   =
#PLATFORMS                = $(IMPLEMENTED_PLATFORM)

IMPLEMENTATIONS         = ontario montoya huron maramba

#ROOT_PLAT_DIRS         = $(PLATFORMS:%=$(ROOT_PLAT_DIR)/%)
#USR_PLAT_DIRS          = $(PLATFORMS:%=$(USR_PLAT_DIR)/%)

#USR_DESKTOP_DIR                = $(USR_PLAT_DIR)/$(IMPLEMENTED_PLATFORM)
#USR_DESKTOP_INC_DIR    = $(USR_DESKTOP_DIR)/include
#USR_DESKTOP_SBIN_DIR   = $(USR_DESKTOP_DIR)/sbin
#USR_DESKTOP_LIB_DIR    = $(USR_DESKTOP_DIR)/lib

#
#       Define supported builds
#
DEF_BUILDS      = $(DEF_BUILDS64)
ALL_BUILDS      = $(ALL_BUILDS64)

#
#       Everybody needs to know how to build modstubs.o and to locate unix.o
#
UNIX_DIR         = $(UTSBASE)/$(PLATFORM)/unix
GENLIB_DIR       = $(UTSBASE)/$(PLATFORM)/genunix
MODSTUBS_DIR     = $(UNIX_DIR)
DSF_DIR          = $(UTSBASE)/$(PLATFORM)/genassym
LINTS_DIR        = $(OBJS_DIR)
LINT_LIB_DIR     = $(UTSBASE)/$(PLATFORM)/lint-libs/$(OBJS_DIR)

DTRACESTUBS_O    = $(OBJS_DIR)/dtracestubs.o
DTRACESTUBS      = $(OBJS_DIR)/libdtracestubs.so

UNIX_O           = $(UNIX_DIR)/$(OBJS_DIR)/unix.o
MODSTUBS_O       = $(MODSTUBS_DIR)/$(OBJS_DIR)/modstubs.o
GENLIB           = $(GENLIB_DIR)/$(OBJS_DIR)/libgenunix.so

LINT_LIB         = $(LINT_LIB_DIR)/llib-lunix.ln
GEN_LINT_LIB     = $(LINT_LIB_DIR)/llib-lgenunix.ln

LINT64_DIRS      = $(LINT64_BUILDS:%=$(UTSBASE)/$(PLATFORM)/lint-libs/%)
LINT64_FILES     = $(LINT64_DIRS:%=%/llib-l$(MODULE).ln)

#
```

```
126 #       cpu and platform modules need to know how to build their own symcheck mo
127 #
128 PLATMOD          = platmod
129 PLATLIB          = $(PLAT_DIR)/$(OBJS_DIR)/libplatmod.so

131 CPUNAME          = cpu
132 CPULIB           = $(CPU_DIR)/$(OBJS_DIR)/libcpu.so

134 SYM_MOD          = $(OBJS_DIR)/unix.sym

136 #
137 #       Include the makefiles which define build rule templates, the
138 #       collection of files per module, and a few specific flags. Note
139 #       that order is significant, just as with an include path. The
140 #       first build rule template which matches the files name will be
141 #       used. By including these in order from most machine dependent
142 #       to most machine independent, we allow a machine dependent file
143 #       to be used in preference over a machine independent version
144 #       (Such as a machine specific optimization, which preserves the
145 #       interfaces.)
146 #
147 include $(UTSBASE)/sun4/Makefile.files
148 include $(UTSBASE)/$(PLATFORM)/Makefile.files
149 include $(UTSBASE)/sfmmu/Makefile.files
150 include $(UTSBASE)/sparc/v9/Makefile.files
151 include $(UTSBASE)/sparc/Makefile.files
152 include $(UTSBASE)/sun/Makefile.files
153 include $(SRC)/psm/promif/$(PROMIF)/common/Makefile.files
154 include $(SRC)/psm/promif/$(PROMIF)/$(PLATFORM)/Makefile.files
155 include $(UTSBASE)/common/Makefile.files

