```
**********************************************************
   66771 Mon Feb 11 00:23:18 2019
new/usr/src/cmd/sgs/include/libld.h
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
______unchanged_portion_omitted_

1168 /*
1169  * The auxiliary symbol descriptor contains the additional information (beyond
1170  * the symbol descriptor) required to process global symbols.  These symbols are
1171  * accessed via an internal symbol hash table where locality of reference is
1172  * important for performance.
1173  */
1174 struct sym_aux {
1175         APlist          *sa_dfiles;     /* files where symbol is defined */
1176         Sym             sa_sym;         /* copy of symtab entry */
1177         const char      *sa_vfile;      /* first unavailable definition */
1178         const char      *sa_rfile;      /* file with first symbol referenced */
1179         Word            sa_hash;        /* the pure hash value of symbol */
1180         Word            sa_PLTndx;      /* index into PLT for symbol */
1181         Word            sa_PLTGOTndx;   /* GOT entry indx for PLT indirection */
1182         Word            sa_linkndx;     /* index of associated symbol from */
1183                                         /*      ET_DYN file */
1184         Half            sa_symspec;     /* special symbol ids */
1185         Half            sa_overndx;     /* output file versioning index */
1186         Half            sa_dverndx;     /* dependency versioning index */
1187         Os_desc         *sa_boundsec;   /* output section of SECBOUND_ syms */
1188 #endif /* ! codereview */
1189 };

1191 /*
1192  * Nodes used to track symbols in the global AVL symbol dictionary.
1193  */
1194 struct sym_avlnode {
1195         avl_node_t      sav_node;       /* AVL node */
1196         Word            sav_hash;       /* symbol hash value */
1197         const char      *sav_name;      /* symbol name */
1198         Sym_desc        *sav_sdp;       /* symbol descriptor */
1199 };

1201 /*
1202  * These are the ids for processing of 'Special symbols'.  They are used
1203  * to set the sym->sd_aux->sa_symspec field.
1204  */
1205 #define SDAUX_ID_ETEXT          1       /* etext && _etext symbol */
1206 #define SDAUX_ID_EDATA          2       /* edata && _edata symbol */
1207 #define SDAUX_ID_END            3       /* end, _end, && _END_ symbol */
1208 #define SDAUX_ID_DYN            4       /* DYNAMIC && _DYNAMIC symbol */
1209 #define SDAUX_ID_PLT            5       /* _PROCEDURE_LINKAGE_TABLE_ symbol */
1210 #define SDAUX_ID_GOT            6       /* _GLOBAL_OFFSET_TABLE_ symbol */
1211 #define SDAUX_ID_START          7       /* START_ && _START_ symbol */
1212 #define SDAUX_ID_SECBOUND_START 8       /* __start_<section> symbols */
1213 #define SDAUX_ID_SECBOUND_STOP  9       /* __stop_<section> symbols */
1214 #endif /* ! codereview */

1216 /*
1217  * Flags for sym_desc.sd_flags
1218  */
1219 #define FLG_SY_MVTOCOMM 0x00000001      /* assign symbol to common (.bss) */
1220                                         /*      this is a result of a */
1221                                         /*      copy reloc against sym */
1222 #define FLG_SY_GLOBREF  0x00000002      /* a global reference has been seen */
1223 #define FLG_SY_WEAKDEF  0x00000004      /* a weak definition has been used */
1224 #define FLG_SY_CLEAN    0x00000008      /* 'Sym' entry points to original */
```



```
1225                                         /*      input file (read-only). */
1226 #define FLG_SY_UPREQD   0x00000010      /* symbol value update is required, */
1227                                         /*      either it's used as an entry */
1228                                         /*      point or for relocation, but */
1229                                         /*      it must be updated even if */
1230                                         /*      the -s flag is in effect */
1231 #define FLG_SY_NOTAVAIL 0x00000020      /* symbol is not available to the */
1232                                         /*      application either because it */
1233                                         /*      originates from an implicitly */
1234                                         /*      referenced shared object, or */
1235                                         /*      because it is not part of a */
1236                                         /*      specified version. */
1237 #define FLG_SY_REDUCED  0x00000040      /* a global is reduced to local */
1238 #define FLG_SY_VERSPROM 0x00000080      /* version definition has been */
1239                                         /*      promoted to output file */
1240 #define FLG_SY_PROT     0x00000100      /* stv_protected visibility seen */
1241 #define FLG_SY_MAPREF   0x00000200      /* symbol reference generated by user */
1242                                         /*      from mapfile */
1243 #define FLG_SY_REFRSD   0x00000400      /* symbols sd_ref has been raised */
1244                                         /*      due to a copy-relocs */
1245                                         /*      weak-strong pairing */
1246 #define FLG_SY_INTPOSE  0x00000800      /* symbol defines an interposer */
1247 #define FLG_SY_INVALID  0x00001000      /* unwanted/erroneous symbol */
1248 #define FLG_SY_SMGOT    0x00002000      /* small got index assigned to symbol */
1249                                         /*      sparc only */
1250 #define FLG_SY_PARENT   0x00004000      /* symbol to be found in parent */
1251                                         /*    only used with direct bindings */
1252 #define FLG_SY_LAZYLD   0x00008000      /* symbol to cause lazyloading of */
1253                                         /*      parent object */
1254 #define FLG_SY_ISDISC   0x00010000      /* symbol is a member of a DISCARDED */
1255                                         /*      section (COMDAT) */
1256 #define FLG_SY_PAREXPN  0x00020000      /* partially init. symbol to be */
1257                                         /*      expanded */
1258 #define FLG_SY_PLTPAD   0x00040000      /* pltpadding has been allocated for */
1259                                         /*      this symbol */
1260 #define FLG_SY_REGSYM   0x00080000      /* REGISTER symbol (sparc only) */
1261 #define FLG_SY_SOFOUND  0x00100000      /* compared against an SO definition */
1262 #define FLG_SY_EXTERN   0x00200000      /* symbol is external, allows -zdefs */
1263                                         /*    error suppression */
1264 #define FLG_SY_MAPUSED  0x00400000      /* mapfile symbol used (occurred */
1265                                         /*    within a relocatable object) */
1266 #define FLG_SY_COMMEXP  0x00800000      /* COMMON symbol which has been */
1267                                         /*      allocated */
1268 #define FLG_SY_CMDREF   0x01000000      /* symbol was referenced from the */
1269                                         /*      command line.  (ld -u <>, */
1270                                         /*      ld -zrtldinfo=<>, ...) */
1271 #define FLG_SY_SPECSEC  0x02000000      /* section index is reserved value */
1272                                         /*      ABS, COMMON, ... */
1273 #define FLG_SY_TENTSYM  0x04000000      /* tentative symbol */
1274 #define FLG_SY_VISIBLE  0x08000000      /* symbols visibility determined */
1275 #define FLG_SY_STDFLTR  0x10000000      /* symbol is a standard filter */
1276 #define FLG_SY_AUXFLTR  0x20000000      /* symbol is an auxiliary filter */
1277 #define FLG_SY_DYNSORT  0x40000000      /* req. in dyn[sym|tls]sort section */
1278 #define FLG_SY_NODYNSORT 0x80000000     /* excluded from dyn[sym_tls]sort sec */

1280 #define FLG_SY_DEFAULT  0x0000100000000 /* global symbol, default */
1281 #define FLG_SY_SINGLE   0x0000200000000 /* global symbol, singleton defined */
1282 #define FLG_SY_PROTECT  0x0000400000000 /* global symbol, protected defined */
1283 #define FLG_SY_EXPORT   0x0000800000000 /* global symbol, exported defined */

1285 #define MSK_SY_GLOBAL \
1286         (FLG_SY_DEFAULT | FLG_SY_SINGLE | FLG_SY_PROTECT | FLG_SY_EXPORT)
1287                                         /* this mask indicates that the */
1288                                         /*    symbol has been explicitly */
1289                                         /*    defined within a mapfile */
1290                                         /*    definition, and is a candidate */
```

```
1291                                    /*    for versioning */

1293 #define FLG_SY_HIDDEN   0x0001000000000 /* global symbol, reduce to local */
1294 #define FLG_SY_ELIM     0x0002000000000 /* global symbol, eliminate */
1295 #define FLG_SY_IGNORE   0x0004000000000 /* global symbol, ignored */

1297 #define MSK_SY_LOCAL    (FLG_SY_HIDDEN | FLG_SY_ELIM | FLG_SY_IGNORE)
1298                                    /* this mask allows all local state */
1299                                    /*    flags to be removed when the */
1300                                    /*    symbol is copy relocated */

1302 #define FLG_SY_EXPDEF   0x0008000000000 /* symbol visibility defined */
1303                                    /*    explicitly */

1305 #define MSK_SY_NOAUTO   (FLG_SY_SINGLE | FLG_SY_EXPORT | FLG_SY_EXPDEF)
1306                                    /* this mask indicates that the */
1307                                    /*    symbol is not a candidate for */
1308                                    /*    auto-reduction/elimination */

1310 #define FLG_SY_MAPFILE  0x0010000000000 /* symbol attribute defined in a */
1311                                    /*    mapfile */
1312 #define FLG_SY_DIR      0x0020000000000 /* global symbol, direct bindings */
1313 #define FLG_SY_NDIR     0x0040000000000 /* global symbol, nondirect bindings */
1314 #define FLG_SY_OVERLAP  0x0080000000000 /* move entry overlap detected */
1315 #define FLG_SY_CAP      0x0100000000000 /* symbol is associated with */
1316                                    /*    capabilities */
1317 #define FLG_SY_DEFERRED 0x0200000000000 /* symbol should not be bound to */
1318                                    /*      during BIND_NOW relocations */

1320 /*
1321  * A symbol can only be truly hidden if it is not a capabilities symbol.
1322  */
1323 #define SYM_IS_HIDDEN(_sdp) \
1324         (((_sdp)->sd_flags & (FLG_SY_HIDDEN | FLG_SY_CAP)) == FLG_SY_HIDDEN)

1326 /*
1327  * Create a mask for (sym.st_other & visibility) since the gABI does not yet
1328  * define a ELF*_ST_OTHER macro.
1329  */
1330 #define MSK_SYM_VISIBILITY      0x7

1332 /*
1333  * Structure to manage the shared object definition lists.  There are two lists
1334  * that use this structure:
1335  *
1336  *  -   ofl_soneed; maintain the list of implicitly required dependencies
1337  *      (ie. shared objects needed by other shared objects).  These definitions
1338  *      may include RPATH's required to locate the dependencies, and any
1339  *      version requirements.
1340  *
1341  *  -   ofl_socntl; maintains the shared object control definitions.  These are
1342  *      provided by the user (via a mapfile) and are used to indicate any
1343  *      version control requirements.
1344  */
1345 struct  sdf_desc {
1346         const char      *sdf_name;      /* the shared objects file name */
1347         char            *sdf_rpath;     /* library search path DT_RPATH */
1348         const char      *sdf_rfile;     /* referencing file for diagnostics */
1349         Ifl_desc        *sdf_file;      /* the final input file descriptor */
1350         Alist           *sdf_vers;      /* list of versions that are required */
1351                                         /*      from this object */
1352         Alist           *sdf_verneed;   /* list of VERNEEDS to create for */
1353                                         /*      object via mapfile ADDVERS */
1354         Word            sdf_flags;
1355 };
```



```
1357 #define FLG_SDF_SELECT  0x01            /* version control selection required */
1358 #define FLG_SDF_VERIFY  0x02            /* version definition verification */
1359                                         /*      required */
1360 #define FLG_SDF_ADDVER  0x04            /* add VERNEED references */

1362 /*
1363  * Structure to manage shared object version usage requirements.
1364  */
1365 struct  sdv_desc {
1366         const char      *sdv_name;      /* version name */
1367         const char      *sdv_ref;       /* versions reference */
1368         Word            sdv_flags;      /* flags */
1369 };

1371 #define FLG_SDV_MATCHED 0x01            /* VERDEF found and matched */

1373 /*
1374  * Structures to manage versioning information.  Two versioning structures are
1375  * defined:
1376  *
1377  *   -  a version descriptor maintains a linked list of versions and their
1378  *      associated dependencies.  This is used to build the version definitions
1379  *      for an image being created (see map_symbol), and to determine the
1380  *      version dependency graph for any input files that are versioned.
1381  *
1382  *   -  a version index array contains each version of an input file that is
1383  *      being processed.  It informs us which versions are available for
1384  *      binding, and is used to generate any version dependency information.
1385  */
1386 struct  ver_desc {
1387         const char      *vd_name;       /* version name */
1388         Ifl_desc        *vd_file;       /* file that defined version */
1389         Word            vd_hash;        /* hash value of name */
1390         Half            vd_ndx;         /* coordinates with symbol index */
1391         Half            vd_flags;       /* version information */
1392         APlist          *vd_deps;       /* version dependencies */
1393         Ver_desc        *vd_ref;        /* dependency's first reference */
1394 };

1396 struct  ver_index {
1397         const char      *vi_name;       /* dependency version name */
1398         Half            vi_flags;       /* communicates availability */
1399         Half            vi_overndx;     /* index assigned to this version in */
1400                                         /*      output object Verneed section */
1401         Ver_desc        *vi_desc;       /* cross reference to descriptor */
1402 };

1404 /*
1405  * Define any internal version descriptor flags ([vd|vi]_flags).  Note that the
1406  * first byte is reserved for user visible flags (refer VER_FLG's in link.h).
1407  */
1408 #define MSK_VER_USER    0x0f            /* mask for user visible flags */

1410 #define FLG_VER_AVAIL   0x10            /* version is available for binding */
1411 #define FLG_VER_REFER   0x20            /* version has been referenced */
1412 #define FLG_VER_CYCLIC  0x40            /* a member of cyclic dependency */

1414 /*
1415  * isalist(1) descriptor - used to break an isalist string into its component
1416  * options.
1417  */
1418 struct  isa_opt {
1419         char            *isa_name;      /* individual isa option name */
1420         size_t          isa_namesz;     /*      and associated size */
1421 };
```

```
1423 struct  isa_desc {
1424         char            *isa_list;      /* sysinfo(SI_ISALIST) list */
1425         size_t          isa_listsz;     /*      and associated size */
1426         Isa_opt         *isa_opt;       /* table of individual isa options */
1427         size_t          isa_optno;      /*      and associated number */
1428 };

1430 /*
1431  * uname(2) descriptor - used to break a utsname structure into its component
1432  * options (at least those that we're interested in).
1433  */
1434 struct  uts_desc {
1435         char            *uts_osname;    /* operating system name */
1436         size_t          uts_osnamesz;   /*      and associated size */
1437         char            *uts_osrel;     /* operating system release */
1438         size_t          uts_osrelsz;    /*      and associated size */
1439 };

1441 /*
1442  * SHT_GROUP descriptor - used to track group sections at the global
1443  * level to resolve conflicts and determine which to keep.
1444  */
1445 struct group_desc {
1446         Is_desc         *gd_isc;        /* input section descriptor */
1447         Is_desc         *gd_oisc;       /* overriding input section */
1448                                         /*      descriptor when discarded */
1449         const char      *gd_name;       /* group name (signature symbol) */
1450         Word            *gd_data;       /* data for group section */
1451         size_t          gd_cnt;         /* number of entries in group data */
1452 };

1454 /*
1455  * Indexes into the ld_support_funcs[] table.
1456  */
1457 typedef enum {
1458         LDS_VERSION = 0,        /* Must be first and have value 0 */
1459         LDS_INPUT_DONE,
1460         LDS_START,
1461         LDS_ATEXIT,
1462         LDS_OPEN,
1463         LDS_FILE,
1464         LDS_INSEC,
1465         LDS_SEC,
1466         LDS_NUM
1467 } Support_ndx;

1469 /*
1470  * Structure to manage archive member caching.  Each archive has an archive
1471  * descriptor (Ar_desc) associated with it.  This contains pointers to the
1472  * archive symbol table (obtained by elf_getarsyms(3e)) and an auxiliary
1473  * structure (Ar_uax[]) that parallels this symbol table.  The member element
1474  * of this auxiliary table indicates whether the archive member associated with
1475  * the symbol offset has already been extracted (AREXTRACTED) or partially
1476  * processed (refer process_member()).
1477  */
1478 typedef struct ar_mem {
1479         Elf             *am_elf;        /* elf descriptor for this member */
1480         const char      *am_name;       /* members name */
1481         const char      *am_path;       /* path (ie. lib(foo.o)) */
1482         Sym             *am_syms;       /* start of global symbols */
1483         char            *am_strs;       /* associated string table start */
1484         Xword           am_symn;        /* no. of global symbols */
1485 } Ar_mem;

1487 typedef struct ar_aux {
1488         Sym_desc        *au_syms;       /* internal symbol descriptor */
```



```
1489         Ar_mem          *au_mem;        /* associated member */
1490 } Ar_aux;

1492 #define FLG_ARMEM_PROC  (Ar_mem *)-1

1494 typedef struct ar_desc {
1495         const char      *ad_name;       /* archive file name */
1496         Elf             *ad_elf;        /* elf descriptor for the archive */
1497         Elf_Arsym       *ad_start;      /* archive symbol table start */
1498         Ar_aux          *ad_aux;        /* auxiliary symbol information */
1499         dev_t           ad_stdev;       /* device id and inode number for */
1500         ino_t           ad_stino;       /*      multiple inclusion checks */
1501         ofl_flag_t      ad_flags;       /* archive specific cmd line flags */
1502 } Ar_desc;

1504 /*
1505  * Define any archive descriptor flags.  NOTE, make sure they do not clash with
1506  * any output file descriptor archive extraction flags, as these are saved in
1507  * the same entry (see MSK_OF1_ARCHIVE).
1508  */
1509 #define FLG_ARD_EXTRACT 0x00010000      /* archive member has been extracted */

1511 /* Mapfile versions supported by libld */
1512 #define MFV_NONE        0       /* Not a valid version */
1513 #define MFV_SYSV        1       /* Original System V syntax */
1514 #define MFV_SOLARIS     2       /* Solaris mapfile syntax */
1515 #define MFV_NUM         3       /* # of mapfile versions */


1518 /*
1519  * Function Declarations.
1520  */
1521 #if     defined(_ELF64)

1523 #define ld_create_outfile       ld64_create_outfile
1524 #define ld_ent_setup            ld64_ent_setup
1525 #define ld_init_strings         ld64_init_strings
1526 #define ld_init_target          ld64_init_target
1527 #define ld_make_sections        ld64_make_sections
1528 #define ld_main                 ld64_main
1529 #define ld_ofl_cleanup          ld64_ofl_cleanup
1530 #define ld_process_mem          ld64_process_mem
1531 #define ld_reloc_init           ld64_reloc_init
1532 #define ld_reloc_process        ld64_reloc_process
1533 #define ld_sym_validate         ld64_sym_validate
1534 #define ld_update_outfile       ld64_update_outfile

1536 #else

1538 #define ld_create_outfile       ld32_create_outfile
1539 #define ld_ent_setup            ld32_ent_setup
1540 #define ld_init_strings         ld32_init_strings
1541 #define ld_init_target          ld32_init_target
1542 #define ld_make_sections        ld32_make_sections
1543 #define ld_main                 ld32_main
1544 #define ld_ofl_cleanup          ld32_ofl_cleanup
1545 #define ld_process_mem          ld32_process_mem
1546 #define ld_reloc_init           ld32_reloc_init
1547 #define ld_reloc_process        ld32_reloc_process
1548 #define ld_sym_validate         ld32_sym_validate
1549 #define ld_update_outfile       ld32_update_outfile

1551 #endif

1553 extern int              ld_getopt(Lm_list *, int, int, char **);
```

```
1555 extern int              ld32_main(int, char **, Half);
1556 extern int              ld64_main(int, char **, Half);

1558 extern uintptr_t        ld_create_outfile(Ofl_desc *);
1559 extern uintptr_t        ld_ent_setup(Ofl_desc *, Xword);
1560 extern uintptr_t        ld_init_strings(Ofl_desc *);
1561 extern int              ld_init_target(Lm_list *, Half mach);
1562 extern uintptr_t        ld_make_sections(Ofl_desc *);
1563 extern void             ld_ofl_cleanup(Ofl_desc *);
1564 extern Ifl_desc         *ld_process_mem(const char *, const char *, char *,
1565                             size_t, Ofl_desc *, Rej_desc *);
1566 extern uintptr_t        ld_reloc_init(Ofl_desc *);
1567 extern uintptr_t        ld_reloc_process(Ofl_desc *);
1568 extern uintptr_t        ld_sym_validate(Ofl_desc *);
1569 extern uintptr_t        ld_update_outfile(Ofl_desc *);

1571 #ifdef  __cplusplus
1572 }
1573 #endif

1575 #endif  /* _LIBLD_H */
```

```
**********************************************************
   60496 Mon Feb 11 00:23:19 2019
new/usr/src/cmd/sgs/libld/common/libld.msg
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

```
#
# 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) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
#

#
# Copyright (c) 2012, Joyent, Inc.  All rights reserved.
# Copyright 2017 RackTop Systems.
#

@ _START_

# Message file for cmd/sgs/libld.

@ MSG_ID_LIBLD

#
# TRANSLATION_NOTE -- Beginning of USAGE message
#       The following messages are the usage messages for the ld command.
#       Tab characters (\t) are used to align the messages.
#
#       Each usage message starts with \t, and if the message has more than one
#               line, the following messages are aligned by 3 tab characters.
#               When you see \n\t\t\t, the first \n is used to change the line,
#               and following 3 tab characters are used to align the line.
#
#       Each usage message option is surrounded by [ and ].  Then the
#       description of the option follows. The descriptions should be aligned,
#       so tab characters are padded as needed after the closing bracket ].
#
#       How to align the messages are up to the translators and the
#       localization engineers.
#
#       In C locale, the first 3 messages would look like the following:
#
#       usage: ld [-6:abc:.....] file(s)
#               [-a]            create an absolute file
#               [-b]            do not do special PIC relocations in a.out
#               [-c file]       record configuration 'file'
```



```
#
@ MSG_ARG_USAGE         "usage: ld [-%s] file(s)\n"
@ MSG_ARG_DETAIL_3      "\t[-32]\t\tenforce a 32-bit link-edit\n"
@ MSG_ARG_DETAIL_6      "\t[-64]\t\tenforce a 64-bit link-edit\n"
@ MSG_ARG_DETAIL_A      "\t[-a]\t\tcreate an absolute file\n"
@ MSG_ARG_DETAIL_B      "\t[-b]\t\tdo not do special PIC relocations in a.out\n"
@ MSG_ARG_DETAIL_CBDR   "\t[-B direct | nodirect]\n\
                         \t\t\testablish direct bindings, or inhibit direct \
                         binding\n\
                         \t\t\tto, the object being created\n"
@ MSG_ARG_DETAIL_CBDY   "\t[-B dynamic | static]\n\
                         \t\t\tsearch for shared libraries|archives\n"
@ MSG_ARG_DETAIL_CBE    "\t[-B eliminate]\teliminate unqualified global \
                         symbols from the\n\t\t\tsymbol table\n"
@ MSG_ARG_DETAIL_CBG    "\t[-B group]\trelocate object from within group\n"
@ MSG_ARG_DETAIL_CBL    "\t[-B local]\treduce unqualified global symbols to \
                         local\n"
@ MSG_ARG_DETAIL_CBR    "\t[-B reduce]\tprocess symbol reductions\n"
@ MSG_ARG_DETAIL_CBS    "\t[-B symbolic]\tbind external references to \
                         definitions when creating\n\
                         \t\t\tshared objects\n"
@ MSG_ARG_DETAIL_C      "\t[-c name]\trecord configuration file 'name'\n"
@ MSG_ARG_DETAIL_CC     "\t[-C]\t\tdemangle C++ symbol name diagnostics\n"
@ MSG_ARG_DETAIL_D      "\t[-d y | n]\toperate in dynamic|static mode\n"
@ MSG_ARG_DETAIL_CD     "\t[-D token,...]\tprint diagnostic messages\n"
@ MSG_ARG_DETAIL_E      "\t[-e epsym], [--entry epsym]\n\
                         \t\t\tuse 'epsym' as entry point address\n"
@ MSG_ARG_DETAIL_F      "\t[-f name], [--auxiliary name]\n\
                         \t\t\tspecify library for which this file is an \
                         auxiliary\n\t\t\tfilter\n"
@ MSG_ARG_DETAIL_CF     "\t[-F name], [--filter name]\n\
                         \t\t\tspecify library for which this file is a filter\n
@ MSG_ARG_DETAIL_CG     "\t[-G], [-shared]\n\
                         \t\t\tcreate a shared object\n"
@ MSG_ARG_DETAIL_H      "\t[-h name], [--soname name]\n\
                         \t\t\tuse 'name' as internal shared object identifier\n
@ MSG_ARG_DETAIL_I      "\t[-i]\t\tignore LD_LIBRARY_PATH setting\n"
@ MSG_ARG_DETAIL_CI     "\t[-I name]\tuse 'name' as path of interpreter\n"
@ MSG_ARG_DETAIL_L      "\t[-l x], [--library x]\n\
                         \t\t\tsearch for libx.so or libx.a\n"
@ MSG_ARG_DETAIL_CL     "\t[-L path], [--library-path path]\n\
                         \t\t\tsearch for libraries in directory 'path'\n"
@ MSG_ARG_DETAIL_M      "\t[-m]\t\tprint memory map\n"
@ MSG_ARG_DETAIL_CM     "\t[-M mapfile]\tuse processing directives contained \
                         in 'mapfile'\n"
@ MSG_ARG_DETAIL_CN     "\t[-N string]\tcreate a dynamic dependency for \
                         'string'\n"
@ MSG_ARG_DETAIL_O      "\t[-o outfile], [--output outfile]\n\
                         \t\t\tname the output file 'outfile'\n"
@ MSG_ARG_DETAIL_P      "\t[-p auditlib]\tidentify audit library to accompany \
                         this object\n"
@ MSG_ARG_DETAIL_CP     "\t[-P auditlib]\tidentify audit library for \
                         processing the dependencies\n\
                         \t\t\tof this object\n"
@ MSG_ARG_DETAIL_CQ     "\t[-Q y | n]\tdo|do not place version information in \
                         output file\n"
@ MSG_ARG_DETAIL_R      "\t[-r], [--relocatable]\n\
                         \t\t\tcreate a relocatable object\n"
@ MSG_ARG_DETAIL_CR     "\t[-R path], [-rpath path]\n\
                         \t\t\tspecify a library search path to be used at run \
                         time\n"
@ MSG_ARG_DETAIL_S      "\t[-s], [--strip-all]\n\
                         \t\t\tstrip any symbol and debugging information\n"
@ MSG_ARG_DETAIL_CS     "\t[-S supportlib]\n\
                         \t\t\tspecify a link-edit support library\n"
@ MSG_ARG_DETAIL_T      "\t[-t]\t\tdo not warn of multiply-defined symbols \
```

```
126                         that have\n\t\t\tdifferent sizes or alignments\n"
127 @ MSG_ARG_DETAIL_U      "\t[-u symname], [--undefined symname]\n\
128                         \t\t\tcreate an undefined symbol 'symname'\n"
129 @ MSG_ARG_DETAIL_CV     "\t[-V], [--version]\n\
130                         \t\t\tprint version information\n"
131 @ MSG_ARG_DETAIL_CY     "\t[-Y P,dirlist]\tuse 'dirlist' as a default path \
132                         when searching for\n\
133                         \t\t\tlibraries\n"
134 @ MSG_ARG_DETAIL_ZA     "\t[-z absexec]\twhen building an executable absolute \
135                         symbols\n \
136                         \t\t\treferenced in dynamic objects are promoted to\n \
137                         \t\t\tthe executable\n"
138 @ MSG_ARG_DETAIL_ZAE    "\t[-z allextract | defaultextract | weakextract],\n\
139                         \t[--whole-archive | --no-whole-archive]\n\
140                         \t\t\textract all member files, only members that \
141                         resolve\n\
142                         \t\t\tundefined or tentative symbols, or \
143                         allow extraction of\n\
144                         \t\t\tarchive members to resolve weak references from \
145                         \n\t\t\t\tarchive files\n"
146 @ MSG_ARG_DETAIL_ZAL    "\t[-z altexec64]\texecute the 64-bit link-editor\n"
147 @ MSG_ARG_DETAIL_ZADLIB "\t[-z assert-deflib]\n\
148                        \t\t\tenables warnings for linking with libraries in \
149                        the \n\t\t\tdefault search path\n\
150                        \t[-z assert-deflib=libname]\n\
151                        \t\t\tenables warnings for linking with libraries in \
152                        the \n\t\t\tdefault search path, but 'libname' is exempt
153 @ MSG_ARG_DETAIL_ZC     "\t[-z combreloc | nocombreloc]\n\
154                         \t\t\tcombine|do not combine multiple relocation \
155                         sections\n"
156 @ MSG_ARG_DETAIL_ZNC    "\t[-z nocompstrtab]\n\t\t\tdisable compression of \
157                         string tables\n"
158 @ MSG_ARG_DETAIL_ZDEF   "\t[-z deferred | nodeferred]\n\
159                         \t\t\tenable|disable deferred identification of \
160                         shared object\n\t\t\tdependencies\n"
161 @ MSG_ARG_DETAIL_ZDFS   "\t[-z defs], [--no-undefined]\n\
162                         \t\t\tdisallow undefined symbol references\n"
163 @ MSG_ARG_DETAIL_ZDRS   "\t[-z direct | nodirect]\n\
164                         \t\t\tenable|disable direct binding to shared object\n\
165                         \t\t\tdependencies\n"
166 @ MSG_ARG_DETAIL_ZE     "\t[-z endfiltee]\tmarks a filtee such that it will \
167                         terminate a filters\n\t\t\tsearch\n"
168 @ MSG_ARG_DETAIL_ZFATW  "\t[-z fatal-warnings | nofatal-warnings],\n\
169                         \t[--fatal-warnings | --no-fatal-warnings]\n\
170                         \t\tenable|disable treatment of warnings as fatal\n"
171 @ MSG_ARG_DETAIL_ZFA    "\t[-z finiarray=function]\n\
172                         \t\t\tname of function to be appended to the \
173                         .fini_array\n"
174 @ MSG_ARG_DETAIL_ZGP    "\t[-z groupperm | nogroupperm]\n\
175                         \t\t\tenable|disable setting of group permissions\n\
176                         \t\t\ton dynamic dependencies\n"
177 @ MSG_ARG_DETAIL_ZGUIDE "\t[-z guidance | -z guidance=item1,item2,...]\n\
178                         \t\t\tenable guidance warnings. items: \
179                         noall, nodefs,\n\
180                         \t\t\tnodirect, nolazyload, nomapfile, notext, \
181                         nounused\n"
182 @ MSG_ARG_DETAIL_ZH     "\t[-z help], [--help]\n\
183                         \t\t\tprint this usage message\n"
184 @ MSG_ARG_DETAIL_ZIG    "\t[-z ignore | record]\n\
185                         \t\t\tignore|record unused dynamic dependencies\n"
186 @ MSG_ARG_DETAIL_ZINA   "\t[-z initarray=function]\n\
187                         \t\t\tname of function to be appended to the \
188                         .init_array\n"
189 @ MSG_ARG_DETAIL_ZINI   "\t[-z initfirst]\tmark object to indicate that its \
190                         .init section should\n\
191                         \t\t\tbe executed before the .init section of any \
```



```
192                         other\n\t\t\tobjects\n"
193 @ MSG_ARG_DETAIL_ZINT   "\t[-z interpose]\
194                         \tdynamic object is to be an 'interposer' on direct\n\
195                         \t\t\tbindings\n"
196 @ MSG_ARG_DETAIL_ZLAZY  "\t[-z lazyload | nolazyload]\n\
197                         \t\t\tenable|disable delayed loading of shared \
198                         object\n\t\t\tdependencies\n"
199 @ MSG_ARG_DETAIL_ZLD32  "\t[-z ld32=arg1,arg2,...]\n\
200                         \t\t\tdefine arguments applicable to the \
201                         32-bit class of ld(1)\n"
202 @ MSG_ARG_DETAIL_ZLD64  "\t[-z ld64=arg1,arg2,...]\n\
203                         \t\t\tdefine arguments applicable to the \
204                         64-bit class of ld(1)\n"
205 @ MSG_ARG_DETAIL_ZLO    "\t[-z loadfltr]\tmark filter as requiring immediate \
206                         loading of its\n\
207                         \t\t\tfiltees at runtime\n"
208 @ MSG_ARG_DETAIL_ZM     "\t[-z muldefs], [--allow-multiple-definition]\n\
209                         \t\t\tallow multiply-defined symbols\n"
210 @ MSG_ARG_DETAIL_ZNDFS  "\t[-z nodefs]\tallow undefined symbol references\n"
211 @ MSG_ARG_DETAIL_ZNDEF  "\t[-z nodefaultlib]\n\
212                         \t\t\tmark object to ignore any default library \
213                         search path\n"
214 @ MSG_ARG_DETAIL_ZNDEL  "\t[-z nodelete]\tmark object as non-deletable\n"
215 @ MSG_ARG_DETAIL_ZNDLO  "\t[-z nodlopen]\tmark object as non-dlopen()'able\n"
216 @ MSG_ARG_DETAIL_ZNDU   "\t[-z nodump]\tmark object as non-dldump()'able\n"
217 @ MSG_ARG_DETAIL_ZNLD   "\t[-z noldynsym]\tdo not add a .SUNW_ldynsym section\n"
218 @ MSG_ARG_DETAIL_ZNPA   "\t[-z nopartial]\texpand any partially initialized \
219                         symbols\n"
220 @ MSG_ARG_DETAIL_ZNV    "\t[-z noversion]\tdo not record any version sections\n"
221 @ MSG_ARG_DETAIL_ZNOW   "\t[-z now]\tmark object as requiring non-lazy \
222                         binding\n"
223 @ MSG_ARG_DETAIL_ZO     "\t[-z origin]\tmark object as requiring $ORIGIN \
224                         processing\n"
225 @ MSG_ARG_DETAIL_ZPIA   "\t[-z preinitarray=function]\n\
226                         \t\t\tname of function to be appended to the \
227                         .preinit_array\n"
228 @ MSG_ARG_DETAIL_ZRL    "\t[-z redlocsym]\treduce local syms in .symtab to \
229                         a minimum\n"
230 @ MSG_ARG_DETAIL_ZRREL  "\t[-z relaxreloc]\trelax rules used for relocations \
231                         against COMDAT sections\n"
232 @ MSG_ARG_DETAIL_ZRS    "\t[-z rescan]\tafter processing all arguments, rescan \
233                         archive list\n\
234                         \t\t\tuntil no further member extraction occurs\n"
235 @ MSG_ARG_DETAIL_ZRSN   "\t[-z rescan-now]\timmediately rescan archive list \
236                         until\n\
237                         \t\t\tno further member extraction occurs\n"
238 @ MSG_ARG_DETAIL_ZRSGRP "\t[-z rescan-start archives... -z rescan-end],\n\
239                         \t[--start-group archives... --end-group], \
240                         [-( archives... -)]\n\
241                         \t\t\trescan specified archive group upon reaching\n\
242                         \t\t\tthe end of the group, until no further\n\
243                         \t\t\tmember extraction occurs\n"
244 @ MSG_ARG_DETAIL_ZSCAP  "\t[-z symbolcap]\tconvert object capabilities to \
245                         symbol capabilities\n"
246 @ MSG_ARG_DETAIL_ZTARG  "\t[-z target=platform]\n\
247                         \t\t\ttarget machine for cross linking\n"
248 @ MSG_ARG_DETAIL_ZT     "\t[-z text]\tdisallow output relocations against \
249                         text\n"
250 @ MSG_ARG_DETAIL_ZTO    "\t[-z textoff]\tallow output relocations against \
251                         text\n"
252 @ MSG_ARG_DETAIL_ZTW    "\t[-z textwarn]\twarn if there are relocations \
253                         against text\n"
254 @ MSG_ARG_DETAIL_ZWRAP  "\t[-z wrap=symbol], [-wrap=symbol], [--wrap=symbol]\n\
255                         \t\t\twrap symbol references\n"
256 @ MSG_ARG_DETAIL_ZVER   "\t[-z verbose]\t\
257                         generate warnings for suspicious processings\n"
```

```
259 #
260 # TRANSLATION_NOTE -- End of USAGE message
261 #
262 @ MSG_GRP_INVALNDX      "file %s: group section [%u]%s: entry %d: \
263                          invalid section index: %d"

265 # Relocation processing messages (some of these are required to satisfy
266 # do_reloc(), which is common code used by cmd/sgs/rtld - make sure both
267 # message files remain consistent).

269 @ MSG_REL_NOFIT         "relocation error: %s: file %s: symbol %s: \
270                          value 0x%llx does not fit"
271 @ MSG_REL_NONALIGN      "relocation error: %s: file %s: symbol %s: \
272                          offset 0x%llx is non-aligned"
273 @ MSG_REL_NULL          "relocation error: file %s: section [%u]%s: \
274                          skipping null relocation record"
275 @ MSG_REL_NOTSUP        "relocation error: %s: file %s: section [%u]%s: \
276                          relocation not currently supported"
277 @ MSG_REL_PICREDLOC     "relocation error: %s: file %s symbol %s: \
278                          -z redlocsym may not be used for pic code"
279 @ MSG_REL_TLSLE         "relocation error: %s: file %s: symbol %s: \
280                          relocation illegal when building a shared object"
281 @ MSG_REL_TLSBND        "relocation error: %s: file %s: symbol %s: \
282                          bound to: %s: relocation illegal when not bound \
283                          to object being created"
284 @ MSG_REL_TLSSTAT       "relocation error: %s: file %s: symbol %s: \
285                          relocation illegal when building a static object"
286 @ MSG_REL_TLSBADSYM     "relocation error: %s: file %s: symbol %s: \
287                          bad symbol type %s: symbol type must be TLS"
288 @ MSG_REL_BADTLS        "relocation error: %s: file %s: symbol %s: \
289                          relocation illegal for TLS symbol"
290 @ MSG_REL_BADGOTBASED   "relocation error: %s: file %s: symbol %s: a GOT \
291                          relative relocation must reference a local symbol"
292 @ MSG_REL_UNKNWSYM      "relocation error: %s: file %s: section [%u]%s: \
293                          attempt to relocate with respect to unknown \
294                          symbol %s: offset 0x%llx, symbol index %d"
295 @ MSG_REL_UNSUPSZ       "relocation error: %s: file %s: symbol %s: \
296                          offset size (%d bytes) is not supported"
297 @ MSG_REL_INVALOFFSET   "relocation error: %s: file %s section [%u]%s: \
298                          invalid offset symbol '%s': offset 0x%llx"
299 @ MSG_REL_INVALRELT     "relocation error: file %s: section [%u]%s: \
300                          invalid relocation type: 0x%x"
301 @ MSG_REL_EMPTYSEC      "relocation error: %s: file %s: symbol %s: \
302                          attempted against empty section [%u]%s"
303 @ MSG_REL_EXTERNSYM     "relocation error: %s: file %s: symbol %s: \
304                          external symbolic relocation against non-allocatable \
305                          section %s; cannot be processed at runtime: \
306                          relocation ignored"
307 @ MSG_REL_UNEXPREL      "relocation error: %s: file %s: symbol %s: \
308                          unexpected relocation; generic processing performed"
309 @ MSG_REL_UNEXPSYM      "relocation error: %s: file %s: symbol %s: \
310                          unexpected symbol referenced from file %s"
311 @ MSG_REL_SYMDISC       "relocation error: %s: file %s: section [%u]%s: \
312                          symbol %s: symbol has been discarded with discarded \
313                          section: [%u]%s"
314 @ MSG_REL_NOSYMBOL      "relocation error: %s: file %s: section: [%u]%s: \
315                          offset: 0x%llx: relocation requires reference symbol"
316 @ MSG_REL_DISPREL1      "relocation error: %s: file %s: symbol %s: \
317                          displacement relocation applied to the symbol \
318                          %s at 0x%llx: symbol %s is a copy relocated symbol"
319 @ MSG_REL_UNSUPSIZE     "relocation error: %s: file %s: section [%u]%s: \
320                          relocation against section symbol unsupported"

322 @ MSG_REL_DISPREL2      "relocation warning: %s: file %s: symbol %s: \
323                          may contain displacement relocation"
```



```
324 @ MSG_REL_DISPREL3      "relocation warning: %s: file %s: symbol %s: \
325                          displacement relocation applied to the symbol \
326                          %s: at 0x%llx: displacement relocation will not be \
327                          visible in output image"
328 @ MSG_REL_DISPREL4      "relocation warning: %s: file %s: symbol %s: \
329                          displacement relocation to be applied to the symbol \
330                          %s: at 0x%llx: displacement relocation will be \
331                          visible in output image"
332 @ MSG_REL_COPY          "relocation warning: %s: file %s: symbol %s: \
333                          relocation bound to a symbol with STV_PROTECTED \
334                          visibility"
335 @ MSG_RELINVSEC         "relocation warning: %s: file %s: section: [%u]%s: \
336                          against suspicious section [%u]%s; relocation ignored"
337 @ MSG_REL_TLSIE         "relocation warning: %s: file %s: symbol %s: \
338                          relocation has restricted use when building a shared \
339                          object"

341 @ MSG_REL_SLOPCDATNONAM "relocation warning: %s: file %s: section [%u]%s: \
342                          relocation against discarded COMDAT section [%u]%s: \
343                          redirected to file %s"
344 @ MSG_REL_SLOPCDATNAM   "relocation warning: %s: file %s: section [%u]%s: \
345                          symbol %s: relocation against discarded COMDAT \
346                          section [%u]%s: redirected to file %s"
347 @ MSG_REL_SLOPCDATNOSYM "relocation warning: %s: file %s: section [%u]%s: \
348                          symbol %s: relocation against discarded COMDAT \
349                          section [%u]%s: symbol not found, relocation ignored"

351 @ MSG_REL_NOREG         "relocation error: REGISTER relocation not supported \
352                          on target architecture"

354 #
355 # TRANSLATION_NOTE
356 #       The following 7 messages are the message to print the
357 #       following example messages.
358 #
359 #Text relocation remains                        referenced
360 #    against symbol                 offset      in file
361 #str                                 0x14       main.o
362 #printf                              0x1c       main.o
363 #
364 #       The first two lines are the header, and the next msgid
365 #       is the format string for the header.
366 #       Tabs and spaces are used for alignment.
367 #       The first and third %s are for: "Text relocation remains against symbol"
368 #       The second %s and fourth %s are for: "referenced in file"
369 #       The third %s is for: "offset"
370 #
371 @ MSG_REL_REMAIN_FMT_1  "%-40s\t%s\n    %s\t\t    %s\t%s"
372 #
373 # TRANSLATION_NOTE
374 #       The next two msdid make a sentence. So translate:
375 #               "Text relocation remain against symbol"
376 #       And separate them into two msgstr considering the proper
377 #       alignment.
378 @ MSG_REL_RMN_ITM_11    "Text relocation remains"
379 @ MSG_REL_RMN_ITM_12    "against symbol"
380 @ MSG_REL_RMN_ITM_13    "warning: Text relocation remains"

382 @ MSG_REL_RMN_ITM_2     "offset"

384 #
385 # TRANSLATION_NOTE
386 #       The next two msdid make a sentence. So translate:
387 #               "referenced in file"
388 #       And separate them into two msgstr considering the proper
389 #       alignment.
```

```
390 @ MSG_REL_RMN_ITM_31    "referenced"
391 @ MSG_REL_RMN_ITM_32    "in file"
392 @ MSG_REL_REMAIN_2      "%-35s 0x%-8llx\t%s"
393 @ MSG_REL_REMAIN_3      "relocations remain against allocatable but \
394                          non-writable sections"

396 # Files processing messages

398 @ MSG_FIL_MULINC_1      "file %s: attempted multiple inclusion of file"
399 @ MSG_FIL_MULINC_2      "file %s: linked to %s: attempted multiple inclusion \
400                          of file"
401 @ MSG_FIL_SOINSTAT      "input of shared object '%s' in static mode"
402 @ MSG_FIL_INVALSEC      "file %s: section [%u]%s has invalid type %s"
403 @ MSG_FIL_NOTFOUND      "file %s: required by %s, not found"
404 @ MSG_FIL_MALSTR        "file %s: section [%u]%s: malformed string table, \
405                          initial or final byte"
406 @ MSG_FIL_PTHTOLONG     "'%s/%s' pathname too long"
407 @ MSG_FIL_EXCLUDE       "file %s: section [%u]%s contains both SHF_EXCLUDE and \
408                          SHF_ALLOC flags: SHF_EXCLUDE ignored"
409 @ MSG_FIL_INTERRUPT     "file %s: creation interrupted: %s"
410 @ MSG_FIL_INVRELOC1     "file %s: section [%u]%s: relocations can not be \
411                          applied against section [%u]%s"
412 @ MSG_FIL_INVSHINFO     "file %s: section [%u]%s: has invalid sh_info: %lld"
413 @ MSG_FIL_INVSHLINK     "file %s: section [%u]%s: has invalid sh_link: %lld"
414 @ MSG_FIL_INVSHENTSIZE  "file %s: section [%u]%s: has invalid sh_entsize: %lld"
415 @ MSG_FIL_NOSTRTABLE    "file %s: section [%u]%s: symbol[%d]: specifies string \
416                          table offset 0x%llx: no string table is available"
417 @ MSG_FIL_EXCSTRTABLE   "file %s: section [%u]%s: symbol[%d]: specifies string \
418                          table offset 0x%llx: exceeds string table %s: \
419                          size 0x%llx"
420 @ MSG_FIL_NONAMESYM     "file %s: section [%u]%s: symbol[%d]: global symbol has
421                          no name"
422 @ MSG_FIL_UNKCAP        "file %s: section [%u]%s: unknown capability tag: %d"
423 @ MSG_FIL_BADSF1        "file %s: section [%u]%s: unknown software \
424                          capabilities: 0x%llx; ignored"
425 @ MSG_FIL_INADDR32SF1   "file %s: section [%u]%s: software capability ADDR32: is
426                          ineffective when building 32-bit object; ignored"
427 @ MSG_FIL_EXADDR32SF1   "file %s: section [%u]%s: software capability ADDR32: \
428                          requires executable be built with ADDR32 capability"

430 @ MSG_FIL_BADORDREF     "file %s: section [%u]%s: contains illegal reference \
431                          to discarded section: [%u]%s"

433 # Recording name conflicts

435 @ MSG_REC_OPTCNFLT      "recording name conflict: file '%s' and %s provide \
436                          identical dependency names: %s"
437 @ MSG_REC_OBJCNFLT      "recording name conflict: file '%s' and file '%s' \
438                          provide identical dependency names: %s  %s"
439 @ MSG_REC_CNFLTHINT     "(possible multiple inclusion of the same file)"

441 # System call messages

443 @ MSG_SYS_OPEN          "file %s: open failed: %s"
444 @ MSG_SYS_UNLINK        "file %s: unlink failed: %s"
445 @ MSG_SYS_MMAPANON      "mmap anon failed: %s"
446 @ MSG_SYS_MALLOC        "malloc failed: %s"


449 # Messages related to platform support

451 @ MSG_TARG_UNSUPPORTED  "unsupported ELF machine type: %s"


454 # ELF processing messages
```



```
456 @ MSG_ELF_LIBELF        "libelf: version not supported: %d"

458 @ MSG_ELF_ARMEM         "file %s: unable to locate archive member;\n\t\
459                          offset=%x, symbol=%s"

461 @ MSG_ELF_ARSYM         "file %s ignored: unable to locate archive symbol table"

463 @ MSG_ELF_VERSYM        "file %s: version symbol section entry mismatch:\n\t\
464                          (section [%u]%s entries=%d; section [%u]%s entries=%d)"

466 @ MSG_ELF_NOGROUPSECT   "file %s: section [%u]%s: SHF_GROUP flag set, but no \
467                          corresponding SHT_GROUP section found"

469 # Section processing errors

471 @ MSG_SCN_NONALLOC      "%s: non-allocatable section '%s' directed to a \
472                          loadable segment: %s"

474 @ MSG_SCN_MULTICOMDAT   "file %s: section [%u]%s: cannot be susceptible to multi
475                          COMDAT mechanisms: %s"

477 @ MSG_SCN_DWFOVRFLW     "%s: section %s: encoded DWARF data exceeds \
478                          section size"
479 @ MSG_SCN_DWFBADENC     "%s: section %s: invalid DWARF encoding: %#x"

481 # Symbol processing errors

483 @ MSG_SYM_NOSECDEF      "symbol '%s' in file %s has no section definition"
484 @ MSG_SYM_INVSEC        "symbol '%s' in file %s associated with invalid \
485                          section[%lld]"
486 @ MSG_SYM_TLS           "symbol '%s' in file %s (STT_TLS), is defined \
487                          in a non-SHF_TLS section"
488 @ MSG_SYM_BADADDR       "symbol '%s' in file %s: section [%u]%s: size %#llx: \
489                          symbol (address %#llx, size %#llx) lies outside \
490                          of containing section"
491 @ MSG_SYM_BADADDR_ROTXT "symbol '%s' in file %s: readonly text section \
492                          [%u]%s: size %#llx: symbol (address %#llx, \
493                          size %#llx) lies outside of containing section"
494 @ MSG_SYM_MULDEF        "symbol '%s' is multiply-defined:"
495 @ MSG_SYM_CONFVIS       "symbol '%s' has conflicting visibilities:"
496 @ MSG_SYM_DIFFTYPE      "symbol '%s' has differing types:"
497 @ MSG_SYM_DIFFATTR      "symbol '%s' has differing %s:\n\
498                          \t(file %s value=0x%llx; file %s value=0x%llx);"
499 @ MSG_SYM_FILETYPES     "\t(file %s type=%s; file %s type=%s);"
500 @ MSG_SYM_VISTYPES      "\t(file %s visibility=%s; file %s visibility=%s);"
501 @ MSG_SYM_DEFTAKEN      "\t%s definition taken"
502 @ MSG_SYM_DEFUPDATE     "\t%s definition taken and updated with larger size"
503 @ MSG_SYM_LARGER        "\tlargest value applied"
504 @ MSG_SYM_TENTERR       "\ttentative symbol cannot override defined symbol \
505                          of smaller size"

507 @ MSG_SYM_INVSHNDX      "symbol %s has invalid section index; \
508                          ignored:\n\t(file %s value=%s);"
509 @ MSG_SYM_NONGLOB       "global symbol %s has non-global binding:\n\
510                          \t(file %s value=%s);"
511 @ MSG_SYM_RESERVE       "reserved symbol '%s' already defined in file %s"
512 @ MSG_SYM_NOTNULL       "undefined symbol '%s' with non-zero value encountered \
513                          from file %s"
514 @ MSG_SYM_DUPSORTADDR   "section %s: symbol '%s' and symbol '%s' have the \
515                          same address: %#llx: remove duplicate with \
516                          NOSORTSYM mapfile directive"

518 @ MSG_PSYM_INVMINFO1    "file %s: section [%u]%s: entry[%d] has invalid m_info:
519                          0x%llx for symbol index"
520 @ MSG_PSYM_INVMINFO2    "file %s: section [%u]%s: entry[%d] has invalid m_info:
521                          0x%llx for size"
```

```
522 @ MSG_PSYM_INVMREPEAT   "file %s: section [%u]%s: entry[%d] has invalid m_repeat
523                          0x%llx"
524 @ MSG_PSYM_CANNOTEXPND  "file %s: section [%u]%s: entry[%d] can not be expanded:
525                          associated symbol size is unknown %s"
526 @ MSG_PSYM_NOSTATIC     "and partial initialization cannot be deferred to \
527                          a static object"
528 @ MSG_MOVE_OVERLAP      "file %s: section [%u]%s: symbol '%s' overlapping move \
529                          initialization: start=0x%llx, length=0x%llx: \
530                          start=0x%llx, length=0x%llx"
531 @ MSG_PSYM_EXPREASON1   "output file is static object"
532 @ MSG_PSYM_EXPREASON2   "-z nopartial option in effect"
533 @ MSG_PSYM_EXPREASON3   "move infrastructure size is greater than move data"

535 #
536 # Support library failures
537 #
538 @ MSG_SUP_NOLOAD        "dlopen() of support library (%s) failed with \
539                          error: %s"
540 @ MSG_SUP_BADVERSION    "initialization of support library (%s) failed with \
541                          bad version.  supported: %d returned: %d"


544 #
545 # TRANSLATION_NOTE
546 #       The following 7 messages are the message to print the
547 #       following example messages.
548 #
549 #Undefined                       first referenced
550 # symbol                            in file
551 #inquire                           halt_hold.o
552 #
553 @ MSG_SYM_FMT_UNDEF     "%s\t\t\t%s\
554                          \n %s  \t\t\t    %s"

556 #
557 # TRANSLATION_NOTE
558 #       The next two msdid make a sentence. So translate:
559 #               "Undefined symbol"
560 #       And separate them into two msgstr considering the proper
561 #       alignment.
562 @ MSG_SYM_UNDEF_ITM_11  "Undefined"
563 @ MSG_SYM_UNDEF_ITM_12  "symbol"
564 #
565 # TRANSLATION_NOTE
566 #       The next two msdid make a sentence. So translate:
567 #               "first referenced in file"
568 #       And separate them into two msgstr considering the proper
569 #       alignment.
570 @ MSG_SYM_UNDEF_ITM_21  "first referenced"
571 @ MSG_SYM_UNDEF_ITM_22  "in file"
572 #

574 @ MSG_SYM_UND_UNDEF     "%-35s %s"
575 @ MSG_SYM_UND_NOVER     "%-35s %s  (symbol has no version assigned)"
576 @ MSG_SYM_UND_IMPL      "%-35s %s  (symbol belongs to implicit dependency %s)"
577 @ MSG_SYM_UND_NOTA      "%-35s %s  (symbol belongs to unavailable version %s \
578                          (%s))"
579 @ MSG_SYM_UND_BNDLOCAL  "%-35s %s  (symbol scope specifies local binding)"

581 @ MSG_SYM_ENTRY         "entry point"
582 @ MSG_SYM_UNDEF         "%s symbol '%s' is undefined"
583 @ MSG_SYM_EXTERN        "%s symbol '%s' is undefined  (symbol belongs to \
584                          dependency %s)"
585 @ MSG_SYM_NOCRT         "symbol '%s' not found, but %s section exists - \
586                          possible link-edit without using the compiler driver"
```



```
588 # Output file update messages

590 @ MSG_UPD_NOREADSEG     "No read-only segments found.  Setting '_etext' to 0"
591 @ MSG_UPD_NORDWRSEG     "No read-write segments found.  Setting '_edata' to 0"
592 @ MSG_UPD_NOSEG         "Setting 'end' and '_end' to 0"

594 @ MSG_UPD_SEGOVERLAP    "%s: segment address overlap;\n\
595                          \tprevious segment ending at address 0x%llx overlaps\n\
596                          \tuser defined segment '%s' starting at address 0x%llx"
597 @ MSG_UPD_LARGSIZE      "%s: segment %s calculated size 0x%llx\n\
598                          \tis larger than user-defined size 0x%llx"

600 @ MSG_UPD_NOBITS        "NOBITS section found before end of initialized data"
601 @ MSG_SEG_FIRNOTLOAD    "First segment has type %s, PT_LOAD required: %s"
602 @ MSG_UPD_MULEHFRAME    "file %s; section [%u]%s and file %s; section [%u]%s \
603                          have incompatibile attributes and cannot \
604                          be merged into a single output section"


607 # Version processing messages

609 @ MSG_VER_HIGHER        "file %s: version revision %d is higher than \
610                          expected %d"
611 @ MSG_VER_NOEXIST       "file %s: version '%s' does not exist:\n\
612                          \trequired by file %s"
613 @ MSG_VER_UNDEF         "version '%s' undefined, referenced by version '%s':\n\
614                          \trequired by file %s"
615 @ MSG_VER_UNAVAIL       "file %s: version '%s' is unavailable:\n\
616                          \trequired by file %s"
617 @ MSG_VER_DEFINED       "version symbol '%s' already defined in file %s"
618 @ MSG_VER_INVALNDX      "version symbol '%s' from file %s has an invalid \
619                          version index (%d)"
620 @ MSG_VER_ADDVERS       "unused $ADDVERS specification from file '%s' \
621                          for object '%s'\nversion(s):"
622 @ MSG_VER_ADDVER        "\t%s"
623 @ MSG_VER_CYCLIC        "following versions generate cyclic dependency:"

625 # Capabilities messages

627 @ MSG_CAP_MULDEF        "capabilities symbol '%s' has multiply-defined members:"
628 @ MSG_CAP_MULDEFSYMS    "\t(file %s symbol '%s'; file %s symbol '%s');"
629 @ MSG_CAP_REDUNDANT     "file %s: section [%u]%s: symbol capabilities \
630                          redundant, as object capabilities are more restrictive"
631 @ MSG_CAP_NOSYMSFOUND   "no global symbols have been found that are associated \
632                          with capabilities identified relocatable objects: \
633                          -z symbolcap has no effect"

635 @ MSG_CAPINFO_INVALSYM  "file %s: capabilities info section [%u]%s: index %d: \
636                          family member symbol '%s': invalid"
637 @ MSG_CAPINFO_INVALLEAD "file %s: capabilities info section [%u]%s: index %d: \
638                          family lead symbol '%s': invalid symbol index %d"

640 # Basic strings

642 @ MSG_STR_ALIGNMENTS    "alignments"
643 @ MSG_STR_COMMAND       "(command line)"
644 @ MSG_STR_TLSREL        "(internal TLS relocation requirement)"
645 @ MSG_STR_SIZES         "sizes"
646 @ MSG_STR_UNKNOWN       "<unknown>"
647 @ MSG_STR_SECTION       "%s (section)"
648 @ MSG_STR_SECTION_MSTR  "%s (merged string section)"

650 #
651 # TRANSLATION_NOTE
652 #       The elf_ function name represents a man page reference and should not
653 #       be translated.
```

```
654 @ MSG_ELF_BEGIN         "file %s: elf_begin"
655 @ MSG_ELF_CNTL          "file %s: elf_cntl"
656 @ MSG_ELF_GETARHDR      "file %s: elf_getarhdr"
657 @ MSG_ELF_GETARSYM      "file %s: elf_getarsym"
658 @ MSG_ELF_GETDATA       "file %s: elf_getdata"
659 @ MSG_ELF_GETEHDR       "file %s: elf_getehdr"
660 @ MSG_ELF_GETPHDR       "file %s: elf_getphdr"
661 @ MSG_ELF_GETSCN        "file %s: elf_getscn: scnndx: %d"
662 @ MSG_ELF_GETSHDR       "file %s: elf_getshdr"
663 @ MSG_ELF_MEMORY        "file %s: elf_memory"
664 @ MSG_ELF_NDXSCN        "file %s: elf_ndxscn"
665 @ MSG_ELF_NEWDATA       "file %s: elf_newdata"
666 @ MSG_ELF_NEWEHDR       "file %s: elf_newehdr"
667 @ MSG_ELF_NEWSCN        "file %s: elf_newscn"
668 @ MSG_ELF_NEWPHDR       "file %s: elf_newphdr"
669 @ MSG_ELF_STRPTR        "file %s: elf_strptr"
670 @ MSG_ELF_UPDATE        "file %s: elf_update"
671 @ MSG_ELF_SWAP_WRIMAGE  "file %s: _elf_swap_wrimage"


674 @ MSG_REJ_MACH          "file %s: wrong ELF machine type: %s"
675 @ MSG_REJ_CLASS         "file %s: wrong ELF class: %s"
676 @ MSG_REJ_DATA          "file %s: wrong ELF data format: %s"
677 @ MSG_REJ_TYPE          "file %s: bad ELF type: %s"
678 @ MSG_REJ_BADFLAG       "file %s: bad ELF flags value: %s"
679 @ MSG_REJ_MISFLAG       "file %s: mismatched ELF flags value: %s"
680 @ MSG_REJ_VERSION       "file %s: mismatched ELF/lib version: %s"
681 @ MSG_REJ_HAL           "file %s: HAL R1 extensions required"
682 @ MSG_REJ_US3           "file %s: Sun UltraSPARC III extensions required"
683 @ MSG_REJ_STR           "file %s: %s"
684 @ MSG_REJ_UNKFILE       "file %s: unknown file type"
685 @ MSG_REJ_UNKCAP        "file=%s; unknown capability: %d"
686 @ MSG_REJ_HWCAP_1       "file %s: hardware capability (CA_SUNW_HW_1) \
687                          unsupported: %s"
688 @ MSG_REJ_SFCAP_1       "file %s: software capability (CA_SUNW_SF_1) \
689                          unsupported: %s"
690 @ MSG_REJ_MACHCAP       "file %s: machine capability (CA_SUNW_MACH) \
691                          unsupported: %s"
692 @ MSG_REJ_PLATCAP       "file %s: platform capability (CA_SUNW_PLAT) \
693                          unsupported: %s"
694 @ MSG_REJ_HWCAP_2       "file %s: hardware capability (CA_SUNW_HW_2) \
695                          unsupported: %s"
696 @ MSG_REJ_ARCHIVE       "file %s: invalid archive use"

698 # Guidance messages
699 @ MSG_GUIDE_SUMMARY     "see ld(1) -z guidance for more information"
700 @ MSG_GUIDE_DEFS        "-z defs option recommended for shared objects"
701 @ MSG_GUIDE_DIRECT      "-B direct or -z direct option recommended before \
702                          first dependency"
703 @ MSG_GUIDE_LAZYLOAD    "-z lazyload option recommended before \
704                          first dependency"
705 @ MSG_GUIDE_MAPFILE     "version 2 mapfile syntax recommended: %s"
706 @ MSG_GUIDE_TEXT        "position independent (PIC) code recommended for \
707                          shared objects"
708 @ MSG_GUIDE_UNUSED      "removal of unused dependency recommended: %s"

710 @ _END_


713 # The following strings represent reserved names.  Reference to these strings
714 # is via the MSG_ORIG() macro, and thus translations are not required.

716 @ MSG_STR_EOF           "<eof>"
717 @ MSG_STR_ERROR         "<error>"
718 @ MSG_STR_EMPTY         ""
719 @ MSG_QSTR_BANG         "'!'"
```



```
720 @ MSG_STR_COLON         ":"
721 @ MSG_QSTR_COLON        "':'"
722 @ MSG_QSTR_SEMICOLON    "';'"
723 @ MSG_QSTR_EQUAL        "'='"
724 @ MSG_QSTR_PLUSEQ       "'+='"
725 @ MSG_QSTR_MINUSEQ      "'-='"
726 @ MSG_QSTR_ATSIGN       "'@'"
727 @ MSG_QSTR_DASH         "'-'"
728 @ MSG_QSTR_LEFTBKT      "'{'"
729 @ MSG_QSTR_RIGHTBKT     "'}'"
730 @ MSG_QSTR_PIPE         "'|'"
731 @ MSG_QSTR_STAR         "'*'"
732 @ MSG_STR_DOT           "."
733 @ MSG_STR_SLASH         "/"
734 @ MSG_STR_COMMA         ","
735 @ MSG_STR_DYNAMIC       "(.dynamic)"
736 @ MSG_STR_ORIGIN        "$ORIGIN"
737 @ MSG_STR_MACHINE       "$MACHINE"
738 @ MSG_STR_PLATFORM      "$PLATFORM"
739 @ MSG_STR_ISALIST       "$ISALIST"
740 @ MSG_STR_OSNAME        "$OSNAME"
741 @ MSG_STR_OSREL         "$OSREL"
742 @ MSG_STR_UU_REAL_U     "__real_"
743 @ MSG_STR_UU_WRAP_U     "__wrap_"
744 @ MSG_STR_UELF32        "_ELF32"
745 @ MSG_STR_UELF64        "_ELF64"
746 @ MSG_STR_USPARC        "_sparc"
747 @ MSG_STR_UX86          "_x86"
748 @ MSG_STR_TRUE          "true"

750 @ MSG_STR_CDIR_ADD      "$add"
751 @ MSG_STR_CDIR_CLEAR    "$clear"
752 @ MSG_STR_CDIR_ERROR    "$error"
753 @ MSG_STR_CDIR_MFVER    "$mapfile_version"
754 @ MSG_STR_CDIR_IF       "$if"
755 @ MSG_STR_CDIR_ELIF     "$elif"
756 @ MSG_STR_CDIR_ELSE     "$else"
757 @ MSG_STR_CDIR_ENDIF    "$endif"

759 @ MSG_STR_GROUP         "GROUP"
760 @ MSG_STR_SUNW_COMDAT   "SUNW_COMDAT"

762 @ MSG_FMT_ARMEM         "%s(%s)"
763 @ MSG_FMT_COLPATH       "%s:%s"
764 @ MSG_FMT_SYMNAM        "'%s'"
765 @ MSG_FMT_NULLSYMNAM    "%s[%d]"
766 @ MSG_FMT_STRCAT        "%s%s"

768 @ MSG_PTH_RTLD          "/usr/lib/ld.so.1"

770 @ MSG_SUNW_OST_SGS      "SUNW_OST_SGS"


773 # Section strings

775 @ MSG_SCN_BSS           ".bss"
776 @ MSG_SCN_DATA          ".data"
777 @ MSG_SCN_COMMENT       ".comment"
778 @ MSG_SCN_DEBUG         ".debug"
779 @ MSG_SCN_DEBUG_INFO    ".debug_info"
780 @ MSG_SCN_DYNAMIC       ".dynamic"
781 @ MSG_SCN_DYNSYMSORT    ".SUNW_dynsymsort"
782 @ MSG_SCN_DYNTLSSORT    ".SUNW_dyntlssort"
783 @ MSG_SCN_DYNSTR        ".dynstr"
784 @ MSG_SCN_DYNSYM        ".dynsym"
785 @ MSG_SCN_DYNSYM_SHNDX  ".dynsym_shndx"
```

```
786 @ MSG_SCN_LDYNSYM       ".SUNW_ldynsym"
787 @ MSG_SCN_LDYNSYM_SHNDX ".SUNW_ldynsym_shndx"
788 @ MSG_SCN_EX_SHARED     ".ex_shared"
789 @ MSG_SCN_EX_RANGES     ".exception_ranges"
790 @ MSG_SCN_EXCL          ".excl"
791 @ MSG_SCN_FINI          ".fini"
792 @ MSG_SCN_FINIARRAY     ".fini_array"
793 @ MSG_SCN_GOT           ".got"
794 @ MSG_SCN_GNU_LINKONCE  ".gnu.linkonce."
795 @ MSG_SCN_HASH          ".hash"
796 @ MSG_SCN_INDEX         ".index"
797 @ MSG_SCN_INIT          ".init"
798 @ MSG_SCN_INITARRAY     ".init_array"
799 @ MSG_SCN_INTERP        ".interp"
800 @ MSG_SCN_LBSS          ".lbss"
801 @ MSG_SCN_LDATA         ".ldata"
802 @ MSG_SCN_LINE          ".line"
803 @ MSG_SCN_LRODATA       ".lrodata"
804 @ MSG_SCN_PLT           ".plt"
805 @ MSG_SCN_PREINITARRAY  ".preinit_array"
806 @ MSG_SCN_REL           ".rel"
807 @ MSG_SCN_RELA          ".rela"
808 @ MSG_SCN_RODATA        ".rodata"
809 @ MSG_SCN_SBSS          ".sbss"
810 @ MSG_SCN_SBSS2         ".sbss2"
811 @ MSG_SCN_SDATA         ".sdata"
812 @ MSG_SCN_SDATA2        ".sdata2"
813 @ MSG_SCN_SHSTRTAB      ".shstrtab"
814 @ MSG_SCN_STAB          ".stab"
815 @ MSG_SCN_STABEXCL      ".stab.exclstr"
816 @ MSG_SCN_STRTAB        ".strtab"
817 @ MSG_SCN_SUNWMOVE      ".SUNW_move"
818 @ MSG_SCN_SUNWRELOC     ".SUNW_reloc"
819 @ MSG_SCN_SUNWSYMINFO   ".SUNW_syminfo"
820 @ MSG_SCN_SUNWVERSION   ".SUNW_version"
821 @ MSG_SCN_SUNWVERSYM    ".SUNW_versym"
822 @ MSG_SCN_SUNWCAP       ".SUNW_cap"
823 @ MSG_SCN_SUNWCAPINFO   ".SUNW_capinfo"
824 @ MSG_SCN_SUNWCAPCHAIN  ".SUNW_capchain"
825 @ MSG_SCN_SYMTAB        ".symtab"
826 @ MSG_SCN_SYMTAB_SHNDX  ".symtab_shndx"
827 @ MSG_SCN_TBSS          ".tbss"
828 @ MSG_SCN_TDATA         ".tdata"
829 @ MSG_SCN_TEXT          ".text"

831 @ MSG_SYM_FINIARRAY     "finiarray"
832 @ MSG_SYM_INITARRAY     "initarray"
833 @ MSG_SYM_PREINITARRAY  "preinitarray"

835 #
836 # GNU section names
837 #
838 @ MSG_SCN_CTORS         ".ctors"
839 @ MSG_SCN_DTORS         ".dtors"
840 @ MSG_SCN_EHFRAME       ".eh_frame"
841 @ MSG_SCN_EHFRAME_HDR   ".eh_frame_hdr"
842 @ MSG_SCN_GCC_X_TBL     ".gcc_except_table"
843 @ MSG_SCN_JCR           ".jcr"

845 # Segment names for segments referenced by entrance criteria

847 @ MSG_ENT_BSS           "bss"
848 @ MSG_ENT_DATA          "data"
849 @ MSG_ENT_EXTRA         "extra"
850 @ MSG_ENT_LDATA         "ldata"
851 @ MSG_ENT_LRODATA       "lrodata"
```



```
852 @ MSG_ENT_NOTE          "note"
853 @ MSG_ENT_TEXT          "text"

855 # Symbol names

857 @ MSG_SYM_START         "_start"
858 @ MSG_SYM_MAIN          "main"

860 @ MSG_SYM_FINI_U        "_fini"
861 @ MSG_SYM_INIT_U        "_init"
862 @ MSG_SYM_DYNAMIC       "DYNAMIC"
863 @ MSG_SYM_DYNAMIC_U     "_DYNAMIC"
864 @ MSG_SYM_EDATA         "edata"
865 @ MSG_SYM_EDATA_U       "_edata"
866 @ MSG_SYM_END           "end"
867 @ MSG_SYM_END_U         "_end"
868 @ MSG_SYM_ETEXT         "etext"
869 @ MSG_SYM_ETEXT_U       "_etext"
870 @ MSG_SYM_GOFTBL        "GLOBAL_OFFSET_TABLE_"
871 @ MSG_SYM_GOFTBL_U      "_GLOBAL_OFFSET_TABLE_"
872 @ MSG_SYM_PLKTBL        "PROCEDURE_LINKAGE_TABLE_"
873 @ MSG_SYM_PLKTBL_U      "_PROCEDURE_LINKAGE_TABLE_"
874 @ MSG_SYM_TLSGETADDR_U  "__tls_get_addr"
875 @ MSG_SYM_TLSGETADDR_UU "___tls_get_addr"

877 @ MSG_SYM_L_END         "END_"
878 @ MSG_SYM_L_END_U       "_END_"
879 @ MSG_SYM_L_START       "START_"
880 @ MSG_SYM_L_START_U     "_START_"

882 @ MSG_SYM_SECBOUND_START "__start_"
883 @ MSG_SYM_SECBOUND_STOP "__stop_"

885 #endif /* ! codereview */
886 # Support functions

888 @ MSG_SUP_VERSION       "ld_version"
889 @ MSG_SUP_INPUT_DONE    "ld_input_done"

891 @ MSG_SUP_START_64      "ld_start64"
892 @ MSG_SUP_ATEXIT_64     "ld_atexit64"
893 @ MSG_SUP_OPEN_64       "ld_open64"
894 @ MSG_SUP_FILE_64       "ld_file64"
895 @ MSG_SUP_INSEC_64      "ld_input_section64"
896 @ MSG_SUP_SEC_64        "ld_section64"

898 @ MSG_SUP_START         "ld_start"
899 @ MSG_SUP_ATEXIT        "ld_atexit"
900 @ MSG_SUP_OPEN          "ld_open"
901 @ MSG_SUP_FILE          "ld_file"
902 @ MSG_SUP_INSEC         "ld_input_section"
903 @ MSG_SUP_SEC           "ld_section"

905 #
906 # Message previously in 'ld'
907 #
908 #
909 @ _START_

911 # System error messages

913 @ MSG_SYS_STAT          "file %s: stat failed: %s"
914 @ MSG_SYS_READ          "file %s: read failed: %s"
915 @ MSG_SYS_NOTREG        "file %s: is not a regular file"

917 # Argument processing messages
```

```
919 @ MSG_ARG_DY_INCOMP     "%s option is incompatible with building a dynamic \
920                          executable"
921 @ MSG_MARG_DY_INCOMP    "%s is incompatible with building a dynamic \
922                          executable"
923 @ MSG_ARG_ST_INCOMP     "%s option is incompatible with building a static \
924                          object (-dn, -r, --relocatable)"
925 @ MSG_MARG_ST_INCOMP    "%s is incompatible with building a static \
926                          object (-dn, -r, --relocatable)"
927 @ MSG_MARG_ST_ONLYAVL   "%s is only available when building a shared object"
928 @ MSG_ARG_INCOMP        "option %s and %s are incompatible"
929 @ MSG_MARG_INCOMP       "%s and %s are incompatible"
930 @ MSG_ARG_MTONCE        "option %s appears more than once, first setting taken"
931 @ MSG_MARG_MTONCE       "%s appears more than once, first setting taken"
932 @ MSG_ARG_ILLEGAL       "option %s has illegal argument '%s'"
933 @ MSG_ARG_YP            "option -YP and -Y%c may not be specified concurrently"
934 @ MSG_ARG_STRIP         "%s specified with %s; only debugging \
935                          information stripped"
936 @ MSG_ARG_NOFILES       "no files on input command line"
937 @ MSG_ARG_NOFLTR        "option %s is only meaningful when building a filter"
938 @ MSG_ARG_NODEFLIB      "the default library search path has been suppressed, \
939                          but no runpaths have been specified via %s"
940 @ MSG_ARG_NOENTRY       "entry point symbol '%s' is undefined"
941 @ MSG_ARG_UNSUPPORTED   "option %s is no longer supported; ignored"
942 @ MSG_MARG_ONLY         "option %s can only be used with a %s"
943 @ MSG_ARG_UNKNOWN       "unrecognized option '-%c'"
944 @ MSG_ARG_LONG_UNKNOWN  "unrecognized option '%s'"
945 @ MSG_ARG_USEHELP       "use the -z help option for usage information"


948 @ MSG_ARG_FLAGS         "flags processing errors"
949 @ MSG_ARG_FILES         "file processing errors. No output written to %s"
950 @ MSG_ARG_SYM_WARN      "symbol referencing errors"
951 @ MSG_ARG_SYM_FATAL     "symbol referencing errors. No output written to %s"
952 @ MSG_ARG_AR_GRP_OLAP   "%s cannot be nested"
953 @ MSG_ARG_AR_GRP_BAD    "%s used without corresponding %s"


956 # Messages used to refer to options where there is more than
957 # one name accepted.