157 #
158 #       Include machine independent rules. Note that this does not imply
159 #       that the resulting module from rules in Makefile.uts is machine
160 #       independent. Only that the build rules are machine independent.
161 #
162 include $(UTSBASE)/Makefile.uts

164 CTFMERGE_GUDIR  = sun4v

164 #
165 #       machine specific optimization, override default in Makefile.master
166 #
167 CC_XARCH        = -m64 -xarch=sparcvis
168 AS_XARCH        = -xarch=v9v
169 COPTIMIZE       = -xO3
170 CCMODE          = -Xa

172 CFLAGS          = -xchip=ultra $(CCABS32) $(CCREGSYM)
173 CFLAGS          += $(CC_XARCH)
174 CFLAGS          += $(COPTIMIZE)
175 CFLAGS          += $(EXTRA_CFLAGS)
176 CFLAGS          += $(XAOPT)
177 CFLAGS          += $(INLINES) -D_ASM_INLINES
178 CFLAGS          += $(CCMODE)
179 CFLAGS          += $(SPACEFLAG)
180 CFLAGS          += $(CERRWARN)
181 CFLAGS          += $(CTF_FLAGS_$(CLASS))
182 CFLAGS          += $(C99MODE)
183 CFLAGS          += $(CCUNBOUND)
184 CFLAGS          += $(CCNOAUTOINLINE)
185 CFLAGS          += $(CCSTATICSYM)
186 CFLAGS          += $(CC32BITCALLERS)
187 CFLAGS          += $(IROPTFLAG)
188 CFLAGS          += $(CGLOBALSTATIC)
189 CFLAGS          += -xregs=no%float
```



```
190 CFLAGS          += -xstrconst
191 CFLAGS          += $(CSOURCEDEBUGFLAGS)
192 CFLAGS          += $(CUSERFLAGS)

194 CPPFLAGS        += -DGLREG

196 ASFLAGS         += $(AS_XARCH) -DGLREG

198 AS_INC_PATH     += -I$(DSF_DIR)/$(OBJS_DIR)

200 LINT_KMODS      += $(GENUNIX_KMODS)

202 LINT_DEFS       = -m64

204 #
205 #       The following must be defined for all implementations:
206 #
207 #       MAPFILE:                ld mapfile for the build of kernel/unix.
208 #       MODSTUBS:               Module stubs source file.
209 #       GENCONST_SRC:           genconst.c
210 #       OFFSETS:                offsets.in
211 #       PLATFORM_OFFSETS:       Platform specific mach_offsets.in
212 #       FDOFFSETS:              fd_offsets.in
213 #
214 MAPFILE          = $(UTSBASE)/sun4/conf/Mapfile
215 MODSTUBS         = $(UTSBASE)/sparc/ml/modstubs.s
216 GENCONST_SRC     = $(UTSBASE)/sun4/ml/genconst.c
217 OFFSETS          = $(UTSBASE)/sun4/ml/offsets.in
218 PLATFORM_OFFSETS = $(UTSBASE)/sun4v/ml/mach_offsets.in
219 FDOFFSETS        = $(UTSBASE)/sun/io/fd_offsets.in

221 #
222 #       Define the actual specific platforms
223 #

225 MACHINE_DEFS    = -D$(PLATFORM) -D_MACHDEP -DSFMMU
226 MACHINE_DEFS    += -DMAX_MEM_NODES=8

228 #
229 #       Software workarounds for hardware "features"
230 #

232 include $(UTSBASE)/$(PLATFORM)/Makefile.workarounds

234 #
235 #       Debugging level
236 #
237 #       Special knowledge of which special debugging options effect which
238 #       file is used to optimize the build if these flags are changed.
239 #
240 #       XXX: The above could possibly be done for more flags and files, but
241 #            is left as an experiment to the interested reader. Be forewarned,
242 #            that excessive use could lead to maintenance difficulties.
243 #
244 #       Note: kslice can be enabled for the sun4v, but is disabled by default
245 #             in all cases.
246 #

248 DEBUG_DEFS_OBJ64        =
249 DEBUG_DEFS_DBG64        = -DDEBUG
250 DEBUG_DEFS              = $(DEBUG_DEFS_$(BUILD_TYPE))