959 @ MSG_MARG_AR_GRPS      "archive rescan groups \
960                          (-z rescan-start, -(, --start-group)"
961 @ MSG_MARG_AR_GRP_END   "archive rescan group end option \
962                          (-z rescan-end, -), --end-group)"
963 @ MSG_MARG_AR_GRP_START "archive rescan group start option \
964                          (-z rescan-start, -(, --start-group)"
965 @ MSG_MARG_ENTRY        "entry point option (-e, --entry)"
966 @ MSG_MARG_FILTER_AUX   "auxiliary filter option (-f, --auxiliary)"
967 @ MSG_MARG_FILTER       "filter option (-F, --filter)"
968 @ MSG_MARG_OUTFILE      "output object option (-o, --output)"
969 @ MSG_MARG_REL          "relocatable object option (-r, --relocatable)"
970 @ MSG_MARG_RPATH        "runpath option (-R, -rpath)"
971 @ MSG_MARG_SO           "shared object option (-G, -shared)"
972 @ MSG_MARG_SONAME       "soname option (-h, --soname)"
973 @ MSG_MARG_STRIP        "strip option (-s, --strip-all)"

975 # Entrance criteria messages

977 @ MSG_ENT_MAP_FMT_TIL_1 "\t\t%s\n\n"
978 @ MSG_ENT_MAP_TITLE_1   "LINK EDITOR MEMORY MAP"

980 #
981 # TRANSLATION_NOTE -- Entry map header
982 #
983 # The next message is a format string for a title. The title is composed of
```



```
984 # two lines. In C locale, it would look like:
985 #
986 #       output          input           new
987 #       section         section         displacement    size
988 #
989 # The \t characters are used for alignment.  (output section), (input section),
990 # and (new displacement) have to be aligned.
991 #
992 @ MSG_ENT_MAP_FMT_TIL_2 "\n%s\t\t%s\t\t%s\n%s\t\t%s\t\t%s\t%s\n\n"
993 @ MSG_ENT_MAP_FMT_TIL_3 "\n%s\t\t%s\t\t%s\n%s\t\t%s\t\t%s\t%s\n\n"
994 @ MSG_ENT_ITM_OUTPUT    "output"
995 @ MSG_ENT_ITM_INPUT     "input"
996 @ MSG_ENT_ITM_NEW       "new"
997 @ MSG_ENT_ITM_SECTION   "section"
998 @ MSG_ENT_ITM_DISPMNT   "displacement"
999 @ MSG_ENT_ITM_SIZE      "size"
1000 @ MSG_ENT_ITM_VIRTUAL   "virtual"
1001 @ MSG_ENT_ITM_ADDRESS   "address"

1003 @ MSG_ENT_MAP_ENTRY_1   "%-8.8s\t\t\t%08.2llx\t%08.2llx\n"
1004 @ MSG_ENT_MAP_ENTRY_2   "\t\t%-8.8s\t%08.2llx\t%08.2llx %s\n"

1006 #
1007 # TRANSLATION_NOTE -- multiple defined symbol table header
1008 #
1009 # In C locale, an example output is:
1010 #
1011 #               MULTIPLY DEFINED SYMBOLS
1012 #
1013 #
1014 #symbol                          definition used     also defined in
1015 #
1016 #variable1                          main.o
1017 #                                                      ./libfred.so
1018 @ MSG_ENT_MUL_FMT_TIL_0 "\n\n\t\t%s\n\n\n"
1019 @ MSG_ENT_MUL_TIL_0     "MULTIPLY DEFINED SYMBOLS"

1021 #
1022 # TRANSLATION_NOTE -- This is the format string for:
1023 #
1024 #symbol                          definition used     also defined in
1025 #
1026 @ MSG_ENT_MUL_FMT_TIL_1 "%s\t\t\t\t  %s     %s\n\n"
1027 @ MSG_ENT_MUL_ITM_SYM   "symbol"
1028 @ MSG_ENT_MUL_ITM_DEF_0 "definition used"
1029 @ MSG_ENT_MUL_ITM_DEF_1 "also defined in"

1031 #
1032 # TRANSLATION_NOTE -- This is the format string for the second item:
1033 #
1034 @ MSG_ENT_MUL_ENTRY_1   "%-35s %s\n"

1036 #
1037 # TRANSLATION_NOTE -- This is the format string for the third item:
1038 #
1039 @ MSG_ENT_MUL_ENTRY_2   "\t\t\t\t\t\t\t%s\n"

1041 @ MSG_ENT_NOSEC_1       "mapfile: %s segment: section '%s' does not appear \
1042                          in mapfile specified input file(s)"
1043 @ MSG_ENT_NOSEC_2       "mapfile: %s segment: section '%s' does not appear \
1044                          in any input file"

1046 # Library messages

1048 @ MSG_LIB_NOTFOUND      "library -l%s: not found"
1049 @ MSG_LIB_MALFORM       "LD_LIBRARY_PATH malformed"
```

```
@ MSG_LIB_BADYP         "-YP library path malformed"


# Mapfile processing messages

@ MSG_MAP_BADAUTORED    "%s: %llu: auto-reduction (’*’) can only be used in \
                         hidden/local, or eliminate scope"
@ MSG_MAP_BADFLAG       "%s: %llu: badly formed section flags ’%s’"
@ MSG_MAP_BADBNAME      "%s: %llu: basename cannot contain path \
                         separator (’/’): %s"
@ MSG_MAP_BADONAME      "%s: %llu: object name cannot contain path \
                         separator (’/’): %s"
@ MSG_MAP_REDEFATT      "%s: %llu: redefining %s attribute for ’%s’"
@ MSG_MAP_PREMEOF       "%s: %llu: premature EOF"
@ MSG_MAP_ILLCHAR       "%s: %llu: illegal character ’\\%03o’"
@ MSG_MAP_MALFORM       "%s: %llu: malformed entry"
@ MSG_MAP_NONLOAD       "%s: %llu: %s not allowed on non-LOAD segments"
@ MSG_MAP_NOSTACK1      "%s: %llu: %s not allowed on STACK segment"
@ MSG_MAP_MOREONCE      "%s: %llu: %s set more than once on same line"
@ MSG_MAP_NOTERM        "%s: %llu: unterminated quoted string: %s"
@ MSG_MAP_SECINSEG      "%s: %llu: section within segment ordering done on \
                         a non-existent segment ’%s’"
@ MSG_MAP_UNEXINHERIT   "%s: %llu: unnamed version cannot inherit from other \
                         versions: %s"
@ MSG_MAP_UNEXTOK       "%s: %llu: unexpected occurrence of ’%c’ token"

@ MSG_MAP_SEGEMPLOAD    "%s: %llu: empty segment must be of type LOAD or NULL"
@ MSG_MAP_SEGEMPEXE     "%s: %llu: a LOAD empty segment definition is only \
                         allowed when creating a dynamic executable"
@ MSG_MAP_SEGEMPATT     "%s: %llu: a LOAD empty segment must have an address \
                         and size"
@ MSG_MAP_SEGEMPNOATT   "%s: %llu: a NULL empty segment must not have an \
                         address or size"
@ MSG_MAP_SEGEMPSEC     "%s: %llu: empty segment can not have sections \
                         assigned to it"
@ MSG_MAP_SEGEMNOPERM   "%s: %llu: empty segment must not have \
                         p_flags set: 0x%x"

@ MSG_MAP_CNTADDRORDER  "%s: %llu: segment cannot have an explicit address \
                         and also be in the SEGMENT_ORDER list: %s"
@ MSG_MAP_CNTDISSEG     "%s: %llu: segment cannot be disabled: %s"
@ MSG_MAP_DUPNAMENT     "%s: %llu: cannot redefine entrance criteria: %s"
@ MSG_MAP_DUPORDSEG     "%s: %llu: segment is already in %s list: %s"
@ MSG_MAP_DUP_OS_ORD    "%s: %llu: section is already in OS_ORDER list: %s"
@ MSG_MAP_DUP_IS_ORD    "%s: %llu: entrance criteria is already in \
                         IS_ORDER list: %s"
@ MSG_MAP_UNKENT        "%s: %llu: unknown entrance criteria \
                         (ASSIGN_SECTION): %s"
@ MSG_MAP_UNKSEG        "%s: %llu: unknown segment: %s"
@ MSG_MAP_UNKSYMDEF     "%s: %llu: unknown symbol definition: %s"
@ MSG_MAP_UNKSEGTYP     "%s: %llu: unknown internal segment type %d"
@ MSG_MAP_UNKSOTYP      "%s: %llu: unknown shared object type: %s"
@ MSG_MAP_UNKSEGATT     "%s: %llu: unknown segment attribute: %s"
@ MSG_MAP_UNKSEGFLG     "%s: %llu: unknown segment flag: ?%c"
@ MSG_MAP_UNKSECTYP     "%s: %llu: unknown section type: %s"

@ MSG_MAP_SEGSIZE       "%s: %lld: existing segment size symbols cannot \
                         be reset: %s"
@ MSG_MAP_SEGADDR       "%s: %llu: segment address or length ’%s’ %s"
@ MSG_MAP_BADCAPVAL     "%s: %llu: bad capability value: %s"
@ MSG_MAP_UNKCAPATTR    "%s: %llu: unknown capability attribute ’%s’"
@ MSG_MAP_EMPTYCAP      "%s: %llu: empty capability definition; ignored"

@ MSG_MAP_SYMDEF1       "%s: %llu: symbol ’%s’ is already defined in file: \
                         %s: %s"
@ MSG_MAP_SYMDEF2       "%s: %llu: symbol ’%s’: %s"
```



```
@ MSG_MAP_EXPSCOL       "%s: %llu: expected a ’;’"
@ MSG_MAP_EXPEQU        "%s: %llu: expected a ’=’, ’:’, ’|’, or ’@’"
@ MSG_MAP_EXPSEGATT     "%s: %llu: expected one or more segment attributes \
                         after an ’=’"
@ MSG_MAP_EXPSEGNAM     "%s: %llu: expected a segment name at the beginning \
                         of a line"
@ MSG_MAP_EXPSEGTYPE    "%s: %llu: %s segment cannot be used with %s \
                         directive: %s"
@ MSG_MAP_EXPSYM_1      "%s: %llu: expected a symbol name after ’@’"
@ MSG_MAP_EXPSYM_2      "%s: %llu: expected a symbol name after ’{’"
@ MSG_MAP_EXPSEC        "%s: %llu: expected a section name after ’|’"
@ MSG_MAP_EXPSO         "%s: %llu: expected a shared object definition \
                         after ’-’"
@ MSG_MAP_MULTFILTEE    "%s: %llu: multiple filtee definitions are unsupported"
@ MSG_MAP_NOFILTER      "%s: %llu: filtee definition required"
@ MSG_MAP_BADSF1        "%s: %llu: unknown software capabilities: 0x%llx; \
                         ignored"
@ MSG_MAP_INADDR32SF1   "%s: %llu: software capability ADDR32: is ineffective \
                         when building 32-bit object: ignored"
@ MSG_MAP_NOINTPOSE     "%s: %llu: interposition symbols can only be defined \
                         when building a dynamic executable"
@ MSG_MAP_NOEXVLSZ      "%s: %llu: value and size attributes are incompatible \
                         with extern or parent symbols"
@ MSG_MAP_FLTR_ONLYAVL  "%s: %llu: symbol filtering is only available when \
                         building a shared object"

@ MSG_MAP_SEGSAME       "segments ’%s’ and ’%s’ have the same assigned \
                         virtual address"
@ MSG_MAP_EXCLIMIT      "exceeds internal limit"
@ MSG_MAP_NOBADFRM      "number is badly formed"

@ MSG_MAP_SEGTYP        "segment type"
@ MSG_MAP_SEGVADDR      "segment virtual address"
@ MSG_MAP_SEGPHYS       "segment physical address"
@ MSG_MAP_SEGLEN        "segment length"
@ MSG_MAP_SEGFLAG       "segment flags"
@ MSG_MAP_SEGALIGN      "segment alignment"
@ MSG_MAP_SEGROUND      "segment rounding"

@ MSG_MAP_SECTYP        "section type"
@ MSG_MAP_SECFLAG       "section flags"
@ MSG_MAP_SECNAME       "section name"

@ MSG_MAP_SYMVAL        "symbol value"
@ MSG_MAP_SYMSIZE       "symbol size"

@ MSG_MAP_DIFF_SYMVAL   "symbol values differ"
@ MSG_MAP_DIFF_SYMSZ    "symbol sizes differ"
@ MSG_MAP_DIFF_SYMTYP   "symbol types differ"
@ MSG_MAP_DIFF_SYMNDX   "symbol indexes differ"
@ MSG_MAP_DIFF_SYMLCL   "symbol scope conflict against local and non-local"
@ MSG_MAP_DIFF_SYMGLOB  "symbol scope conflict against singleton/exported"
@ MSG_MAP_DIFF_SYMPROT  "symbol scope conflict against protected"
@ MSG_MAP_DIFF_SYMVER   "symbol version conflict"
@ MSG_MAP_DIFF_SYMMUL   "symbol multiple definition"
@ MSG_MAP_DIFF_SNGLDIR  "singleton scope and direct declaration are \
                         incompatible"
@ MSG_MAP_DIFF_PROTNDIR "protected scope and no-direct declaration \
                         are incompatible"


@ MSG_MAP_SECORDER      "section ordering requested, but no matching section \
                         found: segment: %s section: %s"
```

```
# Mapfile Directives

@ MSG_MAP_EXP_ATTR      "%s: %llu: expected attribute name (%s), or \
                         terminator (';', '}'): %s"
@ MSG_MAP_EXP_CAPMASK   "%s: %llu: expected capability name, integer value, or \
                         terminator (';', '}'): %s"
@ MSG_MAP_EXP_CAPNAME   "%s: %llu: expected name, or terminator (';', '}'): %s"
@ MSG_MAP_EXP_CAPID     "%s: %llu: expected name, or '{' following %s: %s"
@ MSG_MAP_EXP_CAPHW     "%s: %llu: expected hardware capability, or \
                         terminator (';', '}'): %s"
@ MSG_MAP_EXP_CAPSF     "%s: %llu: expected software capability, or \
                         terminator (';', '}'): %s"
@ MSG_MAP_EXP_EQ        "%s: %llu: expected '=' following %s: %s"
@ MSG_MAP_EXP_EQ_ALL    "%s: %llu: expected '=', '+=', or '-=' following %s: %s"
@ MSG_MAP_EXP_EQ_PEQ    "%s: %llu: expected '=' following %s: %s"
@ MSG_MAP_EXP_DIR       "%s: %llu: expected mapfile directive (%s): %s"
@ MSG_MAP_SFLG_EXBANG   "%s: %llu: '!' appears without corresponding flag"
@ MSG_MAP_EXP_FILNAM    "%s: %llu: expected file name following %s: %s"
@ MSG_MAP_EXP_FILPATH   "%s: %llu: expected file path following %s: %s"
@ MSG_MAP_EXP_INT       "%s: %llu: expected integer value following %s: %s"
@ MSG_MAP_EXP_LBKT      "%s: %llu: expected '{' following %s: %s"
@ MSG_MAP_EXP_OBJNAM    "%s: %llu: expected object name following %s: %s"
@ MSG_MAP_SFLG_ONEBANG  "%s: %llu: '!' can only be specified once per flag"
@ MSG_MAP_EXP_SECFLAG   "%s: %llu: expected section flag (%s), '!', or \
                         terminator (';', '}'): %s"
@ MSG_MAP_EXP_SECNAM    "%s: %llu: expected section name following %s: %s"
@ MSG_MAP_EXP_SEGFLAG   "%s: %llu: expected segment flag (%s), or \
                         terminator (';', '}'): %s"
@ MSG_MAP_EXP_ECNAM     "%s: %llu: expected entrance criteria (ASSIGN_SECTION) \
                         name, or terminator (';', '}'): %s"
@ MSG_MAP_EXP_SEGNAM    "%s: %llu: expected segment name following %s: %s"
@ MSG_MAP_EXP_SEM       "%s: %llu: expected ';' to terminate %s: %s"
@ MSG_MAP_EXP_SEMLBKT   "%s: %llu: expected ';' or '{' following %s: %s"
@ MSG_MAP_EXP_SEMRBKT   "%s: %llu: expected ';' or '}' to terminate %s: %s"
@ MSG_MAP_EXP_SHTYPE    "%s: %llu: expected section type: %s"
@ MSG_MAP_EXP_SYM       "%s: %llu: expected symbol name, symbol scope, \
                         or '*': %s"
@ MSG_MAP_EXP_SYMEND    "%s: %llu: expected inherited version name, or \
                         terminator (';'): %s"
@ MSG_MAP_EXP_SYMDELIM  "%s: %llu: expected one of ':', ';', or '{': %s"
@ MSG_MAP_EXP_SYMFLAG   "%s: %llu: expected symbol flag (%s), or \
                         terminator (';', '}'): %s"
@ MSG_MAP_EXP_SYMNAM    "%s: %llu: expected symbol name following %s: %s"
@ MSG_MAP_EXP_SYMSCOPE  "%s: %llu: expected symbol scope (%s): %s"
@ MSG_MAP_EXP_SYMTYPE   "%s: %llu: expected symbol type (%s): %s"
@ MSG_MAP_EXP_VERSION   "%s: %llu: expected version name following %s: %s"
@ MSG_MAP_BADEXTRA      "%s: %llu: unexpected text found following %s directive"
@ MSG_MAP_VALUELIMIT    "%s: %llu: numeric value exceeds word size: %s"
@ MSG_MAP_MALVALUE      "%s: %llu: malformed numeric value: %s"
@ MSG_MAP_BADVALUETAIL  "%s: %llu: unexpected characters following numeric \
                         constant: %s"
@ MSG_MAP_WSNEEDED      "%s: %llu: whitespace needed before token: %s"
@ MSG_MAP_BADCHAR       "%s: %llu: unexpected text: %s"
@ MSG_MAP_BADKWQUOTE    "%s: %llu: mapfile keywords should not be quoted: %s"
@ MSG_MAP_CDIR_NOTBOL   "%s: %llu: mapfile control directive not at start of \
                         line: %s"
@ MSG_MAP_NOATTR        "%s: %llu: %s specified no attributes (empty {})"
@ MSG_MAP_NOVALUES      "%s: %llu: %s specified without values"
@ MSG_MAP_INTERR        "<internal error>"
@ MSG_MAP_ISORDVER      "%s: %llu: version 0 mapfile ?O flag and version 1 \
                         segment IS_ORDER attribute are mutually exclusive: %s"
@ MSG_MAP_SYMATTR       "symbol attributes";

# Mapfile Control Directives

@ MSG_MAP_CDIR_BADVDIR  "%s: %llu: $mapfile_version directive must specify \
```



```
                         version 2 or higher: %d"
@ MSG_MAP_CDIR_BADVER   "%s: %llu: unknown mapfile version: %d"
@ MSG_MAP_CDIR_REPVER   "%s: %llu: $mapfile_version must be first directive \
                         in file"
@ MSG_MAP_CDIR_REQARG   "%s: %llu: %s directive requires an argument"
@ MSG_MAP_CDIR_REQNOARG "%s: %llu: %s directive does not accept arguments"
@ MSG_MAP_CDIR_BAD      "%s: %llu: unrecognized mapfile control directive"
@ MSG_MAP_CDIR_NOIF     "%s: %llu: %s directive used without opening $if"
@ MSG_MAP_CDIR_ELSE     "%s: %llu: %s directive preceded by $else on line %d"
@ MSG_MAP_CDIR_NOEND    "%s: %llu: EOF encountered without closing $endif \
                         for $if on line %d"
@ MSG_MAP_CDIR_ERROR    "%s: %llu: error: %s"


# Mapfile Conditional Expressions

@ MSG_MAP_CEXP_TOKERR   "%s: %llu: syntax error in conditional expression at: %s
@ MSG_MAP_CEXP_SEMERR   "%s: %llu: malformed conditional expression"
@ MSG_MAP_CEXP_BADOPUSE "%s: %llu: invalid operator use in conditional \
                         expression"
@ MSG_MAP_CEXP_UNBALPAR "%s: %llu: unbalanced parenthesis in conditional \
                         expression"
@ MSG_MAP_BADCESC       "%s: %llu: unrecognized escape in double quoted \
                         token: \\%c\n"

# Generic error diagnostic labels

@ MSG_STR_NULL          "(null)"

@ MSG_DBG_DFLT_FMT      "debug: "
@ MSG_DBG_AOUT_FMT      "debug: a.out: "
@ MSG_DBG_NAME_FMT      "debug: %s: "

# -z assert-deflib strings

@ MSG_ARG_ASSDEFLIB_MALFORMED   "library name malformed: %s"
@ MSG_ARG_ASSDEFLIB_FOUND       "dynamic library found on default search path \
                                 (%s): lib%s.so"

@ _END_


# Software identification.  Note, the SGU strings is historic, and has
# little relevance.  It is preserved as applications have used this
# string to identify the Solaris link-editor.

@ MSG_SGS_ID            "ld: Software Generation Utilities - \
                         Solaris Link Editors: "

# The following strings represent reserved words, files, pathnames and symbols.
# Reference to this strings is via the MSG_ORIG() macro, and thus no message
# translation is required.

@ MSG_DBG_FOPEN_MODE    "w"

@ MSG_DBG_CLS32_FMT     "32: "
@ MSG_DBG_CLS64_FMT     "64: "

@ MSG_STR_PATHTOK       ":"
@ MSG_STR_AOUT          "a.out"

@ MSG_STR_LIB_A         "%s/lib%s.a"
@ MSG_STR_LIB_SO        "%s/lib%s.so"
@ MSG_STR_PATH          "%s/%s"
@ MSG_STR_STRNL         "%s\n"
@ MSG_STR_NL            "\n"
```

```
1314 @ MSG_STR_CAPGROUPID    "CAP_GROUP_%d"

1316 @ MSG_STR_LD_DYNAMIC    "dynamic"
1317 @ MSG_STR_SYMBOLIC      "symbolic"
1318 @ MSG_STR_ELIMINATE     "eliminate"
1319 @ MSG_STR_LOCAL         "local"
1320 @ MSG_STR_PROGBITS      "progbits"
1321 @ MSG_STR_SYMTAB        "symtab"
1322 @ MSG_STR_DYNSYM        "dynsym"
1323 @ MSG_STR_REL           "rel"
1324 @ MSG_STR_RELA          "rela"
1325 @ MSG_STR_STRTAB        "strtab"
1326 @ MSG_STR_HASH          "hash"
1327 @ MSG_STR_LIB           "lib"
1328 @ MSG_STR_NOTE          "note"
1329 @ MSG_STR_NOBITS        "nobits"
1330 @ MSG_STR_HWCAP_1       "hwcap_1"
1331 @ MSG_STR_SFCAP_1       "sfcap_1"
1332 @ MSG_STR_SOEXT         ".so"

1334 @ MSG_STR_OPTIONS       "3:6:abc:d:e:f:h:il:mo:p:rstu:z:B:CD:F:GI:L:M:N:P:Q:R:\
1335                          S:VW:Y:?"

1337 # Argument processing strings

1339 @ MSG_ARG_3             "-3"
1340 @ MSG_ARG_6             "-6"
1341 @ MSG_ARG_A             "-a"
1342 @ MSG_ARG_B             "-b"
1343 @ MSG_ARG_CB            "-B"
1344 @ MSG_ARG_BDIRECT       "-Bdirect"
1345 @ MSG_ARG_BDYNAMIC      "-Bdynamic"
1346 @ MSG_ARG_BELIMINATE    "-Beliminate"
1347 @ MSG_ARG_BGROUP        "-Bgroup"
1348 @ MSG_ARG_BLOCAL        "-Blocal"
1349 @ MSG_ARG_BNODIRECT     "-Bnodirect"
1350 @ MSG_ARG_BSYMBOLIC     "-Bsymbolic"
1351 @ MSG_ARG_BTRANSLATOR   "-Btranslator"
1352 @ MSG_ARG_C             "-c"
1353 @ MSG_ARG_D             "-d"
1354 @ MSG_ARG_DY            "-dy"
1355 @ MSG_ARG_CI            "-I"
1356 @ MSG_ARG_CN            "-N"
1357 @ MSG_ARG_P             "-p"
1358 @ MSG_ARG_CP            "-P"
1359 @ MSG_ARG_CQ            "-Q"
1360 @ MSG_ARG_CY            "-Y"
1361 @ MSG_ARG_CYL           "-YL"
1362 @ MSG_ARG_CYP           "-YP"
1363 @ MSG_ARG_CYU           "-YU"
1364 @ MSG_ARG_Z             "-z"
1365 @ MSG_ARG_ZDEFNODEF     "-z[defs|nodefs]"
1366 @ MSG_ARG_ZASLR         "-zaslr"
1367 @ MSG_ARG_ZGUIDE        "-zguidance"
1368 @ MSG_ARG_ZNODEF        "-znodefs"
1369 @ MSG_ARG_ZNOINTERP     "-znointerp"
1370 @ MSG_ARG_ZRELAXRELOC   "-zrelaxreloc"
1371 @ MSG_ARG_ZNORELAXRELOC "-znorelaxreloc"
1372 @ MSG_ARG_ZTEXT         "-ztext"
1373 @ MSG_ARG_ZTEXTOFF      "-ztextoff"
1374 @ MSG_ARG_ZTEXTWARN     "-ztextwarn"
1375 @ MSG_ARG_ZTEXTALL      "-z[text|textwarn|textoff]"
1376 @ MSG_ARG_ZLOADFLTR     "-zloadfltr"
1377 @ MSG_ARG_ZCOMBRELOC    "-zcombreloc"
1378 @ MSG_ARG_ZSYMBOLCAP    "-zsymbolcap"
1379 @ MSG_ARG_ZFATWNOFATW   "-z[fatal-warnings|nofatalwarnings]"
```



```
1381 @ MSG_ARG_ABSEXEC       "absexec"
1382 @ MSG_ARG_ALTEXEC64     "altexec64"
1383 @ MSG_ARG_ASLR          "aslr"
1384 @ MSG_ARG_NOCOMPSTRTAB  "nocompstrtab"
1385 @ MSG_ARG_GROUPPERM     "groupperm"
1386 @ MSG_ARG_NOGROUPPERM   "nogroupperm"
1387 @ MSG_ARG_LAZYLOAD      "lazyload"
1388 @ MSG_ARG_NOLAZYLOAD    "nolazyload"
1389 @ MSG_ARG_INTERPOSE     "interpose"
1390 @ MSG_ARG_DIRECT        "direct"
1391 @ MSG_ARG_NODIRECT      "nodirect"
1392 @ MSG_ARG_IGNORE        "ignore"
1393 @ MSG_ARG_RECORD        "record"
1394 @ MSG_ARG_INITFIRST     "initfirst"
1395 @ MSG_ARG_INITARRAY     "initarray="
1396 @ MSG_ARG_FINIARRAY     "finiarray="
1397 @ MSG_ARG_PREINITARRAY  "preinitarray="
1398 @ MSG_ARG_RTLDINFO      "rtldinfo="
1399 @ MSG_ARG_DTRACE        "dtrace="
1400 @ MSG_ARG_TRANSLATOR    "translator"
1401 @ MSG_ARG_NOOPEN        "nodlopen"
1402 @ MSG_ARG_NOW           "now"
1403 @ MSG_ARG_ORIGIN        "origin"
1404 @ MSG_ARG_DEFS          "defs"
1405 @ MSG_ARG_NODEFS        "nodefs"
1406 @ MSG_ARG_NODUMP        "nodump"
1407 @ MSG_ARG_NOVERSION     "noversion"
1408 @ MSG_ARG_TEXT          "text"
1409 @ MSG_ARG_TEXTOFF       "textoff"
1410 @ MSG_ARG_TEXTWARN      "textwarn"
1411 @ MSG_ARG_MULDEFS       "muldefs"
1412 @ MSG_ARG_NODELETE      "nodelete"
1413 @ MSG_ARG_NOINTERP      "nointerp"
1414 @ MSG_ARG_NOPARTIAL     "nopartial"
1415 @ MSG_ARG_NORELOC       "noreloc"
1416 @ MSG_ARG_REDLOCSYM     "redlocsym"
1417 @ MSG_ARG_VERBOSE       "verbose"
1418 @ MSG_ARG_WEAKEXT       "weakextract"
1419 @ MSG_ARG_LOADFLTR      "loadfltr"
1420 @ MSG_ARG_ALLEXTRT      "allextract"
1421 @ MSG_ARG_DFLEXTRT      "defaultextract"
1422 @ MSG_ARG_COMBRELOC     "combreloc"
1423 @ MSG_ARG_NOCOMBRELOC   "nocombreloc"
1424 @ MSG_ARG_NODEFAULTLIB  "nodefaultlib"
1425 @ MSG_ARG_ENDFILTEE     "endfiltee"
1426 @ MSG_ARG_LD32          "ld32="
1427 @ MSG_ARG_LD64          "ld64="
1428 @ MSG_ARG_RESCAN        "rescan"
1429 @ MSG_ARG_RESCAN_NOW    "rescan-now"
1430 @ MSG_ARG_RESCAN_START  "rescan-start"
1431 @ MSG_ARG_RESCAN_END    "rescan-end"
1432 @ MSG_ARG_GUIDE         "guidance"
1433 @ MSG_ARG_NOLDYNSYM     "noldynsym"
1434 @ MSG_ARG_RELAXRELOC    "relaxreloc"
1435 @ MSG_ARG_NORELAXRELOC  "norelaxreloc"
1436 @ MSG_ARG_NOSIGHANDLER  "nosighandler"
1437 @ MSG_ARG_GLOBAUDIT     "globalaudit"
1438 @ MSG_ARG_TARGET        "target="
1439 @ MSG_ARG_WRAP          "wrap="
1440 @ MSG_ARG_FATWARN       "fatal-warnings"
1441 @ MSG_ARG_NOFATWARN     "nofatal-warnings"
1442 @ MSG_ARG_HELP          "help"
1443 @ MSG_ARG_GROUP         "group"
1444 @ MSG_ARG_REDUCE        "reduce"
1445 @ MSG_ARG_STATIC        "static"
```

```
1446 @ MSG_ARG_SYMBOLCAP     "symbolcap"
1447 @ MSG_ARG_DEFERRED      "deferred"
1448 @ MSG_ARG_NODEFERRED    "nodeferred"
1449 @ MSG_ARG_ASSDEFLIB     "assert-deflib"

1451 @ MSG_ARG_LCOM          "L,"
1452 @ MSG_ARG_PCOM          "P,"
1453 @ MSG_ARG_UCOM          "U,"

1455 @ MSG_ARG_T_RPATH       "rpath"
1456 @ MSG_ARG_T_SHARED      "shared"
1457 @ MSG_ARG_T_SONAME      "soname"
1458 @ MSG_ARG_T_WL          "l,-"

1460 @ MSG_ARG_T_AUXFLTR     "-auxiliary"
1461 @ MSG_ARG_T_MULDEFS     "-allow-multiple-definition"
1462 @ MSG_ARG_T_INTERP      "-dynamic-linker"
1463 @ MSG_ARG_T_ENDGROUP    "-end-group"
1464 @ MSG_ARG_T_ENTRY       "-entry"
1465 @ MSG_ARG_T_STDFLTR     "-filter"
1466 @ MSG_ARG_T_FATWARN     "-fatal-warnings"
1467 @ MSG_ARG_T_NOFATWARN   "-no-fatal-warnings"
1468 @ MSG_ARG_T_HELP        "-help"
1469 @ MSG_ARG_T_LIBRARY     "-library"
1470 @ MSG_ARG_T_LIBPATH     "-library-path"
1471 @ MSG_ARG_T_NOUNDEF     "-no-undefined"
1472 @ MSG_ARG_T_NOWHOLEARC  "-no-whole-archive"
1473 @ MSG_ARG_T_OUTPUT      "-output"
1474 @ MSG_ARG_T_RELOCATABLE "-relocatable"
1475 @ MSG_ARG_T_STARTGROUP  "-start-group"
1476 @ MSG_ARG_T_STRIP       "-strip-all"
1477 @ MSG_ARG_T_UNDEF       "-undefined"
1478 @ MSG_ARG_T_VERSION     "-version"
1479 @ MSG_ARG_T_WHOLEARC    "-whole-archive"
1480 @ MSG_ARG_T_WRAP        "-wrap"
1481 @ MSG_ARG_T_OPAR        "("
1482 @ MSG_ARG_T_CPAR        ")"

1484 @ MSG_ARG_ENABLED       "enabled"
1485 @ MSG_ARG_DISABLED      "disabled"
1486 @ MSG_ARG_ENABLE        "enable"
1487 @ MSG_ARG_DISABLE       "disable"

1489 # -z guidance=item strings
1490 @ MSG_ARG_GUIDE_DELIM          ",: \t"
1491 @ MSG_ARG_GUIDE_NO_ALL         "noall"
1492 @ MSG_ARG_GUIDE_NO_DEFS        "nodefs"
1493 @ MSG_ARG_GUIDE_NO_DIRECT      "nodirect"
1494 @ MSG_ARG_GUIDE_NO_LAZYLOAD    "nolazyload"
1495 @ MSG_ARG_GUIDE_NO_MAPFILE     "nomapfile"
1496 @ MSG_ARG_GUIDE_NO_TEXT        "notext"
1497 @ MSG_ARG_GUIDE_NO_UNUSED      "nounused"

1499 # Environment variable strings

1501 @ MSG_LD_RUN_PATH       "LD_RUN_PATH"
1502 @ MSG_LD_LIBPATH_32     "LD_LIBRARY_PATH_32"
1503 @ MSG_LD_LIBPATH_64     "LD_LIBRARY_PATH_64"
1504 @ MSG_LD_LIBPATH        "LD_LIBRARY_PATH"

1506 @ MSG_LD_NOVERSION_32   "LD_NOVERSION_32"
1507 @ MSG_LD_NOVERSION_64   "LD_NOVERSION_64"
1508 @ MSG_LD_NOVERSION      "LD_NOVERSION"

1510 @ MSG_SGS_SUPPORT_32    "SGS_SUPPORT_32"
1511 @ MSG_SGS_SUPPORT_64    "SGS_SUPPORT_64"
```



```
1512 @ MSG_SGS_SUPPORT       "SGS_SUPPORT"


1515 # Symbol names

1517 @ MSG_SYM_LIBVER_U      "_lib_version"


1520 # Mapfile tokens

1522 @ MSG_MAP_LOAD          "load"
1523 @ MSG_MAP_NOTE          "note"
1524 @ MSG_MAP_NULL          "null"
1525 @ MSG_MAP_STACK         "stack"
1526 @ MSG_MAP_ADDVERS       "addvers"
1527 @ MSG_MAP_FUNCTION      "function"
1528 @ MSG_MAP_DATA          "data"
1529 @ MSG_MAP_COMMON        "common"
1530 @ MSG_MAP_PARENT        "parent"
1531 @ MSG_MAP_EXTERN        "extern"
1532 @ MSG_MAP_DIRECT        "direct"
1533 @ MSG_MAP_NODIRECT      "nodirect"
1534 @ MSG_MAP_FILTER        "filter"
1535 @ MSG_MAP_AUXILIARY     "auxiliary"
1536 @ MSG_MAP_OVERRIDE      "override"
1537 @ MSG_MAP_INTERPOSE     "interpose"
1538 @ MSG_MAP_DYNSORT       "dynsort"
1539 @ MSG_MAP_NODYNSORT     "nodynsort"

1541 @ MSG_MAPKW_ALIGN               "ALIGN"
1542 @ MSG_MAPKW_ALLOC               "ALLOC"
1543 @ MSG_MAPKW_ALLOW               "ALLOW"
1544 @ MSG_MAPKW_AMD64_LARGE         "AMD64_LARGE"
1545 @ MSG_MAPKW_ASSIGN_SECTION      "ASSIGN_SECTION"
1546 @ MSG_MAPKW_AUX                 "AUXILIARY"
1547 @ MSG_MAPKW_CAPABILITY          "CAPABILITY"
1548 @ MSG_MAPKW_COMMON              "COMMON"
1549 @ MSG_MAPKW_DATA                "DATA"
1550 @ MSG_MAPKW_DEFAULT             "DEFAULT"
1551 @ MSG_MAPKW_DEPEND_VERSIONS     "DEPEND_VERSIONS"
1552 @ MSG_MAPKW_DIRECT              "DIRECT"
1553 @ MSG_MAPKW_DISABLE             "DISABLE"
1554 @ MSG_MAPKW_DYNSORT             "DYNSORT"
1555 @ MSG_MAPKW_ELIMINATE           "ELIMINATE"
1556 @ MSG_MAPKW_EXECUTE             "EXECUTE"
1557 @ MSG_MAPKW_EXPORTED            "EXPORTED"
1558 @ MSG_MAPKW_EXTERN              "EXTERN"
1559 @ MSG_MAPKW_FILTER              "FILTER"
1560 @ MSG_MAPKW_FILE_BASENAME       "FILE_BASENAME"
1561 @ MSG_MAPKW_FILE_PATH           "FILE_PATH"
1562 @ MSG_MAPKW_FILE_OBJNAME        "FILE_OBJNAME"
1563 @ MSG_MAPKW_FUNCTION            "FUNCTION"
1564 @ MSG_MAPKW_FLAGS               "FLAGS"
1565 @ MSG_MAPKW_GLOBAL              "GLOBAL"
1566 @ MSG_MAPKW_INTERPOSE           "INTERPOSE"
1567 @ MSG_MAPKW_HIDDEN              "HIDDEN"
1568 @ MSG_MAPKW_HDR_NOALLOC         "HDR_NOALLOC"
1569 @ MSG_MAPKW_HW                  "HW"
1570 @ MSG_MAPKW_HW_1                "HW_1"
1571 @ MSG_MAPKW_HW_2                "HW_2"
1572 @ MSG_MAPKW_IS_NAME             "IS_NAME"
1573 @ MSG_MAPKW_IS_ORDER            "IS_ORDER"
1574 @ MSG_MAPKW_LOAD_SEGMENT        "LOAD_SEGMENT"
1575 @ MSG_MAPKW_LOCAL               "LOCAL"
1576 @ MSG_MAPKW_MACHINE             "MACHINE"
1577 @ MSG_MAPKW_MAX_SIZE            "MAX_SIZE"
```

```
1578 @ MSG_MAPKW_NOHDR               "NOHDR"
1579 @ MSG_MAPKW_NODIRECT            "NODIRECT"
1580 @ MSG_MAPKW_NODYNSORT           "NODYNSORT"
1581 @ MSG_MAPKW_NOTE_SEGMENT        "NOTE_SEGMENT"
1582 @ MSG_MAPKW_NULL_SEGMENT        "NULL_SEGMENT"
1583 @ MSG_MAPKW_OS_ORDER            "OS_ORDER"
1584 @ MSG_MAPKW_PADDR               "PADDR"
1585 @ MSG_MAPKW_PARENT              "PARENT"
1586 @ MSG_MAPKW_PHDR_ADD_NULL       "PHDR_ADD_NULL"
1587 @ MSG_MAPKW_PLATFORM            "PLATFORM"
1588 @ MSG_MAPKW_PROTECTED           "PROTECTED"
1589 @ MSG_MAPKW_READ                "READ"
1590 @ MSG_MAPKW_ROUND               "ROUND"
1591 @ MSG_MAPKW_REQUIRE             "REQUIRE"
1592 @ MSG_MAPKW_SEGMENT_ORDER       "SEGMENT_ORDER"
1593 @ MSG_MAPKW_SF                  "SF"
1594 @ MSG_MAPKW_SF_1                "SF_1"
1595 @ MSG_MAPKW_SINGLETON           "SINGLETON"
1596 @ MSG_MAPKW_SIZE                "SIZE"
1597 @ MSG_MAPKW_SIZE_SYMBOL         "SIZE_SYMBOL"
1598 @ MSG_MAPKW_STACK               "STACK"
1599 @ MSG_MAPKW_SYMBOL_SCOPE        "SYMBOL_SCOPE"
1600 @ MSG_MAPKW_SYMBOL_VERSION      "SYMBOL_VERSION"
1601 @ MSG_MAPKW_SYMBOLIC            "SYMBOLIC"
1602 @ MSG_MAPKW_TYPE                "TYPE"
1603 @ MSG_MAPKW_VADDR               "VADDR"
1604 @ MSG_MAPKW_VALUE               "VALUE"
1605 @ MSG_MAPKW_WRITE               "WRITE"