252 DEBUG_COND_OBJ64        = $(POUND_SIGN)
253 DEBUG_COND_DBG64        =
254 IF_DEBUG_OBJ            = $(DEBUG_COND_$(BUILD_TYPE))$(OBJS_DIR)/
```

```
256 $(IF_DEBUG_OBJ)trap.o           :=      DEBUG_DEFS      += -DTRAPDEBUG
257 $(IF_DEBUG_OBJ)mach_trap.o      :=      DEBUG_DEFS      += -DTRAPDEBUG
258 $(IF_DEBUG_OBJ)syscall_trap.o   :=      DEBUG_DEFS      += -DSYSCALLTRACE
259 $(IF_DEBUG_OBJ)clock.o          :=      DEBUG_DEFS      += -DKSLICE=0

261 IF_TRAPTRACE_OBJ = $(IF_DEBUG_OBJ)
262 # comment this out for a non-debug kernel with TRAPTRACE
263 #IF_TRAPTRACE_OBJ = $(OBJS_DIR)/

265 $(IF_TRAPTRACE_OBJ)mach_locore.o        :=      DEBUG_DEFS      += -DTRAPTRACE
266 $(IF_TRAPTRACE_OBJ)mlsetup.o            :=      DEBUG_DEFS      += -DTRAPTRACE
267 $(IF_TRAPTRACE_OBJ)syscall_trap.o       :=      DEBUG_DEFS      += -DTRAPTRACE
268 $(IF_TRAPTRACE_OBJ)startup.o            :=      DEBUG_DEFS      += -DTRAPTRACE
269 $(IF_TRAPTRACE_OBJ)mach_startup.o       :=      DEBUG_DEFS      += -DTRAPTRACE
270 $(IF_TRAPTRACE_OBJ)mp_startup.o         :=      DEBUG_DEFS      += -DTRAPTRACE
271 $(IF_TRAPTRACE_OBJ)cpu_states.o         :=      DEBUG_DEFS      += -DTRAPTRACE
272 $(IF_TRAPTRACE_OBJ)mach_cpu_states.o    :=      DEBUG_DEFS      += -DTRAPTRACE
273 $(IF_TRAPTRACE_OBJ)interrupt.o          :=      DEBUG_DEFS      += -DTRAPTRACE
274 $(IF_TRAPTRACE_OBJ)mach_interrupt.o     :=      DEBUG_DEFS      += -DTRAPTRACE
275 $(IF_TRAPTRACE_OBJ)sfmmu_asm.o          :=      DEBUG_DEFS      += -DTRAPTRACE
276 $(IF_TRAPTRACE_OBJ)trap_table.o         :=      DEBUG_DEFS      += -DTRAPTRACE
277 $(IF_TRAPTRACE_OBJ)xc.o                 :=      DEBUG_DEFS      += -DTRAPTRACE
278 $(IF_TRAPTRACE_OBJ)mach_xc.o            :=      DEBUG_DEFS      += -DTRAPTRACE
279 $(IF_TRAPTRACE_OBJ)wbuf.o               :=      DEBUG_DEFS      += -DTRAPTRACE
280 $(IF_TRAPTRACE_OBJ)trap.o               :=      DEBUG_DEFS      += -DTRAPTRACE
281 $(IF_TRAPTRACE_OBJ)mach_trap.o          :=      DEBUG_DEFS      += -DTRAPTRACE
282 $(IF_TRAPTRACE_OBJ)x_call.o             :=      DEBUG_DEFS      += -DTRAPTRACE

284 # Comment these out if you don't want dispatcher lock statistics.