1608 @ MSG_STR_DTRACE        "PT_SUNWDTRACE"
```

```
**********************************************************
   97274 Mon Feb 11 00:23:19 2019
new/usr/src/cmd/sgs/libld/common/syms.c
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

```
/*
 * 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) 1988 AT&T
 *        All Rights Reserved
 *
 *
 * Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Symbol table management routines
 */

#define ELF_TARGET_AMD64

#include        <stdio.h>
#include        <string.h>
#include        <debug.h>
#include        <alloca.h>
#endif /* ! codereview */
#include        "msg.h"
#include        "_libld.h"

/*
 * AVL tree comparator function:
 *
 * The primary key is the symbol name hash with a secondary key of the symbol
 * name itself.
 */
int
ld_sym_avl_comp(const void *elem1, const void *elem2)
{
        Sym_avlnode     *sav1 = (Sym_avlnode *)elem1;
        Sym_avlnode     *sav2 = (Sym_avlnode *)elem2;
        int             res;

        res = sav1->sav_hash - sav2->sav_hash;

        if (res < 0)
```



```
                return (-1);
        if (res > 0)
                return (1);

        /*
         * Hash is equal - now compare name
         */
        res = strcmp(sav1->sav_name, sav2->sav_name);
        if (res == 0)
                return (0);
        if (res > 0)
                return (1);
        return (-1);
}

/*
 * Focal point for verifying symbol names.
 */
inline static const char *
string(Ofl_desc *ofl, Ifl_desc *ifl, Sym *sym, const char *strs, size_t strsize,
    int symndx, Word shndx, Word symsecndx, const char *symsecname,
    const char *strsecname, sd_flag_t *flags)
{
        Word    name = sym->st_name;

        if (name) {
                if ((ifl->ifl_flags & FLG_IF_HSTRTAB) == 0) {
                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_FIL_NOSTRTABLE),
                            ifl->ifl_name, EC_WORD(symsecndx), symsecname,
                            symndx, EC_XWORD(name));
                        return (NULL);
                }
                if (name >= (Word)strsize) {
                        ld_eprintf(ofl, ERR_FATAL,
                            MSG_INTL(MSG_FIL_EXCSTRTABLE), ifl->ifl_name,
                            EC_WORD(symsecndx), symsecname, symndx,
                            EC_XWORD(name), strsecname, EC_XWORD(strsize));
                        return (NULL);
                }
        }

        /*
         * Determine if we're dealing with a register and if so validate it.
         * If it's a scratch register, a fabricated name will be returned.
         */
        if (ld_targ.t_ms.ms_is_regsym != NULL) {
                const char *regname = (*ld_targ.t_ms.ms_is_regsym)(ofl, ifl,
                    sym, strs, symndx, shndx, symsecname, flags);

                if (regname == (const char *)S_ERROR) {
                        return (NULL);
                }
                if (regname)
                        return (regname);
        }

        /*
         * If this isn't a register, but we have a global symbol with a null
         * name, we're not going to be able to hash this, search for it, or
         * do anything interesting.  However, we've been accepting a symbol of
         * this kind for ages now, so give the user a warning (rather than a
         * fatal error), just in case this instance exists somewhere in the
         * world and hasn't, as yet, been a problem.
         */
        if ((name == 0) && (ELF_ST_BIND(sym->st_info) != STB_LOCAL)) {
                ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_FIL_NONAMESYM),
```

```
126                     ifl->ifl_name, EC_WORD(symsecndx), symsecname, symndx,
127                     EC_XWORD(name));
128             }
129             return (strs + name);
130     }
131
132     /*
133      * For producing symbol names strings to use in error messages.
134      * If the symbol has a non-null name, then the string returned by
135      * this function is the output from demangle(), surrounded by
136      * single quotes. For null names, a descriptive string giving
137      * the symbol section and index is generated.
138      *
139      * This function uses an internal static buffer to hold the resulting
140      * string. The value returned is usable by the caller until the next
141      * call, at which point it is overwritten.
142      */
143     static const char *
144     demangle_symname(const char *name, const char *symtab_name, Word symndx)
145     {
146     #define INIT_BUFSIZE 256
147
148             static char     *buf;
149             static size_t   bufsize = 0;
150             size_t          len;
151             int             use_name;
152
153             use_name = (name != NULL) && (*name != '\0');
154
155             if (use_name) {
156                     name = demangle(name);
157                     len = strlen(name) + 2;   /* Include room for quotes */
158             } else {
159                     name = MSG_ORIG(MSG_STR_EMPTY);
160                     len = strlen(symtab_name) + 2 + CONV_INV_BUFSIZE;
161             }
162             len++;                  /* Null termination */
163
164             /* If our buffer is too small, double it until it is big enough */
165             if (len > bufsize) {
166                     size_t  new_bufsize = bufsize;
167                     char    *new_buf;
168
169                     if (new_bufsize == 0)
170                             new_bufsize = INIT_BUFSIZE;
171                     while (len > new_bufsize)
172                             new_bufsize *= 2;
173                     if ((new_buf = libld_malloc(new_bufsize)) == NULL)
174                             return (name);
175                     buf = new_buf;
176                     bufsize = new_bufsize;
177             }
178
179             if (use_name) {
180                     (void) snprintf(buf, bufsize, MSG_ORIG(MSG_FMT_SYMNAM), name);
181             } else {
182                     (void) snprintf(buf, bufsize, MSG_ORIG(MSG_FMT_NULLSYMNAM),
183                         symtab_name, EC_WORD(symndx));
184             }
185
186             return (buf);
187
188     #undef INIT_BUFSIZE
189     }
190
191     /*
```



```
192      * Shared objects can be built that define specific symbols that can not be
193      * directly bound to.  These objects have a syminfo section (and an associated
194      * DF_1_NODIRECT dynamic flags entry).  Scan this table looking for symbols
195      * that can't be bound to directly, and if this files symbol is presently
196      * referenced, mark it so that we don't directly bind to it.
197      */
198     uintptr_t
199     ld_sym_nodirect(Is_desc *isp, Ifl_desc *ifl, Ofl_desc *ofl)
200     {
201             Shdr            *sifshdr, *symshdr;
202             Syminfo         *sifdata;
203             Sym             *symdata;
204             char            *strdata;
205             ulong_t         cnt, _cnt;
206
207             /*
208              * Get the syminfo data, and determine the number of entries.
209              */
210             sifshdr = isp->is_shdr;
211             sifdata = (Syminfo *)isp->is_indata->d_buf;
212             cnt =  sifshdr->sh_size / sifshdr->sh_entsize;
213
214             /*
215              * Get the associated symbol table.
216              */
217             if ((sifshdr->sh_link == 0) || (sifshdr->sh_link >= ifl->ifl_shnum)) {
218                     /*
219                      * Broken input file
220                      */
221                     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_FIL_INVSHINFO),
222                         ifl->ifl_name, isp->is_name, EC_XWORD(sifshdr->sh_link));
223                     return (0);
224             }
225             symshdr = ifl->ifl_isdesc[sifshdr->sh_link]->is_shdr;
226             symdata = ifl->ifl_isdesc[sifshdr->sh_link]->is_indata->d_buf;
227
228             /*
229              * Get the string table associated with the symbol table.
230              */
231             strdata = ifl->ifl_isdesc[symshdr->sh_link]->is_indata->d_buf;
232
233             /*
234              * Traverse the syminfo data for symbols that can't be directly
235              * bound to.
236              */
237             for (_cnt = 1, sifdata++; _cnt < cnt; _cnt++, sifdata++) {
238                     Sym             *sym;
239                     char            *str;
240                     Sym_desc        *sdp;
241
242                     if ((sifdata->si_flags & SYMINFO_FLG_NOEXTDIRECT) == 0)
243                             continue;
244
245                     sym = (Sym *)(symdata + _cnt);
246                     str = (char *)(strdata + sym->st_name);
247
248                     if ((sdp = ld_sym_find(str, SYM_NOHASH, NULL, ofl)) != NULL) {
249                             if (ifl != sdp->sd_file)
250                                     continue;
251
252                             sdp->sd_flags &= ~FLG_SY_DIR;
253                             sdp->sd_flags |= FLG_SY_NDIR;
254                     }
255             }
256             return (0);
257     }
```

```
/*
 * If, during symbol processing, it is necessary to update a local symbols
 * contents before we have generated the symbol tables in the output image,
 * create a new symbol structure and copy the original symbol contents.  While
 * we are processing the input files, their local symbols are part of the
 * read-only mapped image.  Commonly, these symbols are copied to the new output
 * file image and then updated to reflect their new address and any change in
 * attributes.  However, sometimes during relocation counting, it is necessary
 * to adjust the symbols information.  This routine provides for the generation
 * of a new symbol image so that this update can be performed.
 * All global symbols are copied to an internal symbol table to improve locality
 * of reference and hence performance, and thus this copying is not necessary.
 */
uintptr_t
ld_sym_copy(Sym_desc *sdp)
{
        Sym     *nsym;

        if (sdp->sd_flags & FLG_SY_CLEAN) {
                if ((nsym = libld_malloc(sizeof (Sym))) == NULL)
                        return (S_ERROR);
                *nsym = *(sdp->sd_sym);
                sdp->sd_sym = nsym;
                sdp->sd_flags &= ~FLG_SY_CLEAN;
        }
        return (1);
}

/*
 * Finds a given name in the link editors internal symbol table.  If no
 * hash value is specified it is calculated.  A pointer to the located
 * Sym_desc entry is returned, or NULL if the symbol is not found.
 */
Sym_desc *
ld_sym_find(const char *name, Word hash, avl_index_t *where, Ofl_desc *ofl)
{
        Sym_avlnode     qsav, *sav;

        if (hash == SYM_NOHASH)
                /* LINTED */
                hash = (Word)elf_hash((const char *)name);
        qsav.sav_hash = hash;
        qsav.sav_name = name;

        /*
         * Perform search for symbol in AVL tree.  Note that the 'where' field
         * is passed in from the caller.  If a 'where' is present, it can be
         * used in subsequent 'ld_sym_enter()' calls if required.
         */
        sav = avl_find(&ofl->ofl_symavl, &qsav, where);

        /*
         * If symbol was not found in the avl tree, return null to show that.
         */
        if (sav == NULL)
                return (NULL);

        /*
         * Return symbol found.
         */
        return (sav->sav_sdp);
}

/*
 * Enter a new symbol into the link editors internal symbol table.
```



```
 * If the symbol is from an input file, information regarding the input file
 * and input section is also recorded.  Otherwise (file == NULL) the symbol
 * has been internally generated (ie. _etext, _edata, etc.).
 */
Sym_desc *
ld_sym_enter(const char *name, Sym *osym, Word hash, Ifl_desc *ifl,
    Ofl_desc *ofl, Word ndx, Word shndx, sd_flag_t sdflags, avl_index_t *where)
{
        Sym_desc        *sdp;
        Sym_aux         *sap;
        Sym_avlnode     *savl;
        char            *_name;
        Sym             *nsym;
        Half            etype;
        uchar_t         vis;
        avl_index_t     _where;

        /*
         * Establish the file type.
         */
        if (ifl)
                etype = ifl->ifl_ehdr->e_type;
        else
                etype = ET_NONE;

        ofl->ofl_entercnt++;

        /*
         * Allocate a Sym Descriptor, Auxiliary Descriptor, and a Sym AVLNode -
         * contiguously.
         */
        if ((savl = libld_calloc(S_DROUND(sizeof (Sym_avlnode)) +
            S_DROUND(sizeof (Sym_desc)) +
            S_DROUND(sizeof (Sym_aux)), 1)) == NULL)
                return ((Sym_desc *)S_ERROR);
        sdp = (Sym_desc *)((uintptr_t)savl +
            S_DROUND(sizeof (Sym_avlnode)));
        sap = (Sym_aux *)((uintptr_t)sdp +
            S_DROUND(sizeof (Sym_desc)));

        savl->sav_sdp = sdp;
        sdp->sd_file = ifl;
        sdp->sd_aux = sap;
        savl->sav_hash = sap->sa_hash = hash;

        /*
         * Copy the symbol table entry from the input file into the internal
         * entry and have the symbol descriptor use it.
         */
        sdp->sd_sym = nsym = &sap->sa_sym;
        *nsym = *osym;
        sdp->sd_shndx = shndx;
        sdp->sd_flags |= sdflags;

        if ((_name = libld_malloc(strlen(name) + 1)) == NULL)
                return ((Sym_desc *)S_ERROR);
        savl->sav_name = sdp->sd_name = (const char *)strcpy(_name, name);

        /*
         * Enter Symbol in AVL tree.
         */
        if (where == 0) {
                /* LINTED */
                Sym_avlnode     *_savl;
                /*
                 * If a previous ld_sym_find() hasn't initialized 'where' do it
```

```
390                  * now.
391                  */
392                 where = &_where;
393                 _savl = avl_find(&ofl->ofl_symavl, savl, where);
394                 assert(_savl == NULL);
395         }
396         avl_insert(&ofl->ofl_symavl, savl, *where);
397
398         /*
399          * Record the section index.  This is possible because the
400          * 'ifl_isdesc' table is filled before we start symbol processing.
401          */
402         if ((sdflags & FLG_SY_SPECSEC) || (nsym->st_shndx == SHN_UNDEF))
403                 sdp->sd_isc = NULL;
404         else {
405                 sdp->sd_isc = ifl->ifl_isdesc[shndx];
406
407                 /*
408                  * If this symbol is from a relocatable object, make sure that
409                  * it is still associated with a section.  For example, an
410                  * unknown section type (SHT_NULL) would have been rejected on
411                  * input with a warning.  Here, we make the use of the symbol
412                  * fatal.  A symbol descriptor is still returned, so that the
413                  * caller can continue processing all symbols, and hence flush
414                  * out as many error conditions as possible.
415                  */
416                 if ((etype == ET_REL) && (sdp->sd_isc == NULL)) {
417                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_SYM_INVSEC),
418                             name, ifl->ifl_name, EC_XWORD(shndx));
419                         return (sdp);
420                 }
421         }
422
423         /*
424          * Mark any COMMON symbols as 'tentative'.
425          */
426         if (sdflags & FLG_SY_SPECSEC) {
427                 if (nsym->st_shndx == SHN_COMMON)
428                         sdp->sd_flags |= FLG_SY_TENTSYM;
429 #if     defined(_ELF64)
430                 else if ((ld_targ.t_m.m_mach == EM_AMD64) &&
431                     (nsym->st_shndx == SHN_X86_64_LCOMMON))
432                         sdp->sd_flags |= FLG_SY_TENTSYM;
433 #endif
434         }
435
436         /*
437          * Establish the symbols visibility and reference.
438          */
439         vis = ELF_ST_VISIBILITY(nsym->st_other);
440
441         if ((etype == ET_NONE) || (etype == ET_REL)) {
442                 switch (vis) {
443                 case STV_DEFAULT:
444                         sdp->sd_flags |= FLG_SY_DEFAULT;
445                         break;
446                 case STV_INTERNAL:
447                 case STV_HIDDEN:
448                         sdp->sd_flags |= FLG_SY_HIDDEN;
449                         break;
450                 case STV_PROTECTED:
451                         sdp->sd_flags |= FLG_SY_PROTECT;
452                         break;
453                 case STV_EXPORTED:
454                         sdp->sd_flags |= FLG_SY_EXPORT;
455                         break;
```



```
456                 case STV_SINGLETON:
457                         sdp->sd_flags |= (FLG_SY_SINGLE | FLG_SY_NDIR);
458                         ofl->ofl_flags1 |= (FLG_OF1_NDIRECT | FLG_OF1_NGLBDIR);
459                         break;
460                 case STV_ELIMINATE:
461                         sdp->sd_flags |= (FLG_SY_HIDDEN | FLG_SY_ELIM);
462                         break;
463                 default:
464                         assert(vis <= STV_ELIMINATE);
465                 }
466
467                 sdp->sd_ref = REF_REL_NEED;
468
469                 /*
470                  * Under -Bnodirect, all exported interfaces that have not
471                  * explicitly been defined protected or directly bound to, are
472                  * tagged to prevent direct binding.
473                  */
474                 if ((ofl->ofl_flags1 & FLG_OF1_ALNODIR) &&
475                     ((sdp->sd_flags & (FLG_SY_PROTECT | FLG_SY_DIR)) == 0) &&
476                     (nsym->st_shndx != SHN_UNDEF)) {
477                         sdp->sd_flags |= FLG_SY_NDIR;
478                 }
479         } else {
480                 sdp->sd_ref = REF_DYN_SEEN;
481
482                 /*
483                  * If this is a protected symbol, remember this.  Note, this
484                  * state is different from the FLG_SY_PROTECT used to establish
485                  * a symbol definitions visibility.  This state is used to warn
486                  * against possible copy relocations against this referenced
487                  * symbol.
488                  */
489                 if (vis == STV_PROTECTED)
490                         sdp->sd_flags |= FLG_SY_PROT;
491
492                 /*
493                  * If this is a SINGLETON definition, then indicate the symbol
494                  * can not be directly bound to, and retain the visibility.
495                  * This visibility will be inherited by any references made to
496                  * this symbol.
497                  */
498                 if ((vis == STV_SINGLETON) && (nsym->st_shndx != SHN_UNDEF))
499                         sdp->sd_flags |= (FLG_SY_SINGLE | FLG_SY_NDIR);
500
501                 /*
502                  * If the new symbol is from a shared library and is associated
503                  * with a SHT_NOBITS section then this symbol originated from a
504                  * tentative symbol.
505                  */
506                 if (sdp->sd_isc &&
507                     (sdp->sd_isc->is_shdr->sh_type == SHT_NOBITS))
508                         sdp->sd_flags |= FLG_SY_TENTSYM;
509         }
510
511         /*
512          * Reclassify any SHN_SUNW_IGNORE symbols to SHN_UNDEF so as to
513          * simplify future processing.
514          */
515         if (nsym->st_shndx == SHN_SUNW_IGNORE) {
516                 sdp->sd_shndx = shndx = SHN_UNDEF;
517                 sdp->sd_flags |= (FLG_SY_REDUCED |
518                     FLG_SY_HIDDEN | FLG_SY_IGNORE | FLG_SY_ELIM);
519         }
520
521         /*
```

```
522          * If this is an undefined, or common symbol from a relocatable object
523          * determine whether it is a global or weak reference (see build_osym(),
524          * where REF_DYN_NEED definitions are returned back to undefines).
525          */
526         if ((etype == ET_REL) &&
527             (ELF_ST_BIND(nsym->st_info) == STB_GLOBAL) &&
528             ((nsym->st_shndx == SHN_UNDEF) || ((sdflags & FLG_SY_SPECSEC) &&
529 #if     defined(_ELF64)
530             ((nsym->st_shndx == SHN_COMMON) ||
531             ((ld_targ.t_m.m_mach == EM_AMD64) &&
532             (nsym->st_shndx == SHN_X86_64_LCOMMON))))))
533 #else
534         /* BEGIN CSTYLED */
535             (nsym->st_shndx == SHN_COMMON))))
536         /* END CSTYLED */
537 #endif
538                 sdp->sd_flags |= FLG_SY_GLOBREF;

540         /*
541          * Record the input filename on the referenced or defined files list
542          * for possible later diagnostics.  The 'sa_rfile' pointer contains the
543          * name of the file that first referenced this symbol and is used to
544          * generate undefined symbol diagnostics (refer to sym_undef_entry()).
545          * Note that this entry can be overridden if a reference from a
546          * relocatable object is found after a reference from a shared object
547          * (refer to sym_override()).
548          * The 'sa_dfiles' list is used to maintain the list of files that
549          * define the same symbol.  This list can be used for two reasons:
550          *
551          *  -   To save the first definition of a symbol that is not available
552          *      for this link-edit.
553          *
554          *  -   To save all definitions of a symbol when the -m option is in
555          *      effect.  This is optional as it is used to list multiple
556          *      (interposed) definitions of a symbol (refer to ldmap_out()),
557          *      and can be quite expensive.
558          */
559         if (nsym->st_shndx == SHN_UNDEF) {
560                 sap->sa_rfile = ifl->ifl_name;
561         } else {
562                 if (sdp->sd_ref == REF_DYN_SEEN) {
563                         /*
564                          * A symbol is determined to be unavailable if it
565                          * belongs to a version of a shared object that this
566                          * user does not wish to use, or if it belongs to an
567                          * implicit shared object.
568                          */
569                         if (ifl->ifl_vercnt) {
570                                 Ver_index       *vip;
571                                 Half            vndx = ifl->ifl_versym[ndx];

573                                 sap->sa_dverndx = vndx;
574                                 vip = &ifl->ifl_verndx[vndx];
575                                 if (!(vip->vi_flags & FLG_VER_AVAIL)) {
576                                         sdp->sd_flags |= FLG_SY_NOTAVAIL;
577                                         sap->sa_vfile = ifl->ifl_name;
578                                 }
579                         }
580                         if (!(ifl->ifl_flags & FLG_IF_NEEDED))
581                                 sdp->sd_flags |= FLG_SY_NOTAVAIL;

583                 } else if (etype == ET_REL) {
584                         /*
585                          * If this symbol has been obtained from a versioned
586                          * input relocatable object then the new symbol must be
587                          * promoted to the versioning of the output file.
```



```
588                          */
589                         if (ifl->ifl_versym)
590                                 ld_vers_promote(sdp, ndx, ifl, ofl);
591                 }

593                 if ((ofl->ofl_flags & FLG_OF_GENMAP) &&
594                     ((sdflags & FLG_SY_SPECSEC) == 0))
595                         if (aplist_append(&sap->sa_dfiles, ifl->ifl_name,
596                             AL_CNT_SDP_DFILES) == NULL)
597                                 return ((Sym_desc *)S_ERROR);
598         }

600         /*
601          * Provided we're not processing a mapfile, diagnose the entered symbol.
602          * Mapfile processing requires the symbol to be updated with additional
603          * information, therefore the diagnosing of the symbol is deferred until
604          * later (see Dbg_map_symbol()).
605          */
606         if ((ifl == NULL) || ((ifl->ifl_flags & FLG_IF_MAPFILE) == 0))
607                 DBG_CALL(Dbg_syms_entered(ofl, nsym, sdp));

609         return (sdp);
610 }

612 /*
613  * Add a special symbol to the symbol table.  Takes special symbol name with
614  * and without underscores.  This routine is called, after all other symbol
615  * resolution has completed, to generate a reserved absolute symbol (the
616  * underscore version).  Special symbols are updated with the appropriate
617  * values in update_osym().  If the user has already defined this symbol
618  * issue a warning and leave the symbol as is.  If the non-underscore symbol
619  * is referenced then turn it into a weak alias of the underscored symbol.
620  *
621  * The bits in sdflags_u are OR'd into the flags field of the symbol for the
622  * underscored symbol.
623  *
624  * If this is a global symbol, and it hasn't explicitly been defined as being
625  * directly bound to, indicate that it can't be directly bound to.
626  * Historically, most special symbols only have meaning to the object in which
627  * they exist, however, they've always been global.  To ensure compatibility
628  * with any unexpected use presently in effect, ensure these symbols don't get
629  * directly bound to.  Note, that establishing this state here isn't sufficient
630  * to create a syminfo table, only if a syminfo table is being created by some
631  * other symbol directives will the nodirect binding be recorded.  This ensures
632  * we don't create syminfo sections for all objects we create, as this might add
633  * unnecessary bloat to users who haven't explicitly requested extra symbol
634  * information.
635  */
636 static uintptr_t
637 sym_add_spec(const char *name, const char *uname, Word sdaux_id,
638     sd_flag_t sdflags_u, sd_flag_t sdflags, Ofl_desc *ofl)
639 {
640         Sym_desc        *sdp;
641         Sym_desc        *usdp;
642         Sym             *sym;
643         Word            hash;
644         avl_index_t     where;

646         /* LINTED */
647         hash = (Word)elf_hash(uname);
648         if (usdp = ld_sym_find(uname, hash, &where, ofl)) {
649                 /*
650                  * If the underscore symbol exists and is undefined, or was
651                  * defined in a shared library, convert it to a local symbol.
652                  * Otherwise leave it as is and warn the user.
653                  */
```

```
654                 if ((usdp->sd_shndx == SHN_UNDEF) ||
655                     (usdp->sd_ref != REF_REL_NEED)) {
656                         usdp->sd_ref = REF_REL_NEED;
657                         usdp->sd_shndx = usdp->sd_sym->st_shndx = SHN_ABS;
658                         usdp->sd_flags |= FLG_SY_SPECSEC | sdflags_u;
659                         usdp->sd_sym->st_info =
660                             ELF_ST_INFO(STB_GLOBAL, STT_OBJECT);
661                         usdp->sd_isc = NULL;
662                         usdp->sd_sym->st_size = 0;
663                         usdp->sd_sym->st_value = 0;
664                         /* LINTED */
665                         usdp->sd_aux->sa_symspec = (Half)sdaux_id;

667                         /*
668                          * If a user hasn't specifically indicated that the
669                          * scope of this symbol be made local, then leave it
670                          * as global (ie. prevent automatic scoping).  The GOT
671                          * should be defined protected, whereas all other
672                          * special symbols are tagged as no-direct.
673                          */
674                         if (!SYM_IS_HIDDEN(usdp) &&
675                             (sdflags & FLG_SY_DEFAULT)) {
676                                 usdp->sd_aux->sa_overndx = VER_NDX_GLOBAL;
677                                 if (sdaux_id == SDAUX_ID_GOT) {
678                                         usdp->sd_flags &= ~FLG_SY_NDIR;
679                                         usdp->sd_flags |= FLG_SY_PROTECT;
680                                         usdp->sd_sym->st_other = STV_PROTECTED;
681                                 } else if (
682                                     ((usdp->sd_flags & FLG_SY_DIR) == 0) &&
683                                     ((ofl->ofl_flags & FLG_OF_SYMBOLIC) == 0)) {
684                                         usdp->sd_flags |= FLG_SY_NDIR;
685                                 }
686                         }
687                         usdp->sd_flags |= sdflags;

689                         /*
690                          * If the reference originated from a mapfile ensure
691                          * we mark the symbol as used.
692                          */
693                         if (usdp->sd_flags & FLG_SY_MAPREF)
694                                 usdp->sd_flags |= FLG_SY_MAPUSED;

696                         DBG_CALL(Dbg_syms_updated(ofl, usdp, uname));
697                 } else {
 39                 } else
698                         ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_RESERVE),
699                             uname, usdp->sd_file->ifl_name);
700                 }
701 #endif /* ! codereview */
702         } else {
703                 /*
704                  * If the symbol does not exist create it.
705                  */
706                 if ((sym = libld_calloc(sizeof (Sym), 1)) == NULL)
707                         return (S_ERROR);
708                 sym->st_shndx = SHN_ABS;
709                 sym->st_info = ELF_ST_INFO(STB_GLOBAL, STT_OBJECT);
710                 sym->st_size = 0;
711                 sym->st_value = 0;
712                 DBG_CALL(Dbg_syms_created(ofl->ofl_lml, uname));
713                 if ((usdp = ld_sym_enter(uname, sym, hash, (Ifl_desc *)NULL,
714                     ofl, 0, SHN_ABS, (FLG_SY_SPECSEC | sdflags_u), &where)) ==
715                     (Sym_desc *)S_ERROR)
716                         return (S_ERROR);
717                 usdp->sd_ref = REF_REL_NEED;
718                 /* LINTED */
```



```
719                 usdp->sd_aux->sa_symspec = (Half)sdaux_id;

721                 usdp->sd_aux->sa_overndx = VER_NDX_GLOBAL;

723                 if (sdaux_id == SDAUX_ID_GOT) {
724                         usdp->sd_flags |= FLG_SY_PROTECT;
725                         usdp->sd_sym->st_other = STV_PROTECTED;
726                 } else if ((sdflags & FLG_SY_DEFAULT) &&
727                     ((ofl->ofl_flags & FLG_OF_SYMBOLIC) == 0)) {
728                         usdp->sd_flags |= FLG_SY_NDIR;
729                 }
730                 usdp->sd_flags |= sdflags;
731         }

733         if (name && (sdp = ld_sym_find(name, SYM_NOHASH, NULL, ofl)) &&
734             (sdp->sd_sym->st_shndx == SHN_UNDEF)) {
735                 uchar_t bind;

737                 /*
738                  * If the non-underscore symbol exists and is undefined
739                  * convert it to be a local.  If the underscore has
740                  * sa_symspec set (ie. it was created above) then simulate this
741                  * as a weak alias.
742                  */
743                 sdp->sd_ref = REF_REL_NEED;
744                 sdp->sd_shndx = sdp->sd_sym->st_shndx = SHN_ABS;
745                 sdp->sd_flags |= FLG_SY_SPECSEC;
746                 sdp->sd_isc = NULL;
747                 sdp->sd_sym->st_size = 0;
748                 sdp->sd_sym->st_value = 0;
749                 /* LINTED */
750                 sdp->sd_aux->sa_symspec = (Half)sdaux_id;
751                 if (usdp->sd_aux->sa_symspec) {
752                         usdp->sd_aux->sa_linkndx = 0;
753                         sdp->sd_aux->sa_linkndx = 0;
754                         bind = STB_WEAK;
755                 } else
756                         bind = STB_GLOBAL;
757                 sdp->sd_sym->st_info = ELF_ST_INFO(bind, STT_OBJECT);

759                 /*
760                  * If a user hasn't specifically indicated the scope of this
761                  * symbol be made local then leave it as global (ie. prevent
762                  * automatic scoping).  The GOT should be defined protected,
763                  * whereas all other special symbols are tagged as no-direct.
764                  */
765                 if (!SYM_IS_HIDDEN(sdp) &&
766                     (sdflags & FLG_SY_DEFAULT)) {
767                         sdp->sd_aux->sa_overndx = VER_NDX_GLOBAL;
768                         if (sdaux_id == SDAUX_ID_GOT) {
769                                 sdp->sd_flags &= ~FLG_SY_NDIR;
770                                 sdp->sd_flags |= FLG_SY_PROTECT;
771                                 sdp->sd_sym->st_other = STV_PROTECTED;
772                         } else if (((sdp->sd_flags & FLG_SY_DIR) == 0) &&
773                             ((ofl->ofl_flags & FLG_OF_SYMBOLIC) == 0)) {
774                                 sdp->sd_flags |= FLG_SY_NDIR;
775                         }
776                 }
777                 sdp->sd_flags |= sdflags;

779                 /*
780                  * If the reference originated from a mapfile ensure
781                  * we mark the symbol as used.
782                  */
783                 if (sdp->sd_flags & FLG_SY_MAPREF)
784                         sdp->sd_flags |= FLG_SY_MAPUSED;
```

```
786                 DBG_CALL(Dbg_syms_updated(ofl, sdp, name));
787         }
788         return (1);
789 }


792 /*
793  * Undefined symbols can fall into one of four types:
794  *
795  *  -   the symbol is really undefined (SHN_UNDEF).
796  *
797  *  -   versioning has been enabled, however this symbol has not been assigned
798  *      to one of the defined versions.
799  *
800  *  -   the symbol has been defined by an implicitly supplied library, ie. one
801  *      which was encounted because it was NEEDED by another library, rather
802  *      than from a command line supplied library which would become the only
803  *      dependency of the output file being produced.
804  *
805  *  -   the symbol has been defined by a version of a shared object that is
806  *      not permitted for this link-edit.
807  *
808  * In all cases the file who made the first reference to this symbol will have
809  * been recorded via the 'sa_rfile' pointer.
810  */
811 typedef enum {
812         UNDEF,          NOVERSION,      IMPLICIT,       NOTAVAIL,
813         BNDLOCAL
814 } Type;

816 static const Msg format[] = {
817         MSG_SYM_UND_UNDEF,              /* MSG_INTL(MSG_SYM_UND_UNDEF) */
818         MSG_SYM_UND_NOVER,              /* MSG_INTL(MSG_SYM_UND_NOVER) */
819         MSG_SYM_UND_IMPL,               /* MSG_INTL(MSG_SYM_UND_IMPL) */
820         MSG_SYM_UND_NOTA,               /* MSG_INTL(MSG_SYM_UND_NOTA) */
821         MSG_SYM_UND_BNDLOCAL            /* MSG_INTL(MSG_SYM_UND_BNDLOCAL) */
822 };

824 /*
825  * Issue an undefined symbol message for the given symbol.
826  *
827  * entry:
828  *      ofl - Output descriptor
829  *      sdp - Undefined symbol to report
830  *      type - Type of undefined symbol
831  *      ofl_flag - One of 0, FLG_OF_FATAL, or FLG_OF_WARN.
832  *      undef_state - Address of variable to be initialized to 0
833  *              before the first call to sym_undef_entry, and passed
834  *              to each subsequent call. A non-zero value for *undef_state
835  *              indicates that this is not the first call in the series.
836  *
837  * exit:
838  *      If *undef_state is 0, a title is issued.
839  *
840  *      A message for the undefined symbol is issued.
841  *
842  *      If ofl_flag is non-zero, its value is OR'd into *undef_state. Otherwise,
843  *      all bits other than FLG_OF_FATAL and FLG_OF_WARN are set, in order to
844  *      provide *undef_state with a non-zero value. These other bits have
845  *      no meaning beyond that, and serve to ensure that *undef_state is
846  *      non-zero if sym_undef_entry() has been called.
847  */
848 static void
849 sym_undef_entry(Ofl_desc *ofl, Sym_desc *sdp, Type type, ofl_flag_t ofl_flag,
850     ofl_flag_t *undef_state)
```



```
851 {
852         const char      *name1, *name2, *name3;
853         Ifl_desc        *ifl = sdp->sd_file;
854         Sym_aux         *sap = sdp->sd_aux;

856         if (*undef_state == 0)
857                 ld_eprintf(ofl, ERR_NONE, MSG_INTL(MSG_SYM_FMT_UNDEF),
858                     MSG_INTL(MSG_SYM_UNDEF_ITM_11),
859                     MSG_INTL(MSG_SYM_UNDEF_ITM_21),
860                     MSG_INTL(MSG_SYM_UNDEF_ITM_12),
861                     MSG_INTL(MSG_SYM_UNDEF_ITM_22));

863         ofl->ofl_flags |= ofl_flag;
864         *undef_state |= ofl_flag ? ofl_flag : ~(FLG_OF_FATAL | FLG_OF_WARN);

866         switch (type) {
867         case UNDEF:
868         case BNDLOCAL:
869                 name1 = sap->sa_rfile;
870                 break;
871         case NOVERSION:
872                 name1 = ifl->ifl_name;
873                 break;
874         case IMPLICIT:
875                 name1 = sap->sa_rfile;
876                 name2 = ifl->ifl_name;
877                 break;
878         case NOTAVAIL:
879                 name1 = sap->sa_rfile;
880                 name2 = sap->sa_vfile;
881                 name3 = ifl->ifl_verndx[sap->sa_dverndx].vi_name;
882                 break;
883         default:
884                 return;
885         }

887         ld_eprintf(ofl, ERR_NONE, MSG_INTL(format[type]),
888             demangle(sdp->sd_name), name1, name2, name3);
889 }

891 /*
892  * If an undef symbol exists naming a bound for the output section,
893  * turn it into a defined symbol with the correct value.
894  *
895  * We set an arbitrary 1KB limit on the resulting symbol names.
896  */
897 static void
898 sym_add_bounds(Ofl_desc *ofl, Os_desc *osp, Word bound)
899 {
900         Sym_desc *bsdp;
901         char symn[1024];
902         size_t nsz;

904         switch (bound) {
905         case SDAUX_ID_SECBOUND_START:
906                 nsz = snprintf(symn, sizeof (symn), "%s%s",
907                     MSG_ORIG(MSG_SYM_SECBOUND_START), osp->os_name + 1);
908                 if (nsz > sizeof (symn))
909                         return;
910                 break;
911         case SDAUX_ID_SECBOUND_STOP:
912                 nsz = snprintf(symn, sizeof (symn), "%s%s",
913                     MSG_ORIG(MSG_SYM_SECBOUND_STOP), osp->os_name + 1);
914                 if (nsz > sizeof (symn))
915                         return;
916                 break;
```

```
917         default:
918                 assert(0);
919         }

921         if ((bsdp = ld_sym_find(symn, SYM_NOHASH, NULL, ofl)) != NULL) {
922                 if ((bsdp->sd_shndx != SHN_UNDEF) &&
923                     (bsdp->sd_ref == REF_REL_NEED)) {
924                         ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_RESERVE),
925                             symn, bsdp->sd_file->ifl_name);
926                         return;
927                 }

929                 DBG_CALL(Dbg_syms_updated(ofl, bsdp, symn));

931                 bsdp->sd_aux->sa_symspec = bound;
932                 bsdp->sd_aux->sa_boundsec = osp;
933                 bsdp->sd_flags |= FLG_SY_SPECSEC;
934                 bsdp->sd_ref = REF_REL_NEED;
935                 bsdp->sd_sym->st_info = ELF_ST_INFO(STB_GLOBAL, STT_NOTYPE);
936                 bsdp->sd_sym->st_other = STV_PROTECTED;
937                 bsdp->sd_isc = NULL;
938                 bsdp->sd_sym->st_size = 0;
939                 bsdp->sd_sym->st_value = 0;
940                 bsdp->sd_shndx = bsdp->sd_sym->st_shndx = SHN_ABS;
941         }
942 }

944 /*
945 #endif /* ! codereview */
946  * At this point all symbol input processing has been completed, therefore
947  * complete the symbol table entries by generating any necessary internal
948  * symbols.
949  */
950 uintptr_t
951 ld_sym_spec(Ofl_desc *ofl)
952 {
953         Sym_desc        *sdp;
954         Sg_desc         *sgp;
955         Aliste          idx1;
956 #endif /* ! codereview */

958         if (ofl->ofl_flags & FLG_OF_RELOBJ)
959                 return (1);

961         DBG_CALL(Dbg_syms_spec_title(ofl->ofl_lml));

963         /*
964          * For each section in the output file, look for symbols named for the
965          * __start/__stop patterns.  If references exist, flesh the symbols to
966          * be defined.
967          *
968          * the symbols are given values at the same time as the other special
969          * symbols.
970          */
971         for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
972                 Os_desc *osp;
973                 Aliste idx2;

975                 for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
976                         sym_add_bounds(ofl, osp, SDAUX_ID_SECBOUND_START);
977                         sym_add_bounds(ofl, osp, SDAUX_ID_SECBOUND_STOP);
978                 }
979         }

981 #endif /* ! codereview */
982         if (sym_add_spec(MSG_ORIG(MSG_SYM_ETEXT), MSG_ORIG(MSG_SYM_ETEXT_U),
```



```
983             SDAUX_ID_ETEXT, 0, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
984             ofl) == S_ERROR)
985                 return (S_ERROR);
986         if (sym_add_spec(MSG_ORIG(MSG_SYM_EDATA), MSG_ORIG(MSG_SYM_EDATA_U),
987             SDAUX_ID_EDATA, 0, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
988             ofl) == S_ERROR)
989                 return (S_ERROR);
990         if (sym_add_spec(MSG_ORIG(MSG_SYM_END), MSG_ORIG(MSG_SYM_END_U),
991             SDAUX_ID_END, FLG_SY_DYNSORT, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
992             ofl) == S_ERROR)
993                 return (S_ERROR);
994         if (sym_add_spec(MSG_ORIG(MSG_SYM_L_END), MSG_ORIG(MSG_SYM_L_END_U),
995             SDAUX_ID_END, 0, FLG_SY_HIDDEN, ofl) == S_ERROR)
996                 return (S_ERROR);
997         if (sym_add_spec(MSG_ORIG(MSG_SYM_L_START), MSG_ORIG(MSG_SYM_L_START_U),
998             SDAUX_ID_START, 0, FLG_SY_HIDDEN, ofl) == S_ERROR)
999                 return (S_ERROR);

1001        /*
1002         * Historically we've always produced a _DYNAMIC symbol, even for
1003         * static executables (in which case its value will be 0).
1004         */
1005        if (sym_add_spec(MSG_ORIG(MSG_SYM_DYNAMIC), MSG_ORIG(MSG_SYM_DYNAMIC_U),
1006            SDAUX_ID_DYN, FLG_SY_DYNSORT, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
1007            ofl) == S_ERROR)
1008                return (S_ERROR);

1010        if (OFL_ALLOW_DYNSYM(ofl))
1011                if (sym_add_spec(MSG_ORIG(MSG_SYM_PLKTBL),
1012                    MSG_ORIG(MSG_SYM_PLKTBL_U), SDAUX_ID_PLT,
1013                    FLG_SY_DYNSORT, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
1014                    ofl) == S_ERROR)
1015                        return (S_ERROR);

1017        /*
1018         * A GOT reference will be accompanied by the associated GOT symbol.
1019         * Make sure it gets assigned the appropriate special attributes.
1020         */
1021        if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_GOFTBL_U),
1022            SYM_NOHASH, NULL, ofl)) != NULL) && (sdp->sd_ref != REF_DYN_SEEN)) {
1023                if (sym_add_spec(MSG_ORIG(MSG_SYM_GOFTBL),
1024                    MSG_ORIG(MSG_SYM_GOFTBL_U), SDAUX_ID_GOT, FLG_SY_DYNSORT,
1025                    (FLG_SY_DEFAULT | FLG_SY_EXPDEF), ofl) == S_ERROR)
1026                        return (S_ERROR);
1027        }

1029        return (1);
1030 }

1032 /*
1033  * Determine a potential capability symbol's visibility.
1034  *
1035  * The -z symbolcap option transforms an object capabilities relocatable object
1036  * into a symbol capabilities relocatable object.  Any global function symbols,
1037  * or initialized global data symbols are candidates for transforming into local
1038  * symbol capabilities definitions.  However, if a user indicates that a symbol
1039  * should be demoted to local using a mapfile, then there is no need to
1040  * transform the associated global symbol.
1041  *
1042  * Normally, a symbol's visibility is determined after the symbol resolution
1043  * process, after all symbol state has been gathered and resolved.  However,
1044  * for -z symbolcap, this determination is too late.  When a global symbol is
1045  * read from an input file we need to determine it's visibility so as to decide
1046  * whether to create a local or not.
1047  *
1048  * If a user has explicitly defined this symbol as having local scope within a
```

```
 * mapfile, then a symbol of the same name already exists.  However, explicit
 * local definitions are uncommon, as most mapfiles define the global symbol
 * requirements together with an auto-reduction directive '*'.  If this state
 * has been defined, then we must make sure that the new symbol isn't a type
 * that can not be demoted to local.
 */
static int
sym_cap_vis(const char *name, Word hash, Sym *sym, Ofl_desc *ofl)
{
        Sym_desc        *sdp;
        uchar_t         vis;
        avl_index_t     where;
        sd_flag_t       sdflags = 0;

        /*
         * Determine the visibility of the new symbol.
         */
        vis = ELF_ST_VISIBILITY(sym->st_other);
        switch (vis) {
        case STV_EXPORTED:
                sdflags |= FLG_SY_EXPORT;
                break;
        case STV_SINGLETON:
                sdflags |= FLG_SY_SINGLE;
                break;
        }

        /*
         * Determine whether a symbol definition already exists, and if so
         * obtain the visibility.
         */
        if ((sdp = ld_sym_find(name, hash, &where, ofl)) != NULL)
                sdflags |= sdp->sd_flags;

        /*
         * Determine whether the symbol flags indicate this symbol should be
         * hidden.
         */
        if ((ofl->ofl_flags & (FLG_OF_AUTOLCL | FLG_OF_AUTOELM)) &&
            ((sdflags & MSK_SY_NOAUTO) == 0))
                sdflags |= FLG_SY_HIDDEN;

        return ((sdflags & FLG_SY_HIDDEN) == 0);
}

/*
 * This routine checks to see if a symbols visibility needs to be reduced to
 * either SYMBOLIC or LOCAL.  This routine can be called from either
 * reloc_init() or sym_validate().
 */
void
ld_sym_adjust_vis(Sym_desc *sdp, Ofl_desc *ofl)
{
        ofl_flag_t      oflags = ofl->ofl_flags;
        Sym             *sym = sdp->sd_sym;

        if ((sdp->sd_ref == REF_REL_NEED) &&
            (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
                /*
                 * If auto-reduction/elimination is enabled, reduce any
                 * non-versioned, and non-local capabilities global symbols.
                 * A symbol is a candidate for auto-reduction/elimination if:
                 *
                 *  -   the symbol wasn't explicitly defined within a mapfile
                 *      (in which case all the necessary state has been applied
                 *      to the symbol), or
```



```
                 *  -   the symbol isn't one of the family of reserved
                 *      special symbols (ie. _end, _etext, etc.), or
                 *  -   the symbol isn't a SINGLETON, or
                 *  -   the symbol wasn't explicitly defined within a version
                 *      definition associated with an input relocatable object.
                 *
                 * Indicate that the symbol has been reduced as it may be
                 * necessary to print these symbols later.
                 */
                if ((oflags & (FLG_OF_AUTOLCL | FLG_OF_AUTOELM)) &&
                    ((sdp->sd_flags & MSK_SY_NOAUTO) == 0)) {
                        if ((sdp->sd_flags & FLG_SY_HIDDEN) == 0) {
                                sdp->sd_flags |=
                                    (FLG_SY_REDUCED | FLG_SY_HIDDEN);
                        }

                        if (oflags & (FLG_OF_REDLSYM | FLG_OF_AUTOELM)) {
                                sdp->sd_flags |= FLG_SY_ELIM;
                                sym->st_other = STV_ELIMINATE |
                                    (sym->st_other & ~MSK_SYM_VISIBILITY);
                        } else if (ELF_ST_VISIBILITY(sym->st_other) !=
                            STV_INTERNAL)
                                sym->st_other = STV_HIDDEN |
                                    (sym->st_other & ~MSK_SYM_VISIBILITY);
                }

                /*
                 * If -Bsymbolic is in effect, and the symbol hasn't explicitly
                 * been defined nodirect (via a mapfile), then bind the global
                 * symbol symbolically and assign the STV_PROTECTED visibility
                 * attribute.
                 */
                if ((oflags & FLG_OF_SYMBOLIC) &&
                    ((sdp->sd_flags & (FLG_SY_HIDDEN | FLG_SY_NDIR)) == 0)) {
                        sdp->sd_flags |= FLG_SY_PROTECT;
                        if (ELF_ST_VISIBILITY(sym->st_other) == STV_DEFAULT)
                                sym->st_other = STV_PROTECTED |
                                    (sym->st_other & ~MSK_SYM_VISIBILITY);
                }
        }

        /*
         * Indicate that this symbol has had it's visibility checked so that
         * we don't need to do this investigation again.
         */
        sdp->sd_flags |= FLG_SY_VISIBLE;
}

/*
 * Make sure a symbol definition is local to the object being built.
 */
inline static int
ensure_sym_local(Ofl_desc *ofl, Sym_desc *sdp, const char *str)
{
        if (sdp->sd_sym->st_shndx == SHN_UNDEF) {
                if (str) {
                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_SYM_UNDEF),
                            str, demangle((char *)sdp->sd_name));
                }
                return (1);
        }
        if (sdp->sd_ref != REF_REL_NEED) {
                if (str) {
                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_SYM_EXTERN),
                            str, demangle((char *)sdp->sd_name),
                            sdp->sd_file->ifl_name);
```

```
1181                 }
1182                 return (1);
1183         }

1185         sdp->sd_flags |= FLG_SY_UPREQD;
1186         if (sdp->sd_isc) {
1187                 sdp->sd_isc->is_flags |= FLG_IS_SECTREF;
1188                 sdp->sd_isc->is_file->ifl_flags |= FLG_IF_FILEREF;
1189         }
1190         return (0);
1191 }

1193 /*
1194  * Make sure all the symbol definitions required for initarray, finiarray, or
1195  * preinitarray's are local to the object being built.
1196  */
1197 static int
1198 ensure_array_local(Ofl_desc *ofl, APlist *apl, const char *str)
1199 {
1200         Aliste          idx;
1201         Sym_desc        *sdp;
1202         int             ret = 0;

1204         for (APLIST_TRAVERSE(apl, idx, sdp))
1205                 ret += ensure_sym_local(ofl, sdp, str);

1207         return (ret);
1208 }

1210 /*
1211  * After all symbol table input processing has been finished, and all relocation
1212  * counting has been carried out (ie. no more symbols will be read, generated,
1213  * or modified), validate and count the relevant entries:
1214  *
1215  *  -   check and print any undefined symbols remaining.  Note that if a symbol
1216  *      has been defined by virtue of the inclusion of  an implicit shared
1217  *      library, it is still classed as undefined.
1218  *
1219  *  -   count the number of global needed symbols together with the size of
1220  *      their associated name strings (if scoping has been indicated these
1221  *      symbols may be reduced to locals).
1222  *
1223  *  -   establish the size and alignment requirements for the global .bss
1224  *      section (the alignment of this section is based on the first symbol
1225  *      that it will contain).
1226  */
1227 uintptr_t
1228 ld_sym_validate(Ofl_desc *ofl)
1229 {
1230         Sym_avlnode     *sav;
1231         Sym_desc        *sdp;
1232         Sym             *sym;
1233         ofl_flag_t      oflags = ofl->ofl_flags;
1234         ofl_flag_t      undef = 0, needed = 0, verdesc = 0;
1235         Xword           bssalign = 0, tlsalign = 0;
1236         Boolean         need_bss, need_tlsbss;
1237         Xword           bsssize = 0, tlssize = 0;
1238 #if     defined(_ELF64)
1239         Xword           lbssalign = 0, lbsssize = 0;
1240         Boolean         need_lbss;
1241 #endif
1242         int             ret, allow_ldynsym;
1243         uchar_t         type;
1244         ofl_flag_t      undef_state = 0;

1246         DBG_CALL(Dbg_basic_validate(ofl->ofl_lml));
```



```
1248         /*
1249          * The need_XXX booleans are used to determine whether we need to
1250          * create each type of bss section. We used to create these sections
1251          * if the sum of the required sizes for each type were non-zero.
1252          * However, it is possible for a compiler to generate COMMON variables
1253          * of zero-length and this tricks that logic --- even zero-length
1254          * symbols need an output section.
1255          */
1256         need_bss = need_tlsbss = FALSE;
1257 #if     defined(_ELF64)
1258         need_lbss = FALSE;
1259 #endif

1261         /*
1262          * Determine how undefined symbols are handled:
1263          *
1264          * fatal:
1265          *      If this link-edit calls for no undefined symbols to remain
1266          *      (this is the default case when generating an executable but
1267          *      can be enforced for any object using -z defs), a fatal error
1268          *      condition will be indicated.
1269          *
1270          * warning:
1271          *      If we're creating a shared object, and either the -Bsymbolic
1272          *      flag is set, or the user has turned on the -z guidance feature,
1273          *      then a non-fatal warning is issued for each symbol.
1274          *
1275          * ignore:
1276          *      In all other cases, undefined symbols are quietly allowed.
1277          */
1278         if (oflags & FLG_OF_NOUNDEF) {
1279                 undef = FLG_OF_FATAL;
1280         } else if (oflags & FLG_OF_SHAROBJ) {
1281                 if ((oflags & FLG_OF_SYMBOLIC) ||
1282                     OFL_GUIDANCE(ofl, FLG_OFG_NO_DEFS))
1283                         undef = FLG_OF_WARN;
1284         }

1286         /*
1287          * If the symbol is referenced from an implicitly included shared object
1288          * (ie. it's not on the NEEDED list) then the symbol is also classified
1289          * as undefined and a fatal error condition will be indicated.
1290          */
1291         if ((oflags & FLG_OF_NOUNDEF) || !(oflags & FLG_OF_SHAROBJ))
1292                 needed = FLG_OF_FATAL;
1293         else if ((oflags & FLG_OF_SHAROBJ) &&
1294             OFL_GUIDANCE(ofl, FLG_OFG_NO_DEFS))
1295                 needed = FLG_OF_WARN;

1297         /*
1298          * If the output image is being versioned, then all symbol definitions
1299          * must be associated with a version.  Any symbol that isn't associated
1300          * with a version is classified as undefined, and a fatal error
1301          * condition is indicated.
1302          */
1303         if ((oflags & FLG_OF_VERDEF) && (ofl->ofl_vercnt > VER_NDX_GLOBAL))
1304                 verdesc = FLG_OF_FATAL;

1306         allow_ldynsym = OFL_ALLOW_LDYNSYM(ofl);

1308         if (allow_ldynsym) {
1309                 /*
1310                  * Normally, we disallow symbols with 0 size from appearing
1311                  * in a dyn[sym|tls]sort section. However, there are some
1312                  * symbols that serve special purposes that we want to exempt
```

```
1313                 * from this rule. Look them up, and set their
1314                 * FLG_SY_DYNSORT flag.
1315                 */
1316                static const char *special[] = {
1317                        MSG_ORIG(MSG_SYM_INIT_U),       /* _init */
1318                        MSG_ORIG(MSG_SYM_FINI_U),       /* _fini */
1319                        MSG_ORIG(MSG_SYM_START),        /* _start */
1320                        NULL
1321                };
1322                int i;

1324                for (i = 0; special[i] != NULL; i++) {
1325                        if (((sdp = ld_sym_find(special[i],
1326                            SYM_NOHASH, NULL, ofl)) != NULL) &&
1327                            (sdp->sd_sym->st_size == 0)) {
1328                                if (ld_sym_copy(sdp) == S_ERROR)
1329                                        return (S_ERROR);
1330                                sdp->sd_flags |= FLG_SY_DYNSORT;
1331                        }
1332                }
1333        }

1335        /*
1336         * Collect and validate the globals from the internal symbol table.
1337         */
1338        for (sav = avl_first(&ofl->ofl_symavl); sav;
1339            sav = AVL_NEXT(&ofl->ofl_symavl, sav)) {
1340                Is_desc         *isp;
1341                int             undeferr = 0;
1342                uchar_t         vis;

1344                sdp = sav->sav_sdp;

1346                /*
1347                 * If undefined symbols are allowed, and we're not being
1348                 * asked to supply guidance, ignore any symbols that are
1349                 * not needed.
1350                 */
1351                if (!(oflags & FLG_OF_NOUNDEF) &&
1352                    !OFL_GUIDANCE(ofl, FLG_OFG_NO_DEFS) &&
1353                    (sdp->sd_ref == REF_DYN_SEEN))
1354                        continue;

1356                /*
1357                 * If the symbol originates from an external or parent mapfile
1358                 * reference and hasn't been matched to a reference from a
1359                 * relocatable object, ignore it.
1360                 */
1361                if ((sdp->sd_flags & (FLG_SY_EXTERN | FLG_SY_PARENT)) &&
1362                    ((sdp->sd_flags & FLG_SY_MAPUSED) == 0)) {
1363                        sdp->sd_flags |= FLG_SY_INVALID;
1364                        continue;
1365                }

1367                sym = sdp->sd_sym;
1368                type = ELF_ST_TYPE(sym->st_info);

1370                /*
1371                 * Sanity check TLS.
1372                 */
1373                if ((type == STT_TLS) && (sym->st_size != 0) &&
1374                    (sym->st_shndx != SHN_UNDEF) &&
1375                    (sym->st_shndx != SHN_COMMON)) {
1376                        Is_desc         *isp = sdp->sd_isc;
1377                        Ifl_desc        *ifl = sdp->sd_file;
```



```
1379                        if ((isp == NULL) || (isp->is_shdr == NULL) ||
1380                            ((isp->is_shdr->sh_flags & SHF_TLS) == 0)) {
1381                                ld_eprintf(ofl, ERR_FATAL,
1382                                    MSG_INTL(MSG_SYM_TLS),
1383                                    demangle(sdp->sd_name), ifl->ifl_name);
1384                                continue;
1385                        }
1386                }

1388                if ((sdp->sd_flags & FLG_SY_VISIBLE) == 0)
1389                        ld_sym_adjust_vis(sdp, ofl);

1391                if ((sdp->sd_flags & FLG_SY_REDUCED) &&
1392                    (oflags & FLG_OF_PROCRED)) {
1393                        DBG_CALL(Dbg_syms_reduce(ofl, DBG_SYM_REDUCE_GLOBAL,
1394                            sdp, 0, 0));
1395                }

1397                /*
1398                 * Record any STV_SINGLETON existence.
1399                 */
1400                if ((vis = ELF_ST_VISIBILITY(sym->st_other)) == STV_SINGLETON)
1401                        ofl->ofl_dtflags_1 |= DF_1_SINGLETON;

1403                /*
1404                 * If building a shared object or executable, and this is a
1405                 * non-weak UNDEF symbol with reduced visibility (STV_*), then
1406                 * give a fatal error.
1407                 */
1408                if (((oflags & FLG_OF_RELOBJ) == 0) &&
1409                    (sym->st_shndx == SHN_UNDEF) &&
1410                    (ELF_ST_BIND(sym->st_info) != STB_WEAK)) {
1411                        if (vis && (vis != STV_SINGLETON)) {
1412                                sym_undef_entry(ofl, sdp, BNDLOCAL,
1413                                    FLG_OF_FATAL, &undef_state);
1414                                continue;
1415                        }
1416                }

1418                /*
1419                 * If this symbol is defined in a non-allocatable section,
1420                 * reduce it to local symbol.
1421                 */
1422                if (((isp = sdp->sd_isc) != 0) && isp->is_shdr &&
1423                    ((isp->is_shdr->sh_flags & SHF_ALLOC) == 0)) {
1424                        sdp->sd_flags |= (FLG_SY_REDUCED | FLG_SY_HIDDEN);
1425                }

1427                /*
1428                 * If this symbol originated as a SHN_SUNW_IGNORE, it will have
1429                 * been processed as an SHN_UNDEF.  Return the symbol to its
1430                 * original index for validation, and propagation to the output
1431                 * file.
1432                 */
1433                if (sdp->sd_flags & FLG_SY_IGNORE)
1434                        sdp->sd_shndx = SHN_SUNW_IGNORE;

1436                if (undef) {
1437                        /*
1438                         * If a non-weak reference remains undefined, or if a
1439                         * mapfile reference is not bound to the relocatable
1440                         * objects that make up the object being built, we have
1441                         * a fatal error.
1442                         *
1443                         * The exceptions are symbols which are defined to be
1444                         * found in the parent (FLG_SY_PARENT), which is really
```

```
1445                         * only meaningful for direct binding, or are defined
1446                         * external (FLG_SY_EXTERN) so as to suppress -zdefs
1447                         * errors.
1448                         *
1449                         * Register symbols are always allowed to be UNDEF.
1450                         *
1451                         * Note that we don't include references created via -u
1452                         * in the same shared object binding test.  This is for
1453                         * backward compatibility, in that a number of archive
1454                         * makefile rules used -u to cause archive extraction.
1455                         * These same rules have been cut and pasted to apply
1456                         * to shared objects, and thus although the -u reference
1457                         * is redundant, flagging it as fatal could cause some
1458                         * build to fail.  Also we have documented the use of
1459                         * -u as a mechanism to cause binding to weak version
1460                         * definitions, thus giving users an error condition
1461                         * would be incorrect.
1462                         */
1463                        if (!(sdp->sd_flags & FLG_SY_REGSYM) &&
1464                            ((sym->st_shndx == SHN_UNDEF) &&
1465                            ((ELF_ST_BIND(sym->st_info) != STB_WEAK) &&
1466                            ((sdp->sd_flags &
1467                            (FLG_SY_PARENT | FLG_SY_EXTERN)) == 0)) ||
1468                            ((sdp->sd_flags &
1469                            (FLG_SY_MAPREF | FLG_SY_MAPUSED | FLG_SY_HIDDEN |
1470                            FLG_SY_PROTECT)) == FLG_SY_MAPREF))) {
1471                                sym_undef_entry(ofl, sdp, UNDEF, undef,
1472                                    &undef_state);
1473                                undeferr = 1;
1474                        }

1476                } else {
1477                        /*
1478                         * For building things like shared objects (or anything
1479                         * -znodefs), undefined symbols are allowed.
1480                         *
1481                         * If a mapfile reference remains undefined the user
1482                         * would probably like a warning at least (they've
1483                         * usually mis-spelt the reference).  Refer to the above
1484                         * comments for discussion on -u references, which
1485                         * are not tested for in the same manner.
1486                         */
1487                        if ((sdp->sd_flags &
1488                            (FLG_SY_MAPREF | FLG_SY_MAPUSED)) ==
1489                            FLG_SY_MAPREF) {
1490                                sym_undef_entry(ofl, sdp, UNDEF, FLG_OF_WARN,
1491                                    &undef_state);
1492                                undeferr = 1;
1493                        }
1494                }

1496                /*
1497                 * If this symbol comes from a dependency mark the dependency
1498                 * as required (-z ignore can result in unused dependencies
1499                 * being dropped).  If we need to record dependency versioning
1500                 * information indicate what version of the needed shared object
1501                 * this symbol is part of.  Flag the symbol as undefined if it
1502                 * has not been made available to us.
1503                 */
1504                if ((sdp->sd_ref == REF_DYN_NEED) &&
1505                    (!(sdp->sd_flags & FLG_SY_REFRSD))) {
1506                        sdp->sd_file->ifl_flags |= FLG_IF_DEPREQD;

1508                        /*
1509                         * Capture that we've bound to a symbol that doesn't
1510                         * allow being directly bound to.
```



```
1511                         */
1512                        if (sdp->sd_flags & FLG_SY_NDIR)
1513                                ofl->ofl_flags1 |= FLG_OF1_NGLBDIR;

1515                        if (sdp->sd_file->ifl_vercnt) {
1516                                int             vndx;
1517                                Ver_index       *vip;

1519                                vndx = sdp->sd_aux->sa_dverndx;
1520                                vip = &sdp->sd_file->ifl_verndx[vndx];
1521                                if (vip->vi_flags & FLG_VER_AVAIL) {
1522                                        vip->vi_flags |= FLG_VER_REFER;
1523                                } else {
1524                                        sym_undef_entry(ofl, sdp, NOTAVAIL,
1525                                            FLG_OF_FATAL, &undef_state);
1526                                        continue;
1527                                }
1528                        }
1529                }

1531                /*
1532                 * Test that we do not bind to symbol supplied from an implicit
1533                 * shared object.  If a binding is from a weak reference it can
1534                 * be ignored.
1535                 */
1536                if (needed && !undeferr && (sdp->sd_flags & FLG_SY_GLOBREF) &&
1537                    (sdp->sd_ref == REF_DYN_NEED) &&
1538                    (sdp->sd_flags & FLG_SY_NOTAVAIL)) {
1539                        sym_undef_entry(ofl, sdp, IMPLICIT, needed,
1540                            &undef_state);
1541                        if (needed == FLG_OF_FATAL)
1542                                continue;
1543                }

1545                /*
1546                 * Test that a symbol isn't going to be reduced to local scope
1547                 * which actually wants to bind to a shared object - if so it's
1548                 * a fatal error.
1549                 */
1550                if ((sdp->sd_ref == REF_DYN_NEED) &&
1551                    (sdp->sd_flags & (FLG_SY_HIDDEN | FLG_SY_PROTECT))) {
1552                        sym_undef_entry(ofl, sdp, BNDLOCAL, FLG_OF_FATAL,
1553                            &undef_state);
1554                        continue;
1555                }

1557                /*
1558                 * If the output image is to be versioned then all symbol
1559                 * definitions must be associated with a version.  Remove any
1560                 * versioning that might be left associated with an undefined
1561                 * symbol.
1562                 */
1563                if (verdesc && (sdp->sd_ref == REF_REL_NEED)) {
1564                        if (sym->st_shndx == SHN_UNDEF) {
1565                                if (sdp->sd_aux && sdp->sd_aux->sa_overndx)
1566                                        sdp->sd_aux->sa_overndx = 0;
1567                        } else {
1568                                if (!SYM_IS_HIDDEN(sdp) && sdp->sd_aux &&
1569                                    (sdp->sd_aux->sa_overndx == 0)) {
1570                                        sym_undef_entry(ofl, sdp, NOVERSION,
1571                                            verdesc, &undef_state);
1572                                        continue;
1573                                }
1574                        }
1575                }
```

```
1577                /*
1578                 * If we don't need the symbol there's no need to process it
1579                 * any further.
1580                 */
1581                if (sdp->sd_ref == REF_DYN_SEEN)
1582                        continue;

1584                /*
1585                 * Calculate the size and alignment requirements for the global
1586                 * .bss and .tls sections.  If we're building a relocatable
1587                 * object only account for scoped COMMON symbols (these will
1588                 * be converted to .bss references).
1589                 *
1590                 * When -z nopartial is in effect, partially initialized
1591                 * symbols are directed to the special .data section
1592                 * created for that purpose (ofl->ofl_isparexpn).
1593                 * Otherwise, partially initialized symbols go to .bss.
1594                 *
1595                 * Also refer to make_mvsections() in sunwmove.c
1596                 */
1597                if ((sym->st_shndx == SHN_COMMON) &&
1598                    (((oflags & FLG_OF_RELOBJ) == 0) ||
1599                    (SYM_IS_HIDDEN(sdp) && (oflags & FLG_OF_PROCRED)))) {
1600                        if ((sdp->sd_move == NULL) ||
1601                            ((sdp->sd_flags & FLG_SY_PAREXPN) == 0)) {
1602                                if (type != STT_TLS) {
1603                                        need_bss = TRUE;
1604                                        bsssize = (Xword)S_ROUND(bsssize,
1605                                            sym->st_value) + sym->st_size;
1606                                        if (sym->st_value > bssalign)
1607                                                bssalign = sym->st_value;
1608                                } else {
1609                                        need_tlsbss = TRUE;
1610                                        tlssize = (Xword)S_ROUND(tlssize,
1611                                            sym->st_value) + sym->st_size;
1612                                        if (sym->st_value > tlsalign)
1613                                                tlsalign = sym->st_value;
1614                                }
1615                        }
1616                }

1618 #if     defined(_ELF64)
1619                /*
1620                 * Calculate the size and alignment requirement for the global
1621                 * .lbss. TLS or partially initialized symbols do not need to be
1622                 * considered yet.
1623                 */
1624                if ((ld_targ.t_m.m_mach == EM_AMD64) &&
1625                    (sym->st_shndx == SHN_X86_64_LCOMMON)) {
1626                        need_lbss = TRUE;
1627                        lbsssize = (Xword)S_ROUND(lbsssize, sym->st_value) +
1628                            sym->st_size;
1629                        if (sym->st_value > lbssalign)
1630                                lbssalign = sym->st_value;
1631                }
1632 #endif
1633                /*
1634                 * If a symbol was referenced via the command line
1635                 * (ld -u <>, ...), then this counts as a reference against the
1636                 * symbol. Mark any section that symbol is defined in.
1637                 */
1638                if (((isp = sdp->sd_isc) != 0) &&
1639                    (sdp->sd_flags & FLG_SY_CMDREF)) {
1640                        isp->is_flags |= FLG_IS_SECTREF;
1641                        isp->is_file->ifl_flags |= FLG_IF_FILEREF;
1642                }
```



```
1644                /*
1645                 * Update the symbol count and the associated name string size.
1646                 * Note, a capabilities symbol must remain as visible as a
1647                 * global symbol.  However, the runtime linker recognizes the
1648                 * hidden requirement and ensures the symbol isn't made globally
1649                 * available at runtime.
1650                 */
1651                if (SYM_IS_HIDDEN(sdp) && (oflags & FLG_OF_PROCRED)) {
1652                        /*
1653                         * If any reductions are being processed, keep a count
1654                         * of eliminated symbols, and if the symbol is being
1655                         * reduced to local, count it's size for the .symtab.
1656                         */
1657                        if (sdp->sd_flags & FLG_SY_ELIM) {
1658                                ofl->ofl_elimcnt++;
1659                        } else {
1660                                ofl->ofl_scopecnt++;
1661                                if ((((sdp->sd_flags & FLG_SY_REGSYM) == 0) ||
1662                                    sym->st_name) && (st_insert(ofl->ofl_strtab,
1663                                    sdp->sd_name) == -1))
1664                                        return (S_ERROR);
1665                                if (allow_ldynsym && sym->st_name &&
1666                                    ldynsym_symtype[type]) {
1667                                        ofl->ofl_dynscopecnt++;
1668                                        if (st_insert(ofl->ofl_dynstrtab,
1669                                            sdp->sd_name) == -1)
1670                                                return (S_ERROR);
1671                                        /* Include it in sort section? */
1672                                        DYNSORT_COUNT(sdp, sym, type, ++);
1673                                }
1674                        }
1675                } else {
1676                        ofl->ofl_globcnt++;

1678                        /*
1679                         * Check to see if this global variable should go into
1680                         * a sort section. Sort sections require a
1681                         * .SUNW_ldynsym section, so, don't check unless a
1682                         * .SUNW_ldynsym is allowed.
1683                         */
1684                        if (allow_ldynsym)
1685                                DYNSORT_COUNT(sdp, sym, type, ++);

1687                        /*
1688                         * If global direct bindings are in effect, or this
1689                         * symbol has bound to a dependency which was specified
1690                         * as requiring direct bindings, and it hasn't
1691                         * explicitly been defined as a non-direct binding
1692                         * symbol, mark it.
1693                         */
1694                        if (((ofl->ofl_dtflags_1 & DF_1_DIRECT) || (isp &&
1695                            (isp->is_file->ifl_flags & FLG_IF_DIRECT))) &&
1696                            ((sdp->sd_flags & FLG_SY_NDIR) == 0))
1697                                sdp->sd_flags |= FLG_SY_DIR;

1699                        /*
1700                         * Insert the symbol name.
1701                         */
1702                        if (((sdp->sd_flags & FLG_SY_REGSYM) == 0) ||
1703                            sym->st_name) {
1704                                if (st_insert(ofl->ofl_strtab,
1705                                    sdp->sd_name) == -1)
1706                                        return (S_ERROR);

1708                                if (!(ofl->ofl_flags & FLG_OF_RELOBJ) &&
```

```
1709                                    (st_insert(ofl->ofl_dynstrtab,
1710                                    sdp->sd_name) == -1))
1711                                        return (S_ERROR);
1712                        }
1713
1714                        /*
1715                         * If this section offers a global symbol - record that
1716                         * fact.
1717                         */
1718                        if (isp) {
1719                                isp->is_flags |= FLG_IS_SECTREF;
1720                                isp->is_file->ifl_flags |= FLG_IF_FILEREF;
1721                        }
1722                }
1723        }
1724
1725        /*
1726         * Guidance: Use -z defs|nodefs when building shared objects.
1727         *
1728         * Our caller issues this, unless we mask it out here. So we mask it
1729         * out unless we've issued at least one warnings or fatal error.
1730         */
1731        if (!((oflags & FLG_OF_SHAROBJ) && OFL_GUIDANCE(ofl, FLG_OFG_NO_DEFS) &&
1732            (undef_state & (FLG_OF_FATAL | FLG_OF_WARN))))
1733                ofl->ofl_guideflags |= FLG_OFG_NO_DEFS;
1734
1735        /*
1736         * If we've encountered a fatal error during symbol validation then
1737         * return now.
1738         */
1739        if (ofl->ofl_flags & FLG_OF_FATAL)
1740                return (1);
1741
1742        /*
1743         * Now that symbol resolution is completed, scan any register symbols.
1744         * From now on, we're only interested in those that contribute to the
1745         * output file.
1746         */
1747        if (ofl->ofl_regsyms) {
1748                int     ndx;
1749
1750                for (ndx = 0; ndx < ofl->ofl_regsymsno; ndx++) {
1751                        if ((sdp = ofl->ofl_regsyms[ndx]) == NULL)
1752                                continue;
1753                        if (sdp->sd_ref != REF_REL_NEED) {
1754                                ofl->ofl_regsyms[ndx] = NULL;
1755                                continue;
1756                        }
1757
1758                        ofl->ofl_regsymcnt++;
1759                        if (sdp->sd_sym->st_name == 0)
1760                                sdp->sd_name = MSG_ORIG(MSG_STR_EMPTY);
1761
1762                        if (SYM_IS_HIDDEN(sdp) ||
1763                            (ELF_ST_BIND(sdp->sd_sym->st_info) == STB_LOCAL))
1764                                ofl->ofl_lregsymcnt++;
1765                }
1766        }
1767
1768        /*
1769         * Generate the .bss section now that we know its size and alignment.
1770         */
1771        if (need_bss) {
1772                if (ld_make_bss(ofl, bsssize, bssalign,
1773                    ld_targ.t_id.id_bss) == S_ERROR)
1774                        return (S_ERROR);
```



```
1775        }
1776        if (need_tlsbss) {
1777                if (ld_make_bss(ofl, tlssize, tlsalign,
1778                    ld_targ.t_id.id_tlsbss) == S_ERROR)
1779                        return (S_ERROR);
1780        }
1781 #if     defined(_ELF64)
1782        if ((ld_targ.t_m.m_mach == EM_AMD64) &&
1783            need_lbss && !(oflags & FLG_OF_RELOBJ)) {
1784                if (ld_make_bss(ofl, lbsssize, lbssalign,
1785                    ld_targ.t_id.id_lbss) == S_ERROR)
1786                        return (S_ERROR);
1787        }
1788 #endif
1789        /*
1790         * Determine what entry point symbol we need, and if found save its
1791         * symbol descriptor so that we can update the ELF header entry with the
1792         * symbols value later (see update_oehdr).  Make sure the symbol is
1793         * tagged to ensure its update in case -s is in effect.  Use any -e
1794         * option first, or the default entry points '_start' and 'main'.
1795         */
1796        ret = 0;
1797        if (ofl->ofl_entry) {
1798                if ((sdp = ld_sym_find(ofl->ofl_entry, SYM_NOHASH,
1799                    NULL, ofl)) == NULL) {
1800                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_NOENTRY),
1801                            ofl->ofl_entry);
1802                        ret++;
1803                } else if (ensure_sym_local(ofl, sdp,
1804                    MSG_INTL(MSG_SYM_ENTRY)) != 0) {
1805                        ret++;
1806                } else {
1807                        ofl->ofl_entry = (void *)sdp;
1808                }
1809        } else if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_START),
1810            SYM_NOHASH, NULL, ofl)) != NULL) && (ensure_sym_local(ofl,
1811            sdp, 0) == 0)) {
1812                ofl->ofl_entry = (void *)sdp;
1813
1814        } else if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_MAIN),
1815            SYM_NOHASH, NULL, ofl)) != NULL) && (ensure_sym_local(ofl,
1816            sdp, 0) == 0)) {
1817                ofl->ofl_entry = (void *)sdp;
1818        }
1819
1820        /*
1821         * If ld -zdtrace=<sym> was given, then validate that the symbol is
1822         * defined within the current object being built.
1823         */
1824        if ((sdp = ofl->ofl_dtracesym) != 0)
1825                ret += ensure_sym_local(ofl, sdp, MSG_ORIG(MSG_STR_DTRACE));
1826
1827        /*
1828         * If any initarray, finiarray or preinitarray functions have been
1829         * requested, make sure they are defined within the current object
1830         * being built.
1831         */
1832        if (ofl->ofl_initarray) {
1833                ret += ensure_array_local(ofl, ofl->ofl_initarray,
1834                    MSG_ORIG(MSG_SYM_INITARRAY));
1835        }
1836        if (ofl->ofl_finiarray) {
1837                ret += ensure_array_local(ofl, ofl->ofl_finiarray,
1838                    MSG_ORIG(MSG_SYM_FINIARRAY));
1839        }
1840        if (ofl->ofl_preiarray) {
```

```
1841                 ret += ensure_array_local(ofl, ofl->ofl_preiarray,
1842                     MSG_ORIG(MSG_SYM_PREINITARRAY));
1843         }

1845         if (ret)
1846                 return (S_ERROR);

1848         /*
1849          * If we're required to record any needed dependencies versioning
1850          * information calculate it now that all symbols have been validated.
1851          */
1852         if ((oflags & (FLG_OF_VERNEED | FLG_OF_NOVERSEC)) == FLG_OF_VERNEED)
1853                 return (ld_vers_check_need(ofl));
1854         else
1855                 return (1);
1856 }

1858 /*
1859  * qsort(3c) comparison function.  As an optimization for associating weak
1860  * symbols to their strong counterparts sort global symbols according to their
1861  * section index, address and binding.
1862  */
1863 static int
1864 compare(const void *sdpp1, const void *sdpp2)
1865 {
1866         Sym_desc        *sdp1 = *((Sym_desc **)sdpp1);
1867         Sym_desc        *sdp2 = *((Sym_desc **)sdpp2);
1868         Sym             *sym1, *sym2;
1869         uchar_t         bind1, bind2;

1871         /*
1872          * Symbol descriptors may be zero, move these to the front of the
1873          * sorted array.
1874          */
1875         if (sdp1 == NULL)
1876                 return (-1);
1877         if (sdp2 == NULL)
1878                 return (1);

1880         sym1 = sdp1->sd_sym;
1881         sym2 = sdp2->sd_sym;

1883         /*
1884          * Compare the symbols section index.  This is important when sorting
1885          * the symbol tables of relocatable objects.  In this case, a symbols
1886          * value is the offset within the associated section, and thus many
1887          * symbols can have the same value, but are effectively different
1888          * addresses.
1889          */
1890         if (sym1->st_shndx > sym2->st_shndx)
1891                 return (1);
1892         if (sym1->st_shndx < sym2->st_shndx)
1893                 return (-1);

1895         /*
1896          * Compare the symbols value (address).
1897          */
1898         if (sym1->st_value > sym2->st_value)
1899                 return (1);
1900         if (sym1->st_value < sym2->st_value)
1901                 return (-1);

1903         bind1 = ELF_ST_BIND(sym1->st_info);
1904         bind2 = ELF_ST_BIND(sym2->st_info);

1906         /*
```



```
1907          * If two symbols have the same address place the weak symbol before
1908          * any strong counterpart.
1909          */
1910         if (bind1 > bind2)
1911                 return (-1);
1912         if (bind1 < bind2)
1913                 return (1);

1915         return (0);
1916 }

1918 /*
1919  * Issue a MSG_SYM_BADADDR error from ld_sym_process(). This error
1920  * is issued when a symbol address/size is not contained by the
1921  * target section.
1922  *
1923  * Such objects are at least partially corrupt, and the user would
1924  * be well advised to be skeptical of them, and to ask their compiler
1925  * supplier to fix the problem. However, a distinction needs to be
1926  * made between symbols that reference readonly text, and those that
1927  * access writable data. Other than throwing off profiling results,
1928  * the readonly section case is less serious. We have encountered
1929  * such objects in the field. In order to allow existing objects
1930  * to continue working, we issue a warning rather than a fatal error
1931  * if the symbol is against readonly text. Other cases are fatal.
1932  */
1933 static void
1934 issue_badaddr_msg(Ifl_desc *ifl, Ofl_desc *ofl, Sym_desc *sdp,
1935     Sym *sym, Word shndx)
1936 {
1937         Error           err;
1938         const char      *msg;

1940         if ((sdp->sd_isc->is_shdr->sh_flags & (SHF_WRITE | SHF_ALLOC)) ==
1941             SHF_ALLOC) {
1942                 msg = MSG_INTL(MSG_SYM_BADADDR_ROTXT);
1943                 err = ERR_WARNING;
1944         } else {
1945                 msg = MSG_INTL(MSG_SYM_BADADDR);
1946                 err = ERR_FATAL;
1947         }

1949         ld_eprintf(ofl, err, msg, demangle(sdp->sd_name),
1950             ifl->ifl_name, shndx, sdp->sd_isc->is_name,
1951             EC_XWORD(sdp->sd_isc->is_shdr->sh_size),
1952             EC_XWORD(sym->st_value), EC_XWORD(sym->st_size));
1953 }

1955 /*
1956  * Global symbols that are candidates for translation to local capability
1957  * symbols under -z symbolcap, are maintained on a local symbol list.  Once
1958  * all symbols of a file are processed, this list is traversed to cull any
1959  * unnecessary weak symbol aliases.
1960  */
1961 typedef struct {
1962         Sym_desc        *c_nsdp;        /* new lead symbol */
1963         Sym_desc        *c_osdp;        /* original symbol */
1964         Cap_group       *c_group;       /* symbol capability group */
1965         Word            c_ndx;          /* symbol index */
1966 } Cap_pair;

1968 /*
1969  * Process the symbol table for the specified input file.  At this point all
1970  * input sections from this input file have been assigned an input section
1971  * descriptor which is saved in the 'ifl_isdesc' array.
1972  *
```

```
 *  -   local symbols are saved (as is) if the input file is a  relocatable
 *      object
 *
 *  -   global symbols are added to the linkers internal symbol table if they
 *      are not already present, otherwise a symbol resolution function is
 *      called upon to resolve the conflict.
 */
uintptr_t
ld_sym_process(Is_desc *isc, Ifl_desc *ifl, Ofl_desc *ofl)
{
        /*
         * This macro tests the given symbol to see if it is out of
         * range relative to the section it references.
         *
         * entry:
         *      - ifl is a relative object (ET_REL)
         *      _sdp - Symbol descriptor
         *      _sym - Symbol
         *      _type - Symbol type
         *
         * The following are tested:
         *      - Symbol length is non-zero
         *      - Symbol type is a type that references code or data
         *      - Referenced section is not 0 (indicates an UNDEF symbol)
         *        and is not in the range of special values above SHN_LORESERVE
         *        (excluding SHN_XINDEX, which is OK).
         *      - We have a valid section header for the target section
         *
         * If the above are all true, and the symbol position is not
         * contained by the target section, this macro evaluates to
         * True (1). Otherwise, False(0).
         */
#define SYM_LOC_BADADDR(_sdp, _sym, _type) \
        (_sym->st_size && dynsymsort_symtype[_type] && \
        (_sym->st_shndx != SHN_UNDEF) && \
        ((_sym->st_shndx < SHN_LORESERVE) || \
                (_sym->st_shndx == SHN_XINDEX)) && \
        _sdp->sd_isc && _sdp->sd_isc->is_shdr && \
        ((_sym->st_value + _sym->st_size) > _sdp->sd_isc->is_shdr->sh_size))

        Conv_inv_buf_t  inv_buf;
        Sym             *sym = (Sym *)isc->is_indata->d_buf;
        Word            *symshndx = NULL;
        Shdr            *shdr = isc->is_shdr;
        Sym_desc        *sdp;
        size_t          strsize;
        char            *strs;
        uchar_t         type, bind;
        Word            ndx, hash, local, total;
        uchar_t         osabi = ifl->ifl_ehdr->e_ident[EI_OSABI];
        Half            mach = ifl->ifl_ehdr->e_machine;
        Half            etype = ifl->ifl_ehdr->e_type;
        int             etype_rel;
        const char      *symsecname, *strsecname;
        Word            symsecndx;
        avl_index_t     where;
        int             test_gnu_hidden_bit, weak;
        Cap_desc        *cdp = NULL;
        Alist           *cappairs = NULL;

        /*
         * Its possible that a file may contain more that one symbol table,
         * ie. .dynsym and .symtab in a shared library.  Only process the first
         * table (here, we assume .dynsym comes before .symtab).
         */
        if (ifl->ifl_symscnt)
```



```
                return (1);

        if (isc->is_symshndx)
                symshndx = isc->is_symshndx->is_indata->d_buf;

        DBG_CALL(Dbg_syms_process(ofl->ofl_lml, ifl));

        symsecndx = isc->is_scnndx;
        if (isc->is_name)
                symsecname = isc->is_name;
        else
                symsecname = MSG_ORIG(MSG_STR_EMPTY);

        /*
         * From the symbol tables section header information determine which
         * strtab table is needed to locate the actual symbol names.
         */
        if (ifl->ifl_flags & FLG_IF_HSTRTAB) {
                ndx = shdr->sh_link;
                if ((ndx == 0) || (ndx >= ifl->ifl_shnum)) {
                        ld_eprintf(ofl, ERR_FATAL,
                            MSG_INTL(MSG_FIL_INVSHLINK), ifl->ifl_name,
                            EC_WORD(symsecndx), symsecname, EC_XWORD(ndx));
                        return (S_ERROR);
                }
                strsize = ifl->ifl_isdesc[ndx]->is_shdr->sh_size;
                strs = ifl->ifl_isdesc[ndx]->is_indata->d_buf;
                if (ifl->ifl_isdesc[ndx]->is_name)
                        strsecname = ifl->ifl_isdesc[ndx]->is_name;
                else
                        strsecname = MSG_ORIG(MSG_STR_EMPTY);
        } else {
                /*
                 * There is no string table section in this input file
                 * although there are symbols in this symbol table section.
                 * This means that these symbols do not have names.
                 * Currently, only scratch register symbols are allowed
                 * not to have names.
                 */
                strsize = 0;
                strs = (char *)MSG_ORIG(MSG_STR_EMPTY);
                strsecname = MSG_ORIG(MSG_STR_EMPTY);
        }

        /*
         * Determine the number of local symbols together with the total
         * number we have to process.
         */
        total = (Word)(shdr->sh_size / shdr->sh_entsize);
        local = shdr->sh_info;

        /*
         * Allocate a symbol table index array and a local symbol array
         * (global symbols are processed and added to the ofl->ofl_symbkt[]
         * array).  If we are dealing with a relocatable object, allocate the
         * local symbol descriptors.  If this isn't a relocatable object we
         * still have to process any shared object locals to determine if any
         * register symbols exist.  Although these aren't added to the output
         * image, they are used as part of symbol resolution.
         */
        if ((ifl->ifl_oldndx = libld_malloc((size_t)(total *
            sizeof (Sym_desc *)))) == NULL)
                return (S_ERROR);
        etype_rel = (etype == ET_REL);
        if (etype_rel && local) {
                if ((ifl->ifl_locs =
```

```
2105                    libld_calloc(sizeof (Sym_desc), local)) == NULL)
2106                        return (S_ERROR);
2107                /* LINTED */
2108                ifl->ifl_locscnt = (Word)local;
2109        }
2110        ifl->ifl_symscnt = total;
2111
2112        /*
2113         * If there are local symbols to save add them to the symbol table
2114         * index array.
2115         */
2116        if (local) {
2117                int             allow_ldynsym = OFL_ALLOW_LDYNSYM(ofl);
2118                Sym_desc        *last_file_sdp = NULL;
2119                int             last_file_ndx = 0;
2120
2121                for (sym++, ndx = 1; ndx < local; sym++, ndx++) {
2122                        sd_flag_t       sdflags = FLG_SY_CLEAN;
2123                        Word            shndx;
2124                        const char      *name;
2125                        Sym_desc        *rsdp;
2126                        int             shndx_bad = 0;
2127                        int             symtab_enter = 1;
2128
2129                        /*
2130                         * Determine and validate the associated section index.
2131                         */
2132                        if (symshndx && (sym->st_shndx == SHN_XINDEX)) {
2133                                shndx = symshndx[ndx];
2134                        } else if ((shndx = sym->st_shndx) >= SHN_LORESERVE) {
2135                                sdflags |= FLG_SY_SPECSEC;
2136                        } else if (shndx > ifl->ifl_shnum) {
2137                                /* We need the name before we can issue error */
2138                                shndx_bad = 1;
2139                        }
2140
2141                        /*
2142                         * Check if st_name has a valid value or not.
2143                         */
2144                        if ((name = string(ofl, ifl, sym, strs, strsize, ndx,
2145                            shndx, symsecndx, symsecname, strsecname,
2146                            &sdflags)) == NULL)
2147                                continue;
2148
2149                        /*
2150                         * Now that we have the name, if the section index
2151                         * was bad, report it.
2152                         */
2153                        if (shndx_bad) {
2154                                ld_eprintf(ofl, ERR_WARNING,
2155                                    MSG_INTL(MSG_SYM_INVSHNDX),
2156                                    demangle_symname(name, symsecname, ndx),
2157                                    ifl->ifl_name,
2158                                    conv_sym_shndx(osabi, mach, sym->st_shndx,
2159                                    CONV_FMT_DECIMAL, &inv_buf));
2160                                continue;
2161                        }
2162
2163                        /*
2164                         * If this local symbol table originates from a shared
2165                         * object, then we're only interested in recording
2166                         * register symbols.  As local symbol descriptors aren't
2167                         * allocated for shared objects, one will be allocated
2168                         * to associated with the register symbol.  This symbol
2169                         * won't become part of the output image, but we must
2170                         * process it to test for register conflicts.
```



```
2171                         */
2172                        rsdp = sdp = NULL;
2173                        if (sdflags & FLG_SY_REGSYM) {
2174                                /*
2175                                 * The presence of FLG_SY_REGSYM means that
2176                                 * the pointers in ld_targ.t_ms are non-NULL.
2177                                 */
2178                                rsdp = (*ld_targ.t_ms.ms_reg_find)(sym, ofl);
2179                                if (rsdp != 0) {
2180                                        /*
2181                                         * The fact that another register def-
2182                                         * inition has been found is fatal.
2183                                         * Call the verification routine to get
2184                                         * the error message and move on.
2185                                         */
2186                                        (void) (*ld_targ.t_ms.ms_reg_check)
2187                                            (rsdp, sym, name, ifl, ofl);
2188                                        continue;
2189                                }
2190
2191                                if (etype == ET_DYN) {
2192                                        if ((sdp = libld_calloc(
2193                                            sizeof (Sym_desc), 1)) == NULL)
2194                                                return (S_ERROR);
2195                                        sdp->sd_ref = REF_DYN_SEEN;
2196
2197                                        /* Will not appear in output object */
2198                                        symtab_enter = 0;
2199                                }
2200                        } else if (etype == ET_DYN)
2201                                continue;
2202
2203                        /*
2204                         * Fill in the remaining symbol descriptor information.
2205                         */
2206                        if (sdp == NULL) {
2207                                sdp = &(ifl->ifl_locs[ndx]);
2208                                sdp->sd_ref = REF_REL_NEED;
2209                                sdp->sd_symndx = ndx;
2210                        }
2211                        if (rsdp == NULL) {
2212                                sdp->sd_name = name;
2213                                sdp->sd_sym = sym;
2214                                sdp->sd_shndx = shndx;
2215                                sdp->sd_flags = sdflags;
2216                                sdp->sd_file = ifl;
2217                                ifl->ifl_oldndx[ndx] = sdp;
2218                        }
2219
2220                        DBG_CALL(Dbg_syms_entry(ofl->ofl_lml, ndx, sdp));
2221
2222                        /*
2223                         * Reclassify any SHN_SUNW_IGNORE symbols to SHN_UNDEF
2224                         * so as to simplify future processing.
2225                         */
2226                        if (sym->st_shndx == SHN_SUNW_IGNORE) {
2227                                sdp->sd_shndx = shndx = SHN_UNDEF;
2228                                sdp->sd_flags |= (FLG_SY_IGNORE | FLG_SY_ELIM);
2229                        }
2230
2231                        /*
2232                         * Process any register symbols.
2233                         */