286 #$(IF_DEBUG_OBJ)lock_prim.o     := DEBUG_DEFS   += -DDISP_LOCK_STATS
287 #$(IF_DEBUG_OBJ)disp.o          := DEBUG_DEFS   += -DDISP_LOCK_STATS

289 # Comment these out if you don't want dispatcher debugging

291 #$(IF_DEBUG_OBJ)lock_prim.o     := DEBUG_DEFS   += -DDISP_DEBUG

293 #
294 #       Collect the preprocessor definitions to be associated with *all*
295 #       files.
296 #
297 ALL_DEFS         = $(MACHINE_DEFS) $(WORKAROUND_DEFS) $(DEBUG_DEFS) \
298                    $(OPTION_DEFS)
299 GENCONST_DEFS    = $(MACHINE_DEFS) $(OPTION_DEFS)

301 #
302 # ----- TRANSITIONAL SECTION --------------------------------------------------
303 #

305 #
306 #       Not everything which *should* be a module is a module yet. The
307 #       following is a list of such objects which are currently part of
308 #       the base kernel but should soon become kmods.
309 #
310 MACH_NOT_YET_KMODS       = $(AUTOCONF_OBJS)

312 #
313 # ----- END OF TRANSITIONAL SECTION -------------------------------------------
314 #

316 #
317 #       The kernels modules which are "implementation architecture"
318 #       specific for this machine are enumerated below. Note that most
319 #       of these modules must exist (in one form or another) for each
320 #       architecture.
321 #
```



```
322 #       Common Drivers (usually pseudo drivers) (/kernel/drv):
323 #

325 #
326 #       Machine Specific Driver Modules (/kernel/drv):
327 #
328 DRV_KMODS       += bge
329 DRV_KMODS       += cnex
330 DRV_KMODS       += cpc
331 DRV_KMODS       += drctl
332 DRV_KMODS       += ds_pri
333 DRV_KMODS       += ds_snmp
334 DRV_KMODS       += ebus
335 DRV_KMODS       += fpc
336 DRV_KMODS       += glvc
337 DRV_KMODS       += mdesc
338 DRV_KMODS       += niumx
339 DRV_KMODS       += ntwdt
340 DRV_KMODS       += nxge
341 DRV_KMODS       += n2piupc
342 DRV_KMODS       += iospc
343 DRV_KMODS       += n2rng
344 DRV_KMODS       += px
345 DRV_KMODS       += qcn
346 DRV_KMODS       += rootnex
347 DRV_KMODS       += su
348 DRV_KMODS       += tpm
349 DRV_KMODS       += trapstat
350 DRV_KMODS       += vcc
351 DRV_KMODS       += vdc
352 DRV_KMODS       += vds
353 DRV_KMODS       += vldc
354 DRV_KMODS       += vlds
355 DRV_KMODS       += vnet
356 DRV_KMODS       += vnex
357 DRV_KMODS       += vsw

359 #
360 #       Exec Class Modules (/kernel/exec):
361 #
362 EXEC_KMODS      +=

364 #
365 #       Scheduling Class Modules (/kernel/sched):
366 #
367 SCHED_KMODS     +=

369 #
370 #       File System Modules (/kernel/fs):
371 #
372 FS_KMODS        +=

374 #
375 #       Streams Modules (/kernel/strmod):
376 #
377 # STRMOD_KMODS  += kb

379 #
380 #       'System' Modules (/kernel/sys):
381 #
382 SYS_KMODS       +=

384 #
385 #       'User' Modules (/kernel/misc):
386 #
387 MISC_KMODS      += bootdev
```

```
388 MISC_KMODS      += dr_cpu
389 MISC_KMODS      += dr_io
390 MISC_KMODS      += dr_mem
391 MISC_KMODS      += ds
392 MISC_KMODS      += fault_iso
393 MISC_KMODS      += ldc
394 MISC_KMODS      += obpsym
395 MISC_KMODS      += platmod
396 MISC_KMODS      += platsvc
397 MISC_KMODS      += vis
398 MISC_KMODS      += pcie

400 #       md5 optimized for Niagara
401 #
402 MISC_KMODS      += md5

404 #
405 #       Brand modules
406 #
407 BRAND_KMODS     += sn1_brand s10_brand

409 #
410 #       Software Cryptographic Providers (/kernel/crypto):
411 #
412 CRYPTO_KMODS    += arcfour

414 #
415 #       generic-unix module (/kernel/genunix):
416 #
417 GENUNIX_KMODS   += genunix

419 #
420 #       Modules eXcluded from the product:
421 #
422 XMODS           +=