2234                        if (sdp->sd_flags & FLG_SY_REGSYM) {
2235                                /*
2236                                 * Add a diagnostic to indicate we've caught a
```

```
2237                                 * register symbol, as this can be useful if a
2238                                 * register conflict is later discovered.
2239                                 */
2240                                DBG_CALL(Dbg_syms_entered(ofl, sym, sdp));

2242                                /*
2243                                 * If this register symbol hasn't already been
2244                                 * recorded, enter it now.
2245                                 *
2246                                 * The presence of FLG_SY_REGSYM means that
2247                                 * the pointers in ld_targ.t_ms are non-NULL.
2248                                 */
2249                                if ((rsdp == NULL) &&
2250                                    ((*ld_targ.t_ms.ms_reg_enter)(sdp, ofl) ==
2251                                    0))
2252                                        return (S_ERROR);
2253                        }

2255                        /*
2256                         * Assign an input section.
2257                         */
2258                        if ((sym->st_shndx != SHN_UNDEF) &&
2259                            ((sdp->sd_flags & FLG_SY_SPECSEC) == 0))
2260                                sdp->sd_isc = ifl->ifl_isdesc[shndx];

2262                        /*
2263                         * If this symbol falls within the range of a section
2264                         * being discarded, then discard the symbol itself.
2265                         * There is no reason to keep this local symbol.
2266                         */
2267                        if (sdp->sd_isc &&
2268                            (sdp->sd_isc->is_flags & FLG_IS_DISCARD)) {
2269                                sdp->sd_flags |= FLG_SY_ISDISC;
2270                                DBG_CALL(Dbg_syms_discarded(ofl->ofl_lml, sdp));
2271                                continue;
2272                        }

2274                        /*
2275                         * Skip any section symbols as new versions of these
2276                         * will be created.
2277                         */
2278                        if ((type = ELF_ST_TYPE(sym->st_info)) == STT_SECTION) {
2279                                if (sym->st_shndx == SHN_UNDEF) {
2280                                        ld_eprintf(ofl, ERR_WARNING,
2281                                            MSG_INTL(MSG_SYM_INVSHNDX),
2282                                            demangle_symname(name, symsecname,
2283                                            ndx), ifl->ifl_name,
2284                                            conv_sym_shndx(osabi, mach,
2285                                            sym->st_shndx, CONV_FMT_DECIMAL,
2286                                            &inv_buf));
2287                                }
2288                                continue;
2289                        }

2291                        /*
2292                         * For a relocatable object, if this symbol is defined
2293                         * and has non-zero length and references an address
2294                         * within an associated section, then check its extents
2295                         * to make sure the section boundaries encompass it.
2296                         * If they don't, the ELF file is corrupt.
2297                         */
2298                        if (etype_rel) {
2299                                if (SYM_LOC_BADADDR(sdp, sym, type)) {
2300                                        issue_badaddr_msg(ifl, ofl, sdp,
2301                                            sym, shndx);
2302                                        if (ofl->ofl_flags & FLG_OF_FATAL)
```



```
2303                                                continue;
2304                                }

2306                                /*
2307                                 * We have observed relocatable objects
2308                                 * containing identical adjacent STT_FILE
2309                                 * symbols. Discard any other than the first,
2310                                 * as they are all equivalent and the extras
2311                                 * do not add information.
2312                                 *
2313                                 * For the purpose of this test, we assume
2314                                 * that only the symbol type and the string
2315                                 * table offset (st_name) matter.
2316                                 */
2317                                if (type == STT_FILE) {
2318                                        int toss = (last_file_sdp != NULL) &&
2319                                            ((ndx - 1) == last_file_ndx) &&
2320                                            (sym->st_name ==
2321                                            last_file_sdp->sd_sym->st_name);

2323                                        last_file_sdp = sdp;
2324                                        last_file_ndx = ndx;
2325                                        if (toss) {
2326                                                sdp->sd_flags |= FLG_SY_INVALID;
2327                                                DBG_CALL(Dbg_syms_dup_discarded(
2328                                                    ofl->ofl_lml, ndx, sdp));
2329                                                continue;
2330                                        }
2331                                }
2332                        }


2335                        /*
2336                         * Sanity check for TLS
2337                         */
2338                        if ((sym->st_size != 0) && ((type == STT_TLS) &&
2339                            (sym->st_shndx != SHN_COMMON))) {
2340                                Is_desc *isp = sdp->sd_isc;

2342                                if ((isp == NULL) || (isp->is_shdr == NULL) ||
2343                                    ((isp->is_shdr->sh_flags & SHF_TLS) == 0)) {
2344                                        ld_eprintf(ofl, ERR_FATAL,
2345                                            MSG_INTL(MSG_SYM_TLS),
2346                                            demangle(sdp->sd_name),
2347                                            ifl->ifl_name);
2348                                        continue;
2349                                }
2350                        }

2352                        /*
2353                         * Carry our some basic sanity checks (these are just
2354                         * some of the erroneous symbol entries we've come
2355                         * across, there's probably a lot more).  The symbol
2356                         * will not be carried forward to the output file, which
2357                         * won't be a problem unless a relocation is required
2358                         * against it.
2359                         */
2360                        if (((sdp->sd_flags & FLG_SY_SPECSEC) &&
2361                            ((sym->st_shndx == SHN_COMMON)) ||
2362                            ((type == STT_FILE) &&
2363                            (sym->st_shndx != SHN_ABS))) ||
2364                            (sdp->sd_isc && (sdp->sd_isc->is_osdesc == NULL))) {
2365                                ld_eprintf(ofl, ERR_WARNING,
2366                                    MSG_INTL(MSG_SYM_INVSHNDX),
2367                                    demangle_symname(name, symsecname, ndx),
2368                                    ifl->ifl_name,
```

```
2369                                    conv_sym_shndx(osabi, mach, sym->st_shndx,
2370                                    CONV_FMT_DECIMAL, &inv_buf));
2371                                sdp->sd_isc = NULL;
2372                                sdp->sd_flags |= FLG_SY_INVALID;
2373                                continue;
2374                        }

2376                        /*
2377                         * As these local symbols will become part of the output
2378                         * image, record their number and name string size.
2379                         * Globals are counted after all input file processing
2380                         * (and hence symbol resolution) is complete during
2381                         * sym_validate().
2382                         */
2383                        if (!(ofl->ofl_flags & FLG_OF_REDLSYM) &&
2384                            symtab_enter) {
2385                                ofl->ofl_locscnt++;

2387                                if ((((sdp->sd_flags & FLG_SY_REGSYM) == 0) ||
2388                                    sym->st_name) && (st_insert(ofl->ofl_strtab,
2389                                    sdp->sd_name) == -1))
2390                                        return (S_ERROR);

2392                                if (allow_ldynsym && sym->st_name &&
2393                                    ldynsym_symtype[type]) {
2394                                        ofl->ofl_dynlocscnt++;
2395                                        if (st_insert(ofl->ofl_dynstrtab,
2396                                            sdp->sd_name) == -1)
2397                                                return (S_ERROR);
2398                                        /* Include it in sort section? */
2399                                        DYNSORT_COUNT(sdp, sym, type, ++);
2400                                }
2401                        }
2402                }
2403        }

2405        /*
2406         * The GNU ld interprets the top bit of the 16-bit Versym value
2407         * (0x8000) as the "hidden" bit. If this bit is set, the linker
2408         * is supposed to act as if that symbol does not exist. The Solaris
2409         * linker does not support this mechanism, or the model of interface
2410         * evolution that it allows, but we honor it in GNU ld produced
2411         * objects in order to interoperate with them.
2412         *
2413         * Determine if we should honor the GNU hidden bit for this file.
2414         */
2415        test_gnu_hidden_bit = ((ifl->ifl_flags & FLG_IF_GNUVER) != 0) &&
2416            (ifl->ifl_versym != NULL);

2418        /*
2419         * Determine whether object capabilities for this file are being
2420         * converted into symbol capabilities.  If so, global function symbols,
2421         * and initialized global data symbols, need special translation and
2422         * processing.
2423         */
2424        if ((etype == ET_REL) && (ifl->ifl_flags & FLG_IF_OTOSCAP))
2425                cdp = ifl->ifl_caps;

2427        /*
2428         * Now scan the global symbols entering them in the internal symbol
2429         * table or resolving them as necessary.
2430         */
2431        sym = (Sym *)isc->is_indata->d_buf;
2432        sym += local;
2433        weak = 0;
2434        /* LINTED */
```



```
2435        for (ndx = (int)local; ndx < total; sym++, ndx++) {
2436                const char      *name;
2437                sd_flag_t       sdflags = 0;
2438                Word            shndx;
2439                int             shndx_bad = 0;
2440                Sym             *nsym = sym;
2441                Cap_pair        *cpp = NULL;
2442                uchar_t         ntype;

2444                /*
2445                 * Determine and validate the associated section index.
2446                 */
2447                if (symshndx && (nsym->st_shndx == SHN_XINDEX)) {
2448                        shndx = symshndx[ndx];
2449                } else if ((shndx = nsym->st_shndx) >= SHN_LORESERVE) {
2450                        sdflags |= FLG_SY_SPECSEC;
2451                } else if (shndx > ifl->ifl_shnum) {
2452                        /* We need the name before we can issue error */
2453                        shndx_bad = 1;
2454                }

2456                /*
2457                 * Check if st_name has a valid value or not.
2458                 */
2459                if ((name = string(ofl, ifl, nsym, strs, strsize, ndx, shndx,
2460                    symsecndx, symsecname, strsecname, &sdflags)) == NULL)
2461                        continue;

2463                /*
2464                 * Now that we have the name, report an erroneous section index.
2465                 */
2466                if (shndx_bad) {
2467                        ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_INVSHNDX),
2468                            demangle_symname(name, symsecname, ndx),
2469                            ifl->ifl_name,
2470                            conv_sym_shndx(osabi, mach, nsym->st_shndx,
2471                            CONV_FMT_DECIMAL, &inv_buf));
2472                        continue;
2473                }

2475                /*
2476                 * Test for the GNU hidden bit, and ignore symbols that
2477                 * have it set.
2478                 */
2479                if (test_gnu_hidden_bit &&
2480                    ((ifl->ifl_versym[ndx] & 0x8000) != 0))
2481                        continue;

2483                /*
2484                 * The linker itself will generate symbols for _end, _etext,
2485                 * _edata, _DYNAMIC and _PROCEDURE_LINKAGE_TABLE_, so don't
2486                 * bother entering these symbols from shared objects.  This
2487                 * results in some wasted resolution processing, which is hard
2488                 * to feel, but if nothing else, pollutes diagnostic relocation
2489                 * output.
2490                 */
2491                if (name[0] && (etype == ET_DYN) && (nsym->st_size == 0) &&
2492                    (ELF_ST_TYPE(nsym->st_info) == STT_OBJECT) &&
2493                    (name[0] == '_') && ((name[1] == 'e') ||
2494                    (name[1] == 'D') || (name[1] == 'P')) &&
2495                    ((strcmp(name, MSG_ORIG(MSG_SYM_ETEXT_U)) == 0) ||
2496                    (strcmp(name, MSG_ORIG(MSG_SYM_EDATA_U)) == 0) ||
2497                    (strcmp(name, MSG_ORIG(MSG_SYM_END_U)) == 0) ||
2498                    (strcmp(name, MSG_ORIG(MSG_SYM_DYNAMIC_U)) == 0) ||
2499                    (strcmp(name, MSG_ORIG(MSG_SYM_PLKTBL_U)) == 0))) {
2500                        ifl->ifl_oldndx[ndx] = 0;
```

```
2501                    continue;
2502            }
2503
2504            /*
2505             * The '-z wrap=XXX' option emulates the GNU ld --wrap=XXX
2506             * option. When XXX is the symbol to be wrapped:
2507             *
2508             *  -   An undefined reference to XXX is converted to __wrap_XXX
2509             *  -   An undefined reference to __real_XXX is converted to XXX
2510             *
2511             * The idea is that the user can supply a wrapper function
2512             * __wrap_XXX that does some work, and then uses the name
2513             * __real_XXX to pass the call on to the real function. The
2514             * wrapper objects are linked with the original unmodified
2515             * objects to produce a wrapped version of the output object.
2516             */
2517            if (ofl->ofl_wrap && name[0] && (shndx == SHN_UNDEF)) {
2518                    WrapSymNode wsn, *wsnp;
2519
2520                    /*
2521                     * If this is the __real_XXX form, advance the
2522                     * pointer to reference the wrapped name.
2523                     */
2524                    wsn.wsn_name = name;
2525                    if ((*name == '_') &&
2526                        (strncmp(name, MSG_ORIG(MSG_STR_UU_REAL_U),
2527                        MSG_STR_UU_REAL_U_SIZE) == 0))
2528                            wsn.wsn_name += MSG_STR_UU_REAL_U_SIZE;
2529
2530                    /*
2531                     * Is this symbol in the wrap AVL tree? If so, map
2532                     * XXX to __wrap_XXX, and __real_XXX to XXX. Note that
2533                     * wsn.wsn_name will equal the current value of name
2534                     * if the __real_ prefix is not present.
2535                     */
2536                    if ((wsnp = avl_find(ofl->ofl_wrap, &wsn, 0)) != NULL) {
2537                            const char *old_name = name;
2538
2539                            name = (wsn.wsn_name == name) ?
2540                                wsnp->wsn_wrapname : wsn.wsn_name;
2541                            DBG_CALL(Dbg_syms_wrap(ofl->ofl_lml, ndx,
2542                                old_name, name));
2543                    }
2544            }
2545
2546            /*
2547             * Determine and validate the symbols binding.
2548             */
2549            bind = ELF_ST_BIND(nsym->st_info);
2550            if ((bind != STB_GLOBAL) && (bind != STB_WEAK)) {
2551                    ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_NONGLOB),
2552                        demangle_symname(name, symsecname, ndx),
2553                        ifl->ifl_name,
2554                        conv_sym_info_bind(bind, 0, &inv_buf));
2555                    continue;
2556            }
2557            if (bind == STB_WEAK)
2558                    weak++;
2559
2560            /*
2561             * If this symbol falls within the range of a section being
2562             * discarded, then discard the symbol itself.
2563             */
2564            if (((sdflags & FLG_SY_SPECSEC) == 0) &&
2565                (nsym->st_shndx != SHN_UNDEF)) {
2566                    Is_desc *isp;
```



```
2568                    if (shndx >= ifl->ifl_shnum) {
2569                            /*
2570                             * Carry our some basic sanity checks
2571                             * The symbol will not be carried forward to
2572                             * the output file, which won't be a problem
2573                             * unless a relocation is required against it.
2574                             */
2575                            ld_eprintf(ofl, ERR_WARNING,
2576                                MSG_INTL(MSG_SYM_INVSHNDX),
2577                                demangle_symname(name, symsecname, ndx),
2578                                ifl->ifl_name,
2579                                conv_sym_shndx(osabi, mach, nsym->st_shndx,
2580                                CONV_FMT_DECIMAL, &inv_buf));
2581                            continue;
2582                    }
2583
2584                    isp = ifl->ifl_isdesc[shndx];
2585                    if (isp && (isp->is_flags & FLG_IS_DISCARD)) {
2586                            if ((sdp =
2587                                libld_calloc(sizeof (Sym_desc), 1)) == NULL)
2588                                    return (S_ERROR);
2589
2590                            /*
2591                             * Create a dummy symbol entry so that if we
2592                             * find any references to this discarded symbol
2593                             * we can compensate.
2594                             */
2595                            sdp->sd_name = name;
2596                            sdp->sd_sym = nsym;
2597                            sdp->sd_file = ifl;
2598                            sdp->sd_isc = isp;
2599                            sdp->sd_flags = FLG_SY_ISDISC;
2600                            ifl->ifl_oldndx[ndx] = sdp;
2601
2602                            DBG_CALL(Dbg_syms_discarded(ofl->ofl_lml, sdp));
2603                            continue;
2604                    }
2605            }
2606
2607            /*
2608             * If object capabilities for this file are being converted
2609             * into symbol capabilities, then:
2610             *
2611             *  -   Any global function, or initialized global data symbol
2612             *      definitions (ie., those that are not associated with
2613             *      special symbol types, ie., ABS, COMMON, etc.), and which
2614             *      have not been reduced to locals, are converted to symbol
2615             *      references (UNDEF).  This ensures that any reference to
2616             *      the original symbol, for example from a relocation, get
2617             *      associated to a capabilities family lead symbol, ie., a
2618             *      generic instance.
2619             *
2620             *  -   For each global function, or object symbol definition,
2621             *      a new local symbol is created.  The function or object
2622             *      is renamed using the capabilities CA_SUNW_ID definition
2623             *      (which might have been fabricated for this purpose -
2624             *      see get_cap_group()).  The new symbol name is:
2625             *
2626             *          <original name>%<capability group identifier>
2627             *
2628             *      This symbol is associated to the same location, and
2629             *      becomes a capabilities family member.
2630             */
2631            /* LINTED */
2632            hash = (Word)elf_hash(name);
```

```
2634                ntype = ELF_ST_TYPE(nsym->st_info);
2635                if (cdp && (nsym->st_shndx != SHN_UNDEF) &&
2636                    ((sdflags & FLG_SY_SPECSEC) == 0) &&
2637                    ((ntype == STT_FUNC) || (ntype == STT_OBJECT))) {
2638                        /*
2639                         * Determine this symbol's visibility.  If a mapfile has
2640                         * indicated this symbol should be local, then there's
2641                         * no point in transforming this global symbol to a
2642                         * capabilities symbol.  Otherwise, create a symbol
2643                         * capability pair descriptor to record this symbol as
2644                         * a candidate for translation.
2645                         */
2646                        if (sym_cap_vis(name, hash, sym, ofl) &&
2647                            ((cpp = alist_append(&cappairs, NULL,
2648                            sizeof (Cap_pair), AL_CNT_CAP_PAIRS)) == NULL))
2649                                return (S_ERROR);
2650                }

2652                if (cpp) {
2653                        Sym     *rsym;

2655                        DBG_CALL(Dbg_syms_cap_convert(ofl, ndx, name, nsym));

2657                        /*
2658                         * Allocate a new symbol descriptor to represent the
2659                         * transformed global symbol.  The descriptor points
2660                         * to the original symbol information (which might
2661                         * indicate a global or weak visibility).  The symbol
2662                         * information will be transformed into a local symbol
2663                         * later, after any weak aliases are culled.
2664                         */
2665                        if ((cpp->c_osdp =
2666                            libld_malloc(sizeof (Sym_desc))) == NULL)
2667                                return (S_ERROR);

2669                        cpp->c_osdp->sd_name = name;
2670                        cpp->c_osdp->sd_sym = nsym;
2671                        cpp->c_osdp->sd_shndx = shndx;
2672                        cpp->c_osdp->sd_file = ifl;
2673                        cpp->c_osdp->sd_isc = ifl->ifl_isdesc[shndx];
2674                        cpp->c_osdp->sd_ref = REF_REL_NEED;

2676                        /*
2677                         * Save the capabilities group this symbol belongs to,
2678                         * and the original symbol index.
2679                         */
2680                        cpp->c_group = cdp->ca_groups->apl_data[0];
2681                        cpp->c_ndx = ndx;

2683                        /*
2684                         * Replace the original symbol definition with a symbol
2685                         * reference.  Make sure this reference isn't left as a
2686                         * weak.
2687                         */
2688                        if ((rsym = libld_malloc(sizeof (Sym))) == NULL)
2689                                return (S_ERROR);

2691                        *rsym = *nsym;

2693                        rsym->st_info = ELF_ST_INFO(STB_GLOBAL, ntype);
2694                        rsym->st_shndx = shndx = SHN_UNDEF;
2695                        rsym->st_value = 0;
2696                        rsym->st_size = 0;

2698                        sdflags |= FLG_SY_CAP;
```



```
2700                        nsym = rsym;
2701                }

2703                /*
2704                 * If the symbol does not already exist in the internal symbol
2705                 * table add it, otherwise resolve the conflict.  If the symbol
2706                 * from this file is kept, retain its symbol table index for
2707                 * possible use in associating a global alias.
2708                 */
2709                if ((sdp = ld_sym_find(name, hash, &where, ofl)) == NULL) {
2710                        DBG_CALL(Dbg_syms_global(ofl->ofl_lml, ndx, name));
2711                        if ((sdp = ld_sym_enter(name, nsym, hash, ifl, ofl, ndx,
2712                            shndx, sdflags, &where)) == (Sym_desc *)S_ERROR)
2713                                return (S_ERROR);

2715                } else if (ld_sym_resolve(sdp, nsym, ifl, ofl, ndx, shndx,
2716                    sdflags) == S_ERROR)
2717                        return (S_ERROR);

2719                /*
2720                 * Now that we have a symbol descriptor, retain the descriptor
2721                 * for later use by symbol capabilities processing.
2722                 */
2723                if (cpp)
2724                        cpp->c_nsdp = sdp;

2726                /*
2727                 * After we've compared a defined symbol in one shared
2728                 * object, flag the symbol so we don't compare it again.
2729                 */
2730                if ((etype == ET_DYN) && (nsym->st_shndx != SHN_UNDEF) &&
2731                    ((sdp->sd_flags & FLG_SY_SOFOUND) == 0))
2732                        sdp->sd_flags |= FLG_SY_SOFOUND;

2734                /*
2735                 * If the symbol is accepted from this file retain the symbol
2736                 * index for possible use in aliasing.
2737                 */
2738                if (sdp->sd_file == ifl)
2739                        sdp->sd_symndx = ndx;

2741                ifl->ifl_oldndx[ndx] = sdp;

2743                /*
2744                 * If we've accepted a register symbol, continue to validate
2745                 * it.
2746                 */
2747                if (sdp->sd_flags & FLG_SY_REGSYM) {
2748                        Sym_desc        *rsdp;

2750                        /*
2751                         * The presence of FLG_SY_REGSYM means that
2752                         * the pointers in ld_targ.t_ms are non-NULL.
2753                         */
2754                        rsdp = (*ld_targ.t_ms.ms_reg_find)(sdp->sd_sym, ofl);
2755                        if (rsdp == NULL) {
2756                                if ((*ld_targ.t_ms.ms_reg_enter)(sdp, ofl) == 0)
2757                                        return (S_ERROR);
2758                        } else if (rsdp != sdp) {
2759                                (void) (*ld_targ.t_ms.ms_reg_check)(rsdp,
2760                                    sdp->sd_sym, sdp->sd_name, ifl, ofl);
2761                        }
2762                }

2764                /*
```

```
2765                 * For a relocatable object, if this symbol is defined
2766                 * and has non-zero length and references an address
2767                 * within an associated section, then check its extents
2768                 * to make sure the section boundaries encompass it.
2769                 * If they don't, the ELF file is corrupt. Note that this
2770                 * global symbol may have come from another file to satisfy
2771                 * an UNDEF symbol of the same name from this one. In that
2772                 * case, we don't check it, because it was already checked
2773                 * as part of its own file.
2774                 */
2775                if (etype_rel && (sdp->sd_file == ifl)) {
2776                        Sym *tsym = sdp->sd_sym;

2778                        if (SYM_LOC_BADADDR(sdp, tsym,
2779                            ELF_ST_TYPE(tsym->st_info))) {
2780                                issue_badaddr_msg(ifl, ofl, sdp,
2781                                    tsym, tsym->st_shndx);
2782                                continue;
2783                        }
2784                }
2785        }
2786        DBG_CALL(Dbg_util_nl(ofl->ofl_lml, DBG_NL_STD));

2788        /*
2789         * Associate weak (alias) symbols to their non-weak counterparts by
2790         * scanning the global symbols one more time.
2791         *
2792         * This association is needed when processing the symbols from a shared
2793         * object dependency when a a weak definition satisfies a reference:
2794         *
2795         *  -   When building a dynamic executable, if a referenced symbol is a
2796         *      data item, the symbol data is copied to the executables address
2797         *      space.  In this copy-relocation case, we must also reassociate
2798         *      the alias symbol with its new location in the executable.
2799         *
2800         *  -   If the referenced symbol is a function then we may need to
2801         *      promote the symbols binding from undefined weak to undefined,
2802         *      otherwise the run-time linker will not generate the correct
2803         *      relocation error should the symbol not be found.
2804         *
2805         * Weak alias association is also required when a local dynsym table
2806         * is being created.  This table should only contain one instance of a
2807         * symbol that is associated to a given address.
2808         *
2809         * The true association between a weak/strong symbol pair is that both
2810         * symbol entries are identical, thus first we create a sorted symbol
2811         * list keyed off of the symbols section index and value.  If the symbol
2812         * belongs to the same section and has the same value, then the chances
2813         * are that the rest of the symbols data is the same.  This list is then
2814         * scanned for weak symbols, and if one is found then any strong
2815         * association will exist in the entries that follow.  Thus we just have
2816         * to scan one (typically a single alias) or more (in the uncommon
2817         * instance of multiple weak to strong associations) entries to
2818         * determine if a match exists.
2819         */
2820        if (weak && (OFL_ALLOW_LDYNSYM(ofl) || (etype == ET_DYN)) &&
2821            (total > local)) {
2822                static Sym_desc **sort;
2823                static size_t   osize = 0;
2824                size_t          nsize = (total - local) * sizeof (Sym_desc *);

2826                /*
2827                 * As we might be processing many input files, and many symbols,
2828                 * try and reuse a static sort buffer.  Note, presently we're
2829                 * playing the game of never freeing any buffers as there's a
2830                 * belief this wastes time.
```



```
2831                 */
2832                if ((osize == 0) || (nsize > osize)) {
2833                        if ((sort = libld_malloc(nsize)) == NULL)
2834                                return (S_ERROR);
2835                        osize = nsize;
2836                }
2837                (void) memcpy((void *)sort, &ifl->ifl_oldndx[local], nsize);

2839                qsort(sort, (total - local), sizeof (Sym_desc *), compare);

2841                for (ndx = 0; ndx < (total - local); ndx++) {
2842                        Sym_desc        *wsdp = sort[ndx];
2843                        Sym             *wsym;
2844                        int             sndx;

2846                        /*
2847                         * Ignore any empty symbol descriptor, or the case where
2848                         * the symbol has been resolved to a different file.
2849                         */
2850                        if ((wsdp == NULL) || (wsdp->sd_file != ifl))
2851                                continue;

2853                        wsym = wsdp->sd_sym;

2855                        if ((wsym->st_shndx == SHN_UNDEF) ||
2856                            (wsdp->sd_flags & FLG_SY_SPECSEC) ||
2857                            (ELF_ST_BIND(wsym->st_info) != STB_WEAK))
2858                                continue;

2860                        /*
2861                         * We have a weak symbol, if it has a strong alias it
2862                         * will have been sorted to one of the following sort
2863                         * table entries.  Note that we could have multiple weak
2864                         * symbols aliased to one strong (if this occurs then
2865                         * the strong symbol only maintains one alias back to
2866                         * the last weak).
2867                         */
2868                        for (sndx = ndx + 1; sndx < (total - local); sndx++) {
2869                                Sym_desc        *ssdp = sort[sndx];
2870                                Sym             *ssym;
2871                                sd_flag_t       w_dynbits, s_dynbits;

2873                                /*
2874                                 * Ignore any empty symbol descriptor, or the
2875                                 * case where the symbol has been resolved to a
2876                                 * different file.
2877                                 */
2878                                if ((ssdp == NULL) || (ssdp->sd_file != ifl))
2879                                        continue;

2881                                ssym = ssdp->sd_sym;

2883                                if (ssym->st_shndx == SHN_UNDEF)
2884                                        continue;

2886                                if ((ssym->st_shndx != wsym->st_shndx) ||
2887                                    (ssym->st_value != wsym->st_value))
2888                                        break;

2890                                if ((ssym->st_size != wsym->st_size) ||
2891                                    (ssdp->sd_flags & FLG_SY_SPECSEC) ||
2892                                    (ELF_ST_BIND(ssym->st_info) == STB_WEAK))
2893                                        continue;

2895                                /*
2896                                 * If a sharable object, set link fields so
```

```
2897                                * that they reference each other.‘
2898                                */
2899                               if (etype == ET_DYN) {
2900                                       ssdp->sd_aux->sa_linkndx =
2901                                           (Word)wsdp->sd_symndx;
2902                                       wsdp->sd_aux->sa_linkndx =
2903                                           (Word)ssdp->sd_symndx;
2904                               }

2906                               /*
2907                                * Determine which of these two symbols go into
2908                                * the sort section.  If a mapfile has made
2909                                * explicit settings of the FLG_SY_*DYNSORT
2910                                * flags for both symbols, then we do what they
2911                                * say.  If one has the DYNSORT flags set, we
2912                                * set the NODYNSORT bit in the other.  And if
2913                                * neither has an explicit setting, then we
2914                                * favor the weak symbol because they usually
2915                                * lack the leading underscore.
2916                                */
2917                               w_dynbits = wsdp->sd_flags &
2918                                   (FLG_SY_DYNSORT | FLG_SY_NODYNSORT);
2919                               s_dynbits = ssdp->sd_flags &
2920                                   (FLG_SY_DYNSORT | FLG_SY_NODYNSORT);
2921                               if (!(w_dynbits && s_dynbits)) {
2922                                       if (s_dynbits) {
2923                                               if (s_dynbits == FLG_SY_DYNSORT)
2924                                                       wsdp->sd_flags |=
2925                                                           FLG_SY_NODYNSORT;
2926                                       } else if (w_dynbits !=
2927                                           FLG_SY_NODYNSORT) {
2928                                               ssdp->sd_flags |=
2929                                                   FLG_SY_NODYNSORT;
2930                                       }
2931                               }
2932                               break;
2933                       }
2934               }
2935       }

2937       /*
2938        * Having processed all symbols, under -z symbolcap, reprocess any
2939        * symbols that are being translated from global to locals.  The symbol
2940        * pair that has been collected defines the original symbol (c_osdp),
2941        * which will become a local, and the new symbol (c_nsdp), which will
2942        * become a reference (UNDEF) for the original.
2943        *
2944        * Scan these symbol pairs looking for weak symbols, which have non-weak
2945        * aliases.  There is no need to translate both of these symbols to
2946        * locals, only the global is necessary.
2947        */
2948       if (cappairs) {
2949               Aliste          idx1;
2950               Cap_pair        *cpp1;

2952               for (ALIST_TRAVERSE(cappairs, idx1, cpp1)) {
2953                       Sym_desc        *sdp1 = cpp1->c_osdp;
2954                       Sym             *sym1 = sdp1->sd_sym;
2955                       uchar_t         bind1 = ELF_ST_BIND(sym1->st_info);
2956                       Aliste          idx2;
2957                       Cap_pair        *cpp2;

2959                       /*
2960                        * If this symbol isn’t weak, it’s capability member is
2961                        * retained for the creation of a local symbol.
2962                        */
```



```
2963                       if (bind1 != STB_WEAK)
2964                               continue;

2966                       /*
2967                        * If this is a weak symbol, traverse the capabilities
2968                        * list again to determine if a corresponding non-weak
2969                        * symbol exists.
2970                        */
2971                       for (ALIST_TRAVERSE(cappairs, idx2, cpp2)) {
2972                               Sym_desc        *sdp2 = cpp2->c_osdp;
2973                               Sym             *sym2 = sdp2->sd_sym;
2974                               uchar_t         bind2 =
2975                                   ELF_ST_BIND(sym2->st_info);

2977                               if ((cpp1 == cpp2) ||
2978                                   (cpp1->c_group != cpp2->c_group) ||
2979                                   (sym1->st_value != sym2->st_value) ||
2980                                   (bind2 == STB_WEAK))
2981                                       continue;

2983                               /*
2984                                * The weak symbol (sym1) has a non-weak (sym2)
2985                                * counterpart.  There’s no point in translating
2986                                * both of these equivalent symbols to locals.
2987                                * Add this symbol capability alias to the
2988                                * capabilities family information, and remove
2989                                * the weak symbol.
2990                                */
2991                               if (ld_cap_add_family(ofl, cpp2->c_nsdp,
2992                                   cpp1->c_nsdp, NULL, NULL) == S_ERROR)
2993                                       return (S_ERROR);

2995                               free((void *)cpp1->c_osdp);
2996                               (void) alist_delete(cappairs, &idx1);
2997                       }
2998               }

3000               DBG_CALL(Dbg_util_nl(ofl->ofl_lml, DBG_NL_STD));

3002               /*
3003                * The capability pairs information now represents all the
3004                * global symbols that need transforming to locals.  These
3005                * local symbols are renamed using their group identifiers.
3006                */
3007               for (ALIST_TRAVERSE(cappairs, idx1, cpp1)) {
3008                       Sym_desc        *osdp = cpp1->c_osdp;
3009                       Objcapset       *capset;
3010                       size_t          nsize, tsize;
3011                       const char      *oname;
3012                       char            *cname, *idstr;
3013                       Sym             *csym;

3015                       /*
3016                        * If the local symbol has not yet been translated
3017                        * convert it to a local symbol with a name.
3018                        */
3019                       if ((osdp->sd_flags & FLG_SY_CAP) != 0)
3020                               continue;

3022                       /*
3023                        * As we’re converting object capabilities to symbol
3024                        * capabilities, obtain the capabilities set for this
3025                        * object, so as to retrieve the CA_SUNW_ID value.
3026                        */
3027                       capset = &cpp1->c_group->cg_set;
```

```
3029                        /*
3030                         * Create a new name from the existing symbol and the
3031                         * capabilities group identifier.  Note, the delimiter
3032                         * between the symbol name and identifier name is hard-
3033                         * coded here (%), so that we establish a convention
3034                         * for transformed symbol names.
3035                         */
3036                        oname = osdp->sd_name;

3038                        idstr = capset->oc_id.cs_str;
3039                        nsize = strlen(oname);
3040                        tsize = nsize + 1 + strlen(idstr) + 1;
3041                        if ((cname = libld_malloc(tsize)) == 0)
3042                                return (S_ERROR);

3044                        (void) strcpy(cname, oname);
3045                        cname[nsize++] = '%';
3046                        (void) strcpy(&cname[nsize], idstr);

3048                        /*
3049                         * Allocate a new symbol table entry, transform this
3050                         * symbol to a local, and assign the new name.
3051                         */
3052                        if ((csym = libld_malloc(sizeof (Sym))) == NULL)
3053                                return (S_ERROR);

3055                        *csym = *osdp->sd_sym;
3056                        csym->st_info = ELF_ST_INFO(STB_LOCAL,
3057                            ELF_ST_TYPE(osdp->sd_sym->st_info));

3059                        osdp->sd_name = cname;
3060                        osdp->sd_sym = csym;
3061                        osdp->sd_flags = FLG_SY_CAP;

3063                        /*
3064                         * Keep track of this new local symbol.  As -z symbolcap
3065                         * can only be used to create a relocatable object, a
3066                         * dynamic symbol table can't exist.  Ensure there is
3067                         * space reserved in the string table.
3068                         */
3069                        ofl->ofl_caploclcnt++;
3070                        if (st_insert(ofl->ofl_strtab, cname) == -1)
3071                                return (S_ERROR);

3073                        DBG_CALL(Dbg_syms_cap_local(ofl, cpp1->c_ndx,
3074                            cname, csym, osdp));

3076                        /*
3077                         * Establish this capability pair as a family.
3078                         */
3079                        if (ld_cap_add_family(ofl, cpp1->c_nsdp, osdp,
3080                            cpp1->c_group, &ifl->ifl_caps->ca_syms) == S_ERROR)
3081                                return (S_ERROR);
3082                }
3083        }

3085        return (1);

3087 #undef SYM_LOC_BADADDR
3088 }

3090 /*
3091  * Add an undefined symbol to the symbol table.  The reference originates from
3092  * the location identified by the message id (mid).  These references can
3093  * originate from command line options such as -e, -u, -initarray, etc.
3094  * (identified with MSG_INTL(MSG_STR_COMMAND)), or from internally generated
```



```
3095  * TLS relocation references (identified with MSG_INTL(MSG_STR_TLSREL)).
3096  */
3097 Sym_desc *
3098 ld_sym_add_u(const char *name, Ofl_desc *ofl, Msg mid)
3099 {
3100        Sym             *sym;
3101        Ifl_desc        *ifl = NULL, *_ifl;
3102        Sym_desc        *sdp;
3103        Word            hash;
3104        Aliste          idx;
3105        avl_index_t     where;
3106        const char      *reference = MSG_INTL(mid);

3108        /*
3109         * As an optimization, determine whether we've already generated this
3110         * reference.  If the symbol doesn't already exist we'll create it.
3111         * Or if the symbol does exist from a different source, we'll resolve
3112         * the conflict.
3113         */
3114        /* LINTED */
3115        hash = (Word)elf_hash(name);
3116        if ((sdp = ld_sym_find(name, hash, &where, ofl)) != NULL) {
3117                if ((sdp->sd_sym->st_shndx == SHN_UNDEF) &&
3118                    (sdp->sd_file->ifl_name == reference))
3119                        return (sdp);
3120        }

3122        /*
3123         * Determine whether a pseudo input file descriptor exists to represent
3124         * the command line, as any global symbol needs an input file descriptor
3125         * during any symbol resolution (refer to map_ifl() which provides a
3126         * similar method for adding symbols from mapfiles).
3127         */
3128        for (APLIST_TRAVERSE(ofl->ofl_objs, idx, _ifl))
3129                if (strcmp(_ifl->ifl_name, reference) == 0) {
3130                        ifl = _ifl;
3131                        break;
3132                }

3134        /*
3135         * If no descriptor exists create one.
3136         */
3137        if (ifl == NULL) {
3138                if ((ifl = libld_calloc(sizeof (Ifl_desc), 1)) == NULL)
3139                        return ((Sym_desc *)S_ERROR);
3140                ifl->ifl_name = reference;
3141                ifl->ifl_flags = FLG_IF_NEEDED | FLG_IF_FILEREF;
3142                if ((ifl->ifl_ehdr = libld_calloc(sizeof (Ehdr), 1)) == NULL)
3143                        return ((Sym_desc *)S_ERROR);
3144                ifl->ifl_ehdr->e_type = ET_REL;

3146                if (aplist_append(&ofl->ofl_objs, ifl, AL_CNT_OFL_OBJS) == NULL)
3147                        return ((Sym_desc *)S_ERROR);
3148        }

3150        /*
3151         * Allocate a symbol structure and add it to the global symbol table.
3152         */
3153        if ((sym = libld_calloc(sizeof (Sym), 1)) == NULL)
3154                return ((Sym_desc *)S_ERROR);
3155        sym->st_info = ELF_ST_INFO(STB_GLOBAL, STT_NOTYPE);
3156        sym->st_shndx = SHN_UNDEF;

3158        DBG_CALL(Dbg_syms_process(ofl->ofl_lml, ifl));
3159        if (sdp == NULL) {
3160                DBG_CALL(Dbg_syms_global(ofl->ofl_lml, 0, name));
```

```
3161                 if ((sdp = ld_sym_enter(name, sym, hash, ifl, ofl, 0, SHN_UNDEF,
3162                     0, &where)) == (Sym_desc *)S_ERROR)
3163                         return ((Sym_desc *)S_ERROR);
3164         } else if (ld_sym_resolve(sdp, sym, ifl, ofl, 0,
3165             SHN_UNDEF, 0) == S_ERROR)
3166                 return ((Sym_desc *)S_ERROR);

3168         sdp->sd_flags &= ~FLG_SY_CLEAN;
3169         sdp->sd_flags |= FLG_SY_CMDREF;

3171         return (sdp);
3172 }

3174 /*
3175  * STT_SECTION symbols have their st_name field set to NULL, and consequently
3176  * have no name. Generate a name suitable for diagnostic use for such a symbol
3177  * and store it in the input section descriptor. The resulting name will be
3178  * of the form:
3179  *
3180  *      "XXX (section)"
3181  *
3182  * where XXX is the name of the section.
3183  *
3184  * entry:
3185  *      isc - Input section associated with the symbol.
3186  *      fmt - NULL, or format string to use.
3187  *
3188  * exit:
3189  *      Sets isp->is_sym_name to the allocated string. Returns the
3190  *      string pointer, or NULL on allocation failure.
3191  */
3192 const char *
3193 ld_stt_section_sym_name(Is_desc *isp)
3194 {
3195         const char      *fmt;
3196         char            *str;
3197         size_t          len;

3199         if ((isp == NULL) || (isp->is_name == NULL))
3200                 return (NULL);

3202         if (isp->is_sym_name == NULL) {
3203                 fmt = (isp->is_flags & FLG_IS_GNSTRMRG) ?
3204                     MSG_INTL(MSG_STR_SECTION_MSTR) : MSG_INTL(MSG_STR_SECTION);

3206                 len = strlen(fmt) + strlen(isp->is_name) + 1;

3208                 if ((str = libld_malloc(len)) == NULL)
3209                         return (NULL);
3210                 (void) snprintf(str, len, fmt, isp->is_name);
3211                 isp->is_sym_name = str;
3212         }

3214         return (isp->is_sym_name);