424 #
425 #       cpu modules
426 #
427 CPU_KMODS       += generic niagara niagara2 vfalls kt

429 LINT_CPU_KMODS  += generic

431 #
432 #       Performance Counter BackEnd Modules (/usr/kernel/pcbe):
433 #
434 PCBE_KMODS      += niagara_pcbe
435 PCBE_KMODS      += niagara2_pcbe
436 PCBE_KMODS      += vfalls_pcbe
437 PCBE_KMODS      += kt_pcbe
```

```
**********************************************************
    4081 Fri Apr 17 12:29:39 2015
new/usr/src/uts/sun4v/genunix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#
#       This makefile drives the production of the generic
#       unix kernel module.
#
#       sparc implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = genunix
GENUNIX         = $(OBJS_DIR)/$(MODULE)

OBJECTS         = $(GENUNIX_OBJS:%=$(OBJS_DIR)/%) \
                  $(NOT_YET_KMODS:%=$(OBJS_DIR)/%)

LINTS           = $(GENUNIX_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln)

ROOTMODULE      = $(ROOT_PSM_KERN_DIR)/$(MODULE)

PLATFORM        = sun4v
LIBGEN          = $(OBJS_DIR)/libgenunix.so
LIBSTUBS        = $(GENSTUBS_OBJS:%=$(OBJS_DIR)/%)

# LINTFLAGS will be set to include definitions so that the cpu_t
# structure is expanded. However this could be set to look at the
# sun4u version which is not correct for sun4v. Therefore we only
# want to use the LINTFLAGS modification in this Makefile and so
```



```
# suppress the usage of the LINTFLAGS setting in the Makefile.sparc
# file.
#
LINTFLAGSUPPRESS        = $(POUND_SIGN)

#
#       Include common rules.
#
include $(UTSBASE)/sparc/Makefile.sparc

#
#       Define targets
#
ALL_TARGET      = $(LIBGEN)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(GENUNIX) $(ROOTMODULE)

#
#       Override defaults
#
CLEANFILES      += $(LIBSTUBS) $(LIBGEN)

LINT_LIB_DIR    = $(UTSBASE)/$(PLATFORM)/lint-libs/$(OBJS_DIR)
LINT_LIB        = $(LINT_LIB_DIR)/llib-lunix.ln
GEN_LINT_LIB    =

BINARY          =

CLOBBERFILES    += $(GENUNIX)

#
# Non-patch genunix builds merge a version of the ip module called ipctf.  This
# is to ensure that the common network-related types are included in genunix and
# can thus be uniquified out of other modules.  We don't want to do this for
# patch builds, since we can't guarantee that ip and genunix will be in the same
# patch.
#
IPCTF_TARGET    = $(IPCTF)
$(PATCH_BUILD)IPCTF_TARGET =

#
# lint pass one enforcement
#
CFLAGS += $(CCVERBOSE)
CPPFLAGS += -I$(SRC)/common
CPPFLAGS += -I$(SRC)/uts/common/fs/zfs

INC_PATH +=  -I$(UTSBASE)/sun4

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-value
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-parentheses
```

```
116 CERRWARN        += -_gcc=-Wno-switch
117 CERRWARN        += -_gcc=-Wno-type-limits
118 CERRWARN        += -_gcc=-Wno-uninitialized
119 CERRWARN        += -_gcc=-Wno-clobbered
120 CERRWARN        += -_gcc=-Wno-empty-body


123 # Ensure that lint sees 'struct cpu' containing a fully declared
124 # embedded 'struct machcpu'.
125 #
126 LINTFLAGS       += -D_MACHDEP -I../../sun4 -I../../$(PLATFORM) -I../../sfmmu

128 #
129 #       Default build targets.
130 #
131 .KEEP_STATE:

133 .PARALLEL:      $(LIBSTUBS)

135 def:            $(DEF_DEPS)

137 all:            $(ALL_DEPS)

139 clean:          $(CLEAN_DEPS)

141 clobber:        $(CLOBBER_DEPS)

143 lint:           $(LINT_DEPS)

145 modlintlib:     $(MODLINTLIB_DEPS)

147 clean.lint:     $(CLEAN_LINT_DEPS)

149 install:        $(INSTALL_DEPS)

151 $(LIBGEN):      $(GENUNIX) $(LIBSTUBS)
152         $(BUILD.SO) $(GENUNIX) $(LIBSTUBS)

154 $(GENUNIX): $(OBJECTS)
164 $(GENUNIX): $(IPCTF_TARGET) $(OBJECTS)
155         @pwd
156         $(LD) -r $(LDFLAGS) -o $@ $(OBJECTS)
157         $(CTFMERGE_MODULE)
167         $(CTFMERGE_GENUNIX_MERGE)
158         $(POST_PROCESS)

160 $(OBJECTS): $(OBJS_DIR)

162 #
163 #       Include common targets.
164 #
165 include $(UTSBASE)/sparc/Makefile.targ

167 #
168 #       Include workarounds.
169 #
170 include $(UTSBASE)/$(PLATFORM)/Makefile.workarounds

172 ALL_DEFS +=     $(WORKAROUND_DEFS)
```

```
**********************************************************
    5675 Fri Apr 17 12:29:40 2015
new/usr/src/uts/sun4v/unix/Makefile
1961 investigate stopping uniquifying CTF information
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

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

#
#       This makefile drives the production of /unix (and unix.o).
#
#       sun4v implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
UNIX            = unix
OBJECTS         = $(SPECIAL_OBJS:%=$(OBJS_DIR)/%) \
                  $(CORE_OBJS:%=$(OBJS_DIR)/%) \
                  $(MACH_NOT_YET_KMODS:%=$(OBJS_DIR)/%)
LINTS           = $(SPECIAL_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(CORE_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(KRTLD_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(MACH_NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln) \
                  $(LINTS_DIR)/vers.ln \
                  $(LINTS_DIR)/modstubs.ln

KRTLD_MAPFILE   = $(UTSBASE)/sparc/krtld/mapfile
KRTLD_OBJECTS   = $(KRTLD_OBJS:%=$(OBJS_DIR)/%)
KRTLD_O         = $(OBJS_DIR)/krtld.o

ROOTMODULE      = $(ROOT_PSM_KERN_DIR)/$(UNIX)
UNIX32_LINK     = $(ROOT_PSM_KERN_DIR_32)/$(UNIX)
UNIX_BIN        = $(OBJS_DIR)/$(UNIX)
```



```
LIBS            = $(GENLIB) $(PLATLIB) $(CPULIB)

GENUNIX         = genunix
GENUNIX_DIR     = ../$(GENUNIX)
GENOPTS         = -L $(GENUNIX_DIR)/$(OBJS_DIR) -l $(GENUNIX)

CPU_DIR         = .
CPUOPTS         = -L $(CPU_DIR)/$(OBJS_DIR) -l $(CPUNAME)

PLAT_DIR        = ../platmod
PLATOPTS        = -L $(PLAT_DIR)/$(OBJS_DIR) -l $(PLATMOD)

LIBOPTS         = $(GENOPTS) $(PLATOPTS) $(CPUOPTS)

CTFEXTRAOBJS    = $(OBJS_DIR)/vers.o

#
#       Include common rules.
#
include $(UTSBASE)/sun4v/Makefile.sun4v

#
#       Define targets
#
ALL_TARGET      = $(UNIX_BIN)
LINT_TARGET     = $(LINT_LIB)
INSTALL_TARGET  = $(UNIX_BIN) $(ROOTMODULE) $(UNIX32_LINK)

#
#       This is UNIX_DIR. Use a short path.
#
UNIX_DIR        = .

#
#       Overrides
#
CLEANFILES      += $(UNIX_O) $(MODSTUBS_O) $(KRTLD_O) $(KRTLD_OBJECTS) \
                   $(OBJS_DIR)/vers.c $(OBJS_DIR)/vers.o \
                   $(CPU_OBJ) $(CPULIB) \
                   $(DTRACESTUBS_O) $(DTRACESTUBS)

CLOBBERFILES    = $(CLEANFILES) $(UNIX_BIN)
CLEANLINTFILES  += $(LINT_LIB)

#
# lint pass one enforcement
# Turn on doubleword alignment for 64 bit counter timer registers
#
CFLAGS += $(CCVERBOSE) -dalign

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-char-subscripts
CERRWARN        += -_gcc=-Wno-unused-variable
```

```
126 CERRWARN        += -_gcc=-Wno-unused-function
127 CERRWARN        += -_gcc=-Wno-unused-label
128 CERRWARN        += -_gcc=-Wno-type-limits
129 CERRWARN        += -_gcc=-Wno-clobbered
130 CERRWARN        += -_gcc=-Wno-empty-body
131 CERRWARN        += -_gcc=-Wno-unused-value
132 CERRWARN        += -_gcc=-Wno-switch

134 #
135 #       Default build targets.
136 #
137 .KEEP_STATE:

139 def:            $(DEF_DEPS)

141 all:            $(ALL_DEPS)

143 clean:          $(CLEAN_DEPS)

145 clobber:        $(CLOBBER_DEPS)

147 lint:           $(LINT_DEPS)

149 clean.lint:     $(CLEAN_LINT_DEPS)

151 install:        $(INSTALL_DEPS)


154 $(UNIX_BIN):    $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(MAPFILE) $(LIBS) \
155                 $(DTRACESTUBS)
156         $(LD) -dy -b -o $@ -e _start -M $(MAPFILE) \
157             $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)
158         $(CTFMERGE_MODULE)
158         $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
159         $(POST_PROCESS)

161 $(UNIX32_LINK): $(ROOT_PSM_KERN_DIR_32) $(UNIX_BIN)
162         -$(RM) $@; ln -s $(SUBDIR64)/$(UNIX) $@

164 symcheck:       $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBS)
165         $(LD) -dy -b -o $(SYM_MOD) -M $(MAPFILE) \
166         $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)

168 $(UNIX_O):      $(OBJECTS) $(OBJS_DIR)/vers.o
169         $(LD) -r -o $@ $(OBJECTS) $(OBJS_DIR)/vers.o

171 $(KRTLD_O):     $(KRTLD_OBJECTS)
172         $(LD) -r -o $@ -M$(KRTLD_MAPFILE) $(KRTLD_OBJECTS)

174 #
175 #       Special rules for generating assym.h for inclusion in assembly files.
176 #
177 $(DSF_DIR)/$(OBJS_DIR)/assym.h: FRC
178         @cd $(DSF_DIR); $(MAKE) all.targ

180 $(GENLIB):      FRC
181         @pwd
182         @(cd $(GENLIB_DIR); pwd; $(MAKE) all.targ)

184 $(PLATLIB):
185         ?@pwd
186         ?@(cd $(PLAT_DIR); pwd; $(MAKE) all.targ)

188 #
189 #       CPU_OBJ now comprises of 2 object files which come from sun4 common
190 #       and from architecture dependent code.  OBJS_DIR is prepended where
```



```
191 #       CPU_OBJ is defined to allow for building multiple CPU_OBJ's
192 #
193 $(CPULIB):      $(CPU_OBJ)
194         $(BUILD.SO) $(CPU_OBJ)

196 #
197 #       The global lint target builds the kernel lint library (llib-lunix.ln)
198 #       which is equivalent to a lint of /unix.o. Then all kernel modules for
199 #       this architecture are linted against the kernel lint library.
200 #
201 #       Note:   lint errors in the kernel lint library will be repeated for
202 #               each module. It is important that the kernel lint library
203 #               be clean to keep the textual output to a reasonable level.
204 #

206 $(LINT_LIB):    $(LINT_LIB_DIR) $(LINTS)
207         @-$(ECHO) "\n$(UNIX): (library construction):"
208         @$(LINT) -o $(UNIX) $(LINTFLAGS) $(LINTS)
209         @$(MV) $(@F) $@

211 lintlib:        $(LINT_DEPS)

213 #
214 #       Include common targets.
215 #
216 include $(UTSBASE)/$(PLATFORM)/Makefile.targ
```