3215 }
```

```
**********************************************************
  118766 Mon Feb 11 00:23:20 2019
new/usr/src/cmd/sgs/libld/common/update.c
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
______unchanged_portion_omitted_

 132 /*
 133  * Build and update any output symbol tables.  Here we work on all the symbol
 134  * tables at once to reduce the duplication of symbol and string manipulation.
 135  * Symbols and their associated strings are copied from the read-only input
 136  * file images to the output image and their values and index's updated in the
 137  * output image.
 138  */
 139 static Addr
 140 update_osym(Ofl_desc *ofl)
 141 {
 142         /*
 143          * There are several places in this function where we wish
 144          * to insert a symbol index to the combined .SUNW_ldynsym/.dynsym
 145          * symbol table into one of the two sort sections (.SUNW_dynsymsort
 146          * or .SUNW_dyntlssort), if that symbol has the right attributes.
 147          * This macro is used to generate the necessary code from a single
 148          * specification.
 149          *
 150          * entry:
 151          *      _sdp, _sym, _type - As per DYNSORT_COUNT. See _libld.h
 152          *      _sym_ndx - Index that _sym will have in the combined
 153          *              .SUNW_ldynsym/.dynsym symbol table.
 154          */
 155 #define ADD_TO_DYNSORT(_sdp, _sym, _type, _sym_ndx) \
 156         { \
 157                 Word *_dynsort_arr, *_dynsort_ndx; \
 158                 \
 159                 if (dynsymsort_symtype[_type]) { \
 160                         _dynsort_arr = dynsymsort; \
 161                         _dynsort_ndx = &dynsymsort_ndx; \
 162                 } else if (_type == STT_TLS) { \
 163                         _dynsort_arr = dyntlssort; \
 164                         _dynsort_ndx = &dyntlssort_ndx; \
 165                 } else { \
 166                         _dynsort_arr = NULL; \
 167                 } \
 168                 if ((_dynsort_arr != NULL) && DYNSORT_TEST_ATTR(_sdp, _sym)) \
 169                     _dynsort_arr[(*_dynsort_ndx)++] = _sym_ndx; \
 170         }

 172         Sym_desc        *sdp;
 173         Sym_avlnode     *sav;
 174         Sg_desc         *sgp, *tsgp = NULL, *dsgp = NULL, *esgp = NULL;
 175         Os_desc         *osp, *iosp = NULL, *fosp = NULL;
 176         Is_desc         *isc;
 177         Ifl_desc        *ifl;
 178         Word            bssndx, etext_ndx, edata_ndx = 0, end_ndx, start_ndx;
 179         Word            end_abs = 0, etext_abs = 0, edata_abs;
 180         Word            tlsbssndx = 0, parexpnndx;
 181 #if     defined(_ELF64)
 182         Word            lbssndx = 0;
 183         Addr            lbssaddr = 0;
 184 #endif
 185         Addr            bssaddr, etext = 0, edata = 0, end = 0, start = 0;
 186         Addr            tlsbssaddr = 0;
 187         Addr            parexpnbase, parexpnaddr;
 188         int             start_set = 0;
```



```
 189         Sym             _sym = {0}, *sym, *symtab = NULL;
 190         Sym             *dynsym = NULL, *ldynsym = NULL;
 191         Word            symtab_ndx = 0;         /* index into .symtab */
 192         Word            symtab_gbl_bndx;        /* .symtab ndx 1st global */
 193         Word            ldynsym_ndx = 0;        /* index into .SUNW_ldynsym */
 194         Word            dynsym_ndx = 0;         /* index into .dynsym */
 195         Word            scopesym_ndx = 0;       /* index into scoped symbols */
 196         Word            scopesym_bndx = 0;      /* .symtab ndx 1st scoped sym */
 197         Word            ldynscopesym_ndx = 0;   /* index to ldynsym scoped */
 198                                                 /*      symbols */
 199         Word            *dynsymsort = NULL;     /* SUNW_dynsymsort index */
 200                                                 /*      vector */
 201         Word            *dyntlssort = NULL;     /* SUNW_dyntlssort index */
 202                                                 /*      vector */
 203         Word            dynsymsort_ndx;         /* index dynsymsort array */
 204         Word            dyntlssort_ndx;         /* index dyntlssort array */
 205         Word            *symndx;                /* symbol index (for */
 206                                                 /*      relocation use) */
 207         Word            *symshndx = NULL;       /* .symtab_shndx table */
 208         Word            *dynshndx = NULL;       /* .dynsym_shndx table */
 209         Word            *ldynshndx = NULL;      /* .SUNW_ldynsym_shndx table */
 210         Word            ldynsym_cnt = NULL;     /* number of items in */
 211                                                 /*      .SUNW_ldynsym */
 212         Str_tbl         *shstrtab;
 213         Str_tbl         *strtab;
 214         Str_tbl         *dynstr;
 215         Word            *hashtab;       /* hash table pointer */
 216         Word            *hashbkt;       /* hash table bucket pointer */
 217         Word            *hashchain;     /* hash table chain pointer */
 218         Wk_desc         *wkp;
 219         Alist           *weak = NULL;
 220         ofl_flag_t      flags = ofl->ofl_flags;
 221         Versym          *versym;
 222         Gottable        *gottable;      /* used for display got debugging */
 223                                         /*      information */
 224         Syminfo         *syminfo;
 225         Sym_s_list      *sorted_syms;   /* table to hold sorted symbols */
 226         Word            ssndx;          /* global index into sorted_syms */
 227         Word            scndx;          /* scoped index into sorted_syms */
 228         size_t          stoff;          /* string offset */
 229         Aliste          idx1;

 231         /*
 232          * Initialize pointers to the symbol table entries and the symbol
 233          * table strings.  Skip the first symbol entry and the first string
 234          * table byte.  Note that if we are not generating any output symbol
 235          * tables we must still generate and update internal copies so
 236          * that the relocation phase has the correct information.
 237          */
 238         if (!(flags & FLG_OF_STRIP) || (flags & FLG_OF_RELOBJ) ||
 239             ((flags & FLG_OF_STATIC) && ofl->ofl_osversym)) {
 240                 symtab = (Sym *)ofl->ofl_ossymtab->os_outdata->d_buf;
 241                 symtab[symtab_ndx++] = _sym;
 242                 if (ofl->ofl_ossymshndx)
 243                         symshndx =
 244                             (Word *)ofl->ofl_ossymshndx->os_outdata->d_buf;
 245         }
 246         if (OFL_ALLOW_DYNSYM(ofl)) {
 247                 dynsym = (Sym *)ofl->ofl_osdynsym->os_outdata->d_buf;
 248                 dynsym[dynsym_ndx++] = _sym;
 249                 /*
 250                  * If we are also constructing a .SUNW_ldynsym section
 251                  * to contain local function symbols, then set it up too.
 252                  */
 253                 if (ofl->ofl_osldynsym) {
 254                         ldynsym = (Sym *)ofl->ofl_osldynsym->os_outdata->d_buf;
```

```
255                         ldynsym[ldynsym_ndx++] = _sym;
256                         ldynsym_cnt = 1 + ofl->ofl_dynlocscnt +
257                             ofl->ofl_dynscopecnt;

259                         /*
260                          * If there is a SUNW_ldynsym, then there may also
261                          * be a .SUNW_dynsymsort and/or .SUNW_dyntlssort
262                          * sections, used to collect indices of function
263                          * and data symbols sorted by address order.
264                          */
265                         if (ofl->ofl_osdynsymsort) {    /* .SUNW_dynsymsort */
266                                 dynsymsort = (Word *)
267                                     ofl->ofl_osdynsymsort->os_outdata->d_buf;
268                                 dynsymsort_ndx = 0;
269                         }
270                         if (ofl->ofl_osdyntlssort) {    /* .SUNW_dyntlssort */
271                                 dyntlssort = (Word *)
272                                     ofl->ofl_osdyntlssort->os_outdata->d_buf;
273                                 dyntlssort_ndx = 0;
274                         }
275                 }

277                 /*
278                  * Initialize the hash table.
279                  */
280                 hashtab = (Word *)(ofl->ofl_oshash->os_outdata->d_buf);
281                 hashbkt = &hashtab[2];
282                 hashchain = &hashtab[2 + ofl->ofl_hashbkts];
283                 hashtab[0] = ofl->ofl_hashbkts;
284                 hashtab[1] = DYNSYM_ALL_CNT(ofl);
285                 if (ofl->ofl_osdynshndx)
286                         dynshndx =
287                             (Word *)ofl->ofl_osdynshndx->os_outdata->d_buf;
288                 if (ofl->ofl_osldynshndx)
289                         ldynshndx =
290                             (Word *)ofl->ofl_osldynshndx->os_outdata->d_buf;
291         }

293         /*
294          * symndx is the symbol index to be used for relocation processing.  It
295          * points to the relevant symtab's (.dynsym or .symtab) symbol ndx.
296          */
297         if (dynsym)
298                 symndx = &dynsym_ndx;
299         else
300                 symndx = &symtab_ndx;

302         /*
303          * If we have version definitions initialize the version symbol index
304          * table.  There is one entry for each symbol which contains the symbols
305          * version index.
306          */
307         if (!(flags & FLG_OF_NOVERSEC) &&
308             (flags & (FLG_OF_VERNEED | FLG_OF_VERDEF))) {
309                 versym = (Versym *)ofl->ofl_osversym->os_outdata->d_buf;
310                 versym[0] = NULL;
311         } else
312                 versym = NULL;

314         /*
315          * If syminfo section exists be prepared to fill it in.
316          */
317         if (ofl->ofl_ossyminfo) {
318                 syminfo = ofl->ofl_ossyminfo->os_outdata->d_buf;
319                 syminfo[0].si_flags = SYMINFO_CURRENT;
320         } else
```



```
321                 syminfo = NULL;

323         /*
324          * Setup our string tables.
325          */
326         shstrtab = ofl->ofl_shdrsttab;
327         strtab = ofl->ofl_strtab;
328         dynstr = ofl->ofl_dynstrtab;

330         DBG_CALL(Dbg_syms_sec_title(ofl->ofl_lml));

332         /*
333          * Put output file name to the first .symtab and .SUNW_ldynsym symbol.
334          */
335         if (symtab) {
336                 (void) st_setstring(strtab, ofl->ofl_name, &stoff);
337                 sym = &symtab[symtab_ndx++];
338                 /* LINTED */
339                 sym->st_name = stoff;
340                 sym->st_value = 0;
341                 sym->st_size = 0;
342                 sym->st_info = ELF_ST_INFO(STB_LOCAL, STT_FILE);
343                 sym->st_other = 0;
344                 sym->st_shndx = SHN_ABS;

346                 if (versym && !dynsym)
347                         versym[1] = 0;
348         }
349         if (ldynsym) {
350                 (void) st_setstring(dynstr, ofl->ofl_name, &stoff);
351                 sym = &ldynsym[ldynsym_ndx];
352                 /* LINTED */
353                 sym->st_name = stoff;
354                 sym->st_value = 0;
355                 sym->st_size = 0;
356                 sym->st_info = ELF_ST_INFO(STB_LOCAL, STT_FILE);
357                 sym->st_other = 0;
358                 sym->st_shndx = SHN_ABS;

360                 /* Scoped symbols get filled in global loop below */
361                 ldynscopesym_ndx = ldynsym_ndx + 1;
362                 ldynsym_ndx += ofl->ofl_dynscopecnt;
363         }

365         /*
366          * If we are to display GOT summary information, then allocate
367          * the buffer to 'cache' the GOT symbols into now.
368          */
369         if (DBG_ENABLED) {
370                 if ((ofl->ofl_gottable = gottable =
371                     libld_calloc(ofl->ofl_gotcnt, sizeof (Gottable))) == NULL)
372                         return ((Addr)S_ERROR);
373         }

375         /*
376          * Traverse the program headers.  Determine the last executable segment
377          * and the last data segment so that we can update etext and edata. If
378          * we have empty segments (reservations) record them for setting _end.
379          */
380         for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
381                 Phdr    *phd = &(sgp->sg_phdr);
382                 Os_desc *osp;
383                 Aliste  idx2;

385                 if (phd->p_type == PT_LOAD) {
386                         if (sgp->sg_osdescs != NULL) {
```

```
387                                 Word    _flags = phd->p_flags & (PF_W | PF_R);
389                                 if (_flags == PF_R)
390                                         tsgp = sgp;
391                                 else if (_flags == (PF_W | PF_R))
392                                         dsgp = sgp;
393                         } else if (sgp->sg_flags & FLG_SG_EMPTY)
394                                 esgp = sgp;
395                 }
397                 /*
398                  * Generate a section symbol for each output section.
399                  */
400                 for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
401                         Word    sectndx;
403                         sym = &_sym;
404                         sym->st_value = osp->os_shdr->sh_addr;
405                         sym->st_info = ELF_ST_INFO(STB_LOCAL, STT_SECTION);
406                         /* LINTED */
407                         sectndx = elf_ndxscn(osp->os_scn);
409                         if (symtab) {
410                                 if (sectndx >= SHN_LORESERVE) {
411                                         symshndx[symtab_ndx] = sectndx;
412                                         sym->st_shndx = SHN_XINDEX;
413                                 } else {
414                                         /* LINTED */
415                                         sym->st_shndx = (Half)sectndx;
416                                 }
417                                 symtab[symtab_ndx++] = *sym;
418                         }
420                         if (dynsym && (osp->os_flags & FLG_OS_OUTREL))
421                                 dynsym[dynsym_ndx++] = *sym;
423                         if ((dynsym == NULL) ||
424                             (osp->os_flags & FLG_OS_OUTREL)) {
425                                 if (versym)
426                                         versym[*symndx - 1] = 0;
427                                 osp->os_identndx = *symndx - 1;
428                                 DBG_CALL(Dbg_syms_sec_entry(ofl->ofl_lml,
429                                     osp->os_identndx, sgp, osp));
430                         }
432                         /*
433                          * Generate the .shstrtab for this section.
434                          */
435                         (void) st_setstring(shstrtab, osp->os_name, &stoff);
436                         osp->os_shdr->sh_name = (Word)stoff;
438                         /*
439                          * Find the section index for our special symbols.
440                          */
441                         if (sgp == tsgp) {
442                                 /* LINTED */
443                                 etext_ndx = elf_ndxscn(osp->os_scn);
444                         } else if (dsgp == sgp) {
445                                 if (osp->os_shdr->sh_type != SHT_NOBITS) {
446                                         /* LINTED */
447                                         edata_ndx = elf_ndxscn(osp->os_scn);
448                                 }
449                         }
451                         if (start_set == 0) {
452                                 start = sgp->sg_phdr.p_vaddr;
```



```
453                                 /* LINTED */
454                                 start_ndx = elf_ndxscn(osp->os_scn);
455                                 start_set++;
456                         }
458                         /*
459                          * While we're here, determine whether a .init or .fini
460                          * section exist.
461                          */
462                         if ((iosp == NULL) && (strcmp(osp->os_name,
463                             MSG_ORIG(MSG_SCN_INIT)) == 0))
464                                 iosp = osp;
465                         if ((fosp == NULL) && (strcmp(osp->os_name,
466                             MSG_ORIG(MSG_SCN_FINI)) == 0))
467                                 fosp = osp;
468                 }
469         }
471         /*
472          * Add local register symbols to the .dynsym.  These are required as
473          * DT_REGISTER .dynamic entries must have a symbol to reference.
474          */
475         if (ofl->ofl_regsyms && dynsym) {
476                 int     ndx;
478                 for (ndx = 0; ndx < ofl->ofl_regsymsno; ndx++) {
479                         Sym_desc        *rsdp;
481                         if ((rsdp = ofl->ofl_regsyms[ndx]) == NULL)
482                                 continue;
484                         if (!SYM_IS_HIDDEN(rsdp) &&
485                             (ELF_ST_BIND(rsdp->sd_sym->st_info) != STB_LOCAL))
486                                 continue;
488                         dynsym[dynsym_ndx] = *(rsdp->sd_sym);
489                         rsdp->sd_symndx = *symndx;
491                         if (dynsym[dynsym_ndx].st_name) {
492                                 (void) st_setstring(dynstr, rsdp->sd_name,
493                                     &stoff);
494                                 dynsym[dynsym_ndx].st_name = stoff;
495                         }
496                         dynsym_ndx++;
497                 }
498         }
500         /*
501          * Having traversed all the output segments, warn the user if the
502          * traditional text or data segments don't exist.  Otherwise from these
503          * segments establish the values for 'etext', 'edata', 'end', 'END',
504          * and 'START'.
505          */
506         if (!(flags & FLG_OF_RELOBJ)) {
507                 Sg_desc *sgp;
509                 if (tsgp)
510                         etext = tsgp->sg_phdr.p_vaddr + tsgp->sg_phdr.p_filesz;
511                 else {
512                         etext = (Addr)0;
513                         etext_ndx = SHN_ABS;
514                         etext_abs = 1;
515                         if (flags & FLG_OF_VERBOSE)
516                                 ld_eprintf(ofl, ERR_WARNING,
517                                     MSG_INTL(MSG_UPD_NOREADSEG));
518                 }
```

```
519                 if (dsgp) {
520                         edata = dsgp->sg_phdr.p_vaddr + dsgp->sg_phdr.p_filesz;
521                 } else {
522                         edata = (Addr)0;
523                         edata_ndx = SHN_ABS;
524                         edata_abs = 1;
525                         if (flags & FLG_OF_VERBOSE)
526                                 ld_eprintf(ofl, ERR_WARNING,
527                                     MSG_INTL(MSG_UPD_NORDWRSEG));
528                 }

530                 if (dsgp == NULL) {
531                         if (tsgp)
532                                 sgp = tsgp;
533                         else
534                                 sgp = 0;
535                 } else if (tsgp == NULL)
536                         sgp = dsgp;
537                 else if (dsgp->sg_phdr.p_vaddr > tsgp->sg_phdr.p_vaddr)
538                         sgp = dsgp;
539                 else if (dsgp->sg_phdr.p_vaddr < tsgp->sg_phdr.p_vaddr)
540                         sgp = tsgp;
541                 else {
542                         /*
543                          * One of the segments must be of zero size.
544                          */
545                         if (tsgp->sg_phdr.p_memsz)
546                                 sgp = tsgp;
547                         else
548                                 sgp = dsgp;
549                 }

551                 if (esgp && (esgp->sg_phdr.p_vaddr > sgp->sg_phdr.p_vaddr))
552                         sgp = esgp;

554                 if (sgp) {
555                         end = sgp->sg_phdr.p_vaddr + sgp->sg_phdr.p_memsz;

557                         /*
558                          * If the last loadable segment is a read-only segment,
559                          * then the application which uses the symbol _end to
560                          * find the beginning of writable heap area may cause
561                          * segmentation violation. We adjust the value of the
562                          * _end to skip to the next page boundary.
563                          *
564                          * 6401812 System interface which returs beginning
565                          *         heap would be nice.
566                          * When the above RFE is implemented, the changes below
567                          * could be changed in a better way.
568                          */
569                         if ((sgp->sg_phdr.p_flags & PF_W) == 0)
570                                 end = (Addr)S_ROUND(end, sysconf(_SC_PAGESIZE));

572                         /*
573                          * If we're dealing with a memory reservation there are
574                          * no sections to establish an index for _end, so assign
575                          * it as an absolute.
576                          */
577                         if (sgp->sg_osdescs != NULL) {
578                                 /*
579                                  * Determine the last section for this segment.
580                                  */
581                                 Os_desc *osp = sgp->sg_osdescs->apl_data
582                                     [sgp->sg_osdescs->apl_nitems - 1];

584                                 /* LINTED */
```



```
585                                 end_ndx = elf_ndxscn(osp->os_scn);
586                         } else {
587                                 end_ndx = SHN_ABS;
588                                 end_abs = 1;
589                         }
590                 } else {
591                         end = (Addr) 0;
592                         end_ndx = SHN_ABS;
593                         end_abs = 1;
594                         ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_UPD_NOSEG));
595                 }
596         }

598         /*
599          * Initialize the scoped symbol table entry point.  This is for all
600          * the global symbols that have been scoped to locals and will be
601          * filled in during global symbol processing so that we don't have
602          * to traverse the globals symbol hash array more than once.
603          */
604         if (symtab) {
605                 scopesym_bndx = symtab_ndx;
606                 scopesym_ndx = scopesym_bndx;
607                 symtab_ndx += ofl->ofl_scopecnt;
608         }

610         /*
611          * If expanding partially expanded symbols under '-z nopartial',
612          * prepare to do that.
613          */
614         if (ofl->ofl_isparexpn) {
615                 osp = ofl->ofl_isparexpn->is_osdesc;
616                 parexpnbase = parexpnaddr = (Addr)(osp->os_shdr->sh_addr +
617                     ofl->ofl_isparexpn->is_indata->d_off);
618                 /* LINTED */
619                 parexpnndx = elf_ndxscn(osp->os_scn);
620                 ofl->ofl_parexpnndx = osp->os_identndx;
621         }

623         /*
624          * If we are generating a .symtab collect all the local symbols,
625          * assigning a new virtual address or displacement (value).
626          */
627         for (APLIST_TRAVERSE(ofl->ofl_objs, idx1, ifl)) {
628                 Xword           lndx, local = ifl->ifl_locscnt;
629                 Cap_desc        *cdp = ifl->ifl_caps;

631                 for (lndx = 1; lndx < local; lndx++) {
632                         Gotndx          *gnp;
633                         uchar_t         type;
634                         Word            *_symshndx;
635                         int             enter_in_symtab, enter_in_ldynsym;
636                         int             update_done;

638                         sdp = ifl->ifl_oldndx[lndx];
639                         sym = sdp->sd_sym;

641                         /*
642                          * Assign a got offset if necessary.
643                          */
644                         if ((ld_targ.t_mr.mr_assign_got != NULL) &&
645                             (*ld_targ.t_mr.mr_assign_got)(ofl, sdp) == S_ERROR)
646                                 return ((Addr)S_ERROR);

648                         if (DBG_ENABLED) {
649                                 Aliste  idx2;
```

```
651                             for (ALIST_TRAVERSE(sdp->sd_GOTndxs,
652                                 idx2, gnp)) {
653                                     gottable->gt_sym = sdp;
654                                     gottable->gt_gndx.gn_gotndx =
655                                         gnp->gn_gotndx;
656                                     gottable->gt_gndx.gn_addend =
657                                         gnp->gn_addend;
658                                     gottable++;
659                             }
660                     }

662                     if ((type = ELF_ST_TYPE(sym->st_info)) == STT_SECTION)
663                             continue;

665                     /*
666                      * Ignore any symbols that have been marked as invalid
667                      * during input processing.  Providing these aren't used
668                      * for relocation they'll just be dropped from the
669                      * output image.
670                      */
671                     if (sdp->sd_flags & FLG_SY_INVALID)
672                             continue;

674                     /*
675                      * If the section that this symbol was associated
676                      * with has been discarded - then we discard
677                      * the local symbol along with it.
678                      */
679                     if (sdp->sd_flags & FLG_SY_ISDISC)
680                             continue;

682                     /*
683                      * If this symbol is from a different file
684                      * than the input descriptor we are processing,
685                      * treat it as if it has FLG_SY_ISDISC set.
686                      * This happens when sloppy_comdat_reloc()
687                      * replaces a symbol to a discarded comdat section
688                      * with an equivalent symbol from a different
689                      * file. We only want to enter such a symbol
690                      * once --- as part of the file that actually
691                      * supplies it.
692                      */
693                     if (ifl != sdp->sd_file)
694                             continue;

696                     /*
697                      * Generate an output symbol to represent this input
698                      * symbol.  Even if the symbol table is to be stripped
699                      * we still need to update any local symbols that are
700                      * used during relocation.
701                      */
702                     enter_in_symtab = symtab &&
703                         (!(ofl->ofl_flags & FLG_OF_REDLSYM) ||
704                         sdp->sd_move);
705                     enter_in_ldynsym = ldynsym && sdp->sd_name &&
706                         ldynsym_symtype[type] &&
707                         !(ofl->ofl_flags & FLG_OF_REDLSYM);
708                     _symshndx = NULL;

710                     if (enter_in_symtab) {
711                             if (!dynsym)
712                                     sdp->sd_symndx = *symndx;
713                             symtab[symtab_ndx] = *sym;

715                             /*
716                              * Provided this isn't an unnamed register
```



```
717                              * symbol, update its name.
718                              */
719                             if (((sdp->sd_flags & FLG_SY_REGSYM) == 0) ||
720                                 symtab[symtab_ndx].st_name) {
721                                     (void) st_setstring(strtab,
722                                         sdp->sd_name, &stoff);
723                                     symtab[symtab_ndx].st_name = stoff;
724                             }
725                             sdp->sd_flags &= ~FLG_SY_CLEAN;
726                             if (symshndx)
727                                     _symshndx = &symshndx[symtab_ndx];
728                             sdp->sd_sym = sym = &symtab[symtab_ndx++];

730                             if ((sdp->sd_flags & FLG_SY_SPECSEC) &&
731                                 (sym->st_shndx == SHN_ABS) &&
732                                 !enter_in_ldynsym)
733                                     continue;
734                     } else if (enter_in_ldynsym) {
735                             /*
736                              * Not using symtab, but we do have ldynsym
737                              * available.
738                              */
739                             ldynsym[ldynsym_ndx] = *sym;
740                             (void) st_setstring(dynstr, sdp->sd_name,
741                                 &stoff);
742                             ldynsym[ldynsym_ndx].st_name = stoff;

744                             sdp->sd_flags &= ~FLG_SY_CLEAN;
745                             if (ldynshndx)
746                                     _symshndx = &ldynshndx[ldynsym_ndx];
747                             sdp->sd_sym = sym = &ldynsym[ldynsym_ndx];
748                             /* Add it to sort section if it qualifies */
749                             ADD_TO_DYNSORT(sdp, sym, type, ldynsym_ndx);
750                             ldynsym_ndx++;
751                     } else {        /* Not using symtab or ldynsym */
752                             /*
753                              * If this symbol requires modifying to provide
754                              * for a relocation or move table update, make
755                              * a copy of it.
756                              */
757                             if (!(sdp->sd_flags & FLG_SY_UPREQD) &&
758                                 !(sdp->sd_move))
759                                     continue;
760                             if ((sdp->sd_flags & FLG_SY_SPECSEC) &&
761                                 (sym->st_shndx == SHN_ABS))
762                                     continue;

764                             if (ld_sym_copy(sdp) == S_ERROR)
765                                     return ((Addr)S_ERROR);
766                             sym = sdp->sd_sym;
767                     }

769                     /*
770                      * Update the symbols contents if necessary.
771                      */
772                     update_done = 0;
773                     if (type == STT_FILE) {
774                             sdp->sd_shndx = sym->st_shndx = SHN_ABS;
775                             sdp->sd_flags |= FLG_SY_SPECSEC;
776                             update_done = 1;
777                     }

779                     /*
780                      * If we are expanding the locally bound partially
781                      * initialized symbols, then update the address here.
782                      */
```

```
783                     if (ofl->ofl_isparexpn &&
784                         (sdp->sd_flags & FLG_SY_PAREXPN) && !update_done) {
785                             sym->st_shndx = parexpnndx;
786                             sdp->sd_isc = ofl->ofl_isparexpn;
787                             sym->st_value = parexpnaddr;
788                             parexpnaddr += sym->st_size;
789                             if ((flags & FLG_OF_RELOBJ) == 0)
790                                     sym->st_value -= parexpnbase;
791                     }

793                     /*
794                      * If this isn't an UNDEF symbol (ie. an input section
795                      * is associated), update the symbols value and index.
796                      */
797                     if (((isc = sdp->sd_isc) != NULL) && !update_done) {
798                             Word    sectndx;

800                             osp = isc->is_osdesc;
801                             /* LINTED */
802                             sym->st_value +=
803                                 (Off)_elf_getxoff(isc->is_indata);
804                             if ((flags & FLG_OF_RELOBJ) == 0) {
805                                     sym->st_value += osp->os_shdr->sh_addr;
806                                     /*
807                                      * TLS symbols are relative to
808                                      * the TLS segment.
809                                      */
810                                     if ((type == STT_TLS) &&
811                                         (ofl->ofl_tlsphdr)) {
812                                             sym->st_value -=
813                                                 ofl->ofl_tlsphdr->p_vaddr;
814                                     }
815                             }
816                             /* LINTED */
817                             if ((sdp->sd_shndx = sectndx =
818                                 elf_ndxscn(osp->os_scn)) >= SHN_LORESERVE) {
819                                     if (_symshndx) {
820                                             *_symshndx = sectndx;
821                                     }
822                                     sym->st_shndx = SHN_XINDEX;
823                             } else {
824                                     /* LINTED */
825                                     sym->st_shndx = sectndx;
826                             }
827                     }

829                     /*
830                      * If entering the symbol in both the symtab and the
831                      * ldynsym, then the one in symtab needs to be
832                      * copied to ldynsym. If it is only in the ldynsym,
833                      * then the code above already set it up and we have
834                      * nothing more to do here.
835                      */
836                     if (enter_in_symtab && enter_in_ldynsym) {
837                             ldynsym[ldynsym_ndx] = *sym;
838                             (void) st_setstring(dynstr, sdp->sd_name,
839                                 &stoff);
840                             ldynsym[ldynsym_ndx].st_name = stoff;

842                             if (_symshndx && ldynshndx)
843                                     ldynshndx[ldynsym_ndx] = *_symshndx;

845                             /* Add it to sort section if it qualifies */
846                             ADD_TO_DYNSORT(sdp, sym, type, ldynsym_ndx);

848                             ldynsym_ndx++;
```



```
849                     }
850             }

852             /*
853              * If this input file has undergone object to symbol
854              * capabilities conversion, supply any new capabilities symbols.
855              * These symbols are copies of the original global symbols, and
856              * follow the existing local symbols that are supplied from this
857              * input file (which are identified with a preceding STT_FILE).
858              */
859             if (symtab && cdp && cdp->ca_syms) {
860                     Aliste          idx2;
861                     Cap_sym         *csp;

863                     for (APLIST_TRAVERSE(cdp->ca_syms, idx2, csp)) {
864                             Is_desc *isp;

866                             sdp = csp->cs_sdp;
867                             sym = sdp->sd_sym;

869                             if ((isp = sdp->sd_isc) != NULL) {
870                                     Os_desc *osp = isp->is_osdesc;

872                                     /*
873                                      * Update the symbols value.
874                                      */
875                                     /* LINTED */
876                                     sym->st_value +=
877                                         (Off)_elf_getxoff(isp->is_indata);
878                                     if ((flags & FLG_OF_RELOBJ) == 0)
879                                             sym->st_value +=
880                                                 osp->os_shdr->sh_addr;

882                                     /*
883                                      * Update the symbols section index.
884                                      */
885                                     sdp->sd_shndx = sym->st_shndx =
886                                         elf_ndxscn(osp->os_scn);
887                             }

889                             symtab[symtab_ndx] = *sym;
890                             (void) st_setstring(strtab, sdp->sd_name,
891                                 &stoff);
892                             symtab[symtab_ndx].st_name = stoff;
893                             sdp->sd_symndx = symtab_ndx++;
894                     }
895             }
896     }

898     symtab_gbl_bndx = symtab_ndx;   /* .symtab index of 1st global entry */

900     /*
901      * Two special symbols are '_init' and '_fini'.  If these are supplied
902      * by crti.o then they are used to represent the total concatenation of
903      * the '.init' and '.fini' sections.
904      *
905      * Determine whether any .init or .fini sections exist.  If these
906      * sections exist and a dynamic object is being built, but no '_init'
907      * or '_fini' symbols are found, then the user is probably building
908      * this object directly from ld(1) rather than using a compiler driver
909      * that provides the symbols via crt's.
910      *
911      * If the .init or .fini section exist, and their associated symbols,
912      * determine the size of the sections and updated the symbols value
913      * accordingly.
914      */
```

```
915         if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_INIT_U), SYM_NOHASH, 0,
916             ofl)) != NULL) && (sdp->sd_ref == REF_REL_NEED) && sdp->sd_isc &&
917             (sdp->sd_isc->is_osdesc == iosp)) {
918                 if (ld_sym_copy(sdp) == S_ERROR)
919                         return ((Addr)S_ERROR);
920                 sdp->sd_sym->st_size = sdp->sd_isc->is_osdesc->os_shdr->sh_size;

922         } else if (iosp && !(flags & FLG_OF_RELOBJ)) {
923                 ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_NOCRT),
924                     MSG_ORIG(MSG_SYM_INIT_U), MSG_ORIG(MSG_SCN_INIT));
925         }

927         if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_FINI_U), SYM_NOHASH, 0,
928             ofl)) != NULL) && (sdp->sd_ref == REF_REL_NEED) && sdp->sd_isc &&
929             (sdp->sd_isc->is_osdesc == fosp)) {
930                 if (ld_sym_copy(sdp) == S_ERROR)
931                         return ((Addr)S_ERROR);
932                 sdp->sd_sym->st_size = sdp->sd_isc->is_osdesc->os_shdr->sh_size;

934         } else if (fosp && !(flags & FLG_OF_RELOBJ)) {
935                 ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_NOCRT),
936                     MSG_ORIG(MSG_SYM_FINI_U), MSG_ORIG(MSG_SCN_FINI));
937         }

939         /*
940          * Assign .bss information for use with updating COMMON symbols.
941          */
942         if (ofl->ofl_isbss) {
943                 isc = ofl->ofl_isbss;
944                 osp = isc->is_osdesc;

946                 bssaddr = osp->os_shdr->sh_addr +
947                     (Off)_elf_getxoff(isc->is_indata);
948                 /* LINTED */
949                 bssndx = elf_ndxscn(osp->os_scn);
950         }

952 #if     defined(_ELF64)
953         /*
954          * For amd64 target, assign .lbss information for use
955          * with updating LCOMMON symbols.
956          */
957         if ((ld_targ.t_m.m_mach == EM_AMD64) && ofl->ofl_islbss) {
958                 osp = ofl->ofl_islbss->is_osdesc;

960                 lbssaddr = osp->os_shdr->sh_addr +
961                     (Off)_elf_getxoff(ofl->ofl_islbss->is_indata);
962                 /* LINTED */
963                 lbssndx = elf_ndxscn(osp->os_scn);
964         }
965 #endif
966         /*
967          * Assign .tlsbss information for use with updating COMMON symbols.
968          */
969         if (ofl->ofl_istlsbss) {
970                 osp = ofl->ofl_istlsbss->is_osdesc;
971                 tlsbssaddr = osp->os_shdr->sh_addr +
972                     (Off)_elf_getxoff(ofl->ofl_istlsbss->is_indata);
973                 /* LINTED */
974                 tlsbssndx = elf_ndxscn(osp->os_scn);
975         }

977         if ((sorted_syms = libld_calloc(ofl->ofl_globcnt +
978             ofl->ofl_elimcnt + ofl->ofl_scopecnt,
979             sizeof (*sorted_syms))) == NULL)
980                 return ((Addr)S_ERROR);
```



```
982         scndx = 0;
983         ssndx = ofl->ofl_scopecnt + ofl->ofl_elimcnt;

985         DBG_CALL(Dbg_syms_up_title(ofl->ofl_lml));

987         /*
988          * Traverse the internal symbol table updating global symbol information
989          * and allocating common.
990          */
991         for (sav = avl_first(&ofl->ofl_symavl); sav;
992             sav = AVL_NEXT(&ofl->ofl_symavl, sav)) {
993                 Sym     *symptr;
994                 int     local;
995                 int     restore;

997                 sdp = sav->sav_sdp;

999                 /*
1000                 * Ignore any symbols that have been marked as invalid during
1001                 * input processing.  Providing these aren't used for
1002                 * relocation, they will be dropped from the output image.
1003                 */
1004                if (sdp->sd_flags & FLG_SY_INVALID) {
1005                        DBG_CALL(Dbg_syms_old(ofl, sdp));
1006                        DBG_CALL(Dbg_syms_ignore(ofl, sdp));
1007                        continue;
1008                }

1010                /*
1011                 * Only needed symbols are copied to the output symbol table.
1012                 */
1013                if (sdp->sd_ref == REF_DYN_SEEN)
1014                        continue;

1016                if (SYM_IS_HIDDEN(sdp) && (flags & FLG_OF_PROCRED))
1017                        local = 1;
1018                else
1019                        local = 0;

1021                if (local || (ofl->ofl_hashbkts == 0)) {
1022                        sorted_syms[scndx++].sl_sdp = sdp;
1023                } else {
1024                        sorted_syms[ssndx].sl_hval = sdp->sd_aux->sa_hash %
1025                            ofl->ofl_hashbkts;
1026                        sorted_syms[ssndx].sl_sdp = sdp;
1027                        ssndx++;
1028                }

1030                /*
1031                 * Note - expand the COMMON symbols here because an address
1032                 * must be assigned to them in the same order that space was
1033                 * calculated in sym_validate().  If this ordering isn't
1034                 * followed differing alignment requirements can throw us all
1035                 * out of whack.
1036                 *
1037                 * The expanded .bss global symbol is handled here as well.
1038                 *
1039                 * The actual adding entries into the symbol table still occurs
1040                 * below in hashbucket order.
1041                 */
1042                symptr = sdp->sd_sym;
1043                restore = 0;
1044                if ((sdp->sd_flags & FLG_SY_PAREXPN) ||
1045                    ((sdp->sd_flags & FLG_SY_SPECSEC) &&
1046                    (sdp->sd_shndx = symptr->st_shndx) == SHN_COMMON)) {
```

```
1048                        /*
1049                         * An expanded symbol goes to a special .data section
1050                         * prepared for that purpose (ofl->ofl_isparexpn).
1051                         * Assign COMMON allocations to .bss.
1052                         * Otherwise leave it as is.
1053                         */
1054                        if (sdp->sd_flags & FLG_SY_PAREXPN) {
1055                                restore = 1;
1056                                sdp->sd_shndx = parexpnndx;
1057                                sdp->sd_flags &= ~FLG_SY_SPECSEC;
1058                                symptr->st_value = (Xword) S_ROUND(
1059                                    parexpnaddr, symptr->st_value);
1060                                parexpnaddr = symptr->st_value +
1061                                    symptr->st_size;
1062                                sdp->sd_isc = ofl->ofl_isparexpn;
1063                                sdp->sd_flags |= FLG_SY_COMMEXP;

1065                        } else if (ELF_ST_TYPE(symptr->st_info) != STT_TLS &&
1066                            (local || !(flags & FLG_OF_RELOBJ))) {
1067                                restore = 1;
1068                                sdp->sd_shndx = bssndx;
1069                                sdp->sd_flags &= ~FLG_SY_SPECSEC;
1070                                symptr->st_value = (Xword)S_ROUND(bssaddr,
1071                                    symptr->st_value);
1072                                bssaddr = symptr->st_value + symptr->st_size;
1073                                sdp->sd_isc = ofl->ofl_isbss;
1074                                sdp->sd_flags |= FLG_SY_COMMEXP;

1076                        } else if (ELF_ST_TYPE(symptr->st_info) == STT_TLS &&
1077                            (local || !(flags & FLG_OF_RELOBJ))) {
1078                                restore = 1;
1079                                sdp->sd_shndx = tlsbssndx;
1080                                sdp->sd_flags &= ~FLG_SY_SPECSEC;
1081                                symptr->st_value = (Xword)S_ROUND(tlsbssaddr,
1082                                    symptr->st_value);
1083                                tlsbssaddr = symptr->st_value + symptr->st_size;
1084                                sdp->sd_isc = ofl->ofl_istlsbss;
1085                                sdp->sd_flags |= FLG_SY_COMMEXP;
1086                                /*
1087                                 * TLS symbols are relative to the TLS segment.
1088                                 */
1089                                symptr->st_value -= ofl->ofl_tlsphdr->p_vaddr;
1090                        }
1091 #if     defined(_ELF64)
1092                } else if ((ld_targ.t_m.m_mach == EM_AMD64) &&
1093                    (sdp->sd_flags & FLG_SY_SPECSEC) &&
1094                    ((sdp->sd_shndx = symptr->st_shndx) ==
1095                    SHN_X86_64_LCOMMON) &&
1096                    ((local || !(flags & FLG_OF_RELOBJ)))) {
1097                        restore = 1;
1098                        sdp->sd_shndx = lbssndx;
1099                        sdp->sd_flags &= ~FLG_SY_SPECSEC;
1100                        symptr->st_value = (Xword)S_ROUND(lbssaddr,
1101                            symptr->st_value);
1102                        lbssaddr = symptr->st_value + symptr->st_size;
1103                        sdp->sd_isc = ofl->ofl_islbss;
1104                        sdp->sd_flags |= FLG_SY_COMMEXP;
1105 #endif
1106                }

1108                if (restore != 0) {
1109                        uchar_t         type, bind;

1111                        /*
1112                         * Make sure this COMMON symbol is returned to the same
```



```
1113                         * binding as was defined in the original relocatable
1114                         * object reference.
1115                         */
1116                        type = ELF_ST_TYPE(symptr->st_info);
1117                        if (sdp->sd_flags & FLG_SY_GLOBREF)
1118                                bind = STB_GLOBAL;
1119                        else
1120                                bind = STB_WEAK;

1122                        symptr->st_info = ELF_ST_INFO(bind, type);
1123                }
1124        }

1126        /*
1127         * If this is a dynamic object then add any local capabilities symbols.
1128         */
1129        if (dynsym && ofl->ofl_capfamilies) {
1130                Cap_avlnode     *cav;

1132                for (cav = avl_first(ofl->ofl_capfamilies); cav;
1133                    cav = AVL_NEXT(ofl->ofl_capfamilies, cav)) {
1134                        Cap_sym         *csp;
1135                        Aliste          idx;

1137                        for (APLIST_TRAVERSE(cav->cn_members, idx, csp)) {
1138                                sdp = csp->cs_sdp;

1140                                DBG_CALL(Dbg_syms_created(ofl->ofl_lml,
1141                                    sdp->sd_name));
1142                                DBG_CALL(Dbg_syms_entered(ofl, sdp->sd_sym,
1143                                    sdp));

1145                                dynsym[dynsym_ndx] = *sdp->sd_sym;

1147                                (void) st_setstring(dynstr, sdp->sd_name,
1148                                    &stoff);
1149                                dynsym[dynsym_ndx].st_name = stoff;

1151                                sdp->sd_sym = &dynsym[dynsym_ndx];
1152                                sdp->sd_symndx = dynsym_ndx;

1154                                /*
1155                                 * Indicate that this is a capabilities symbol.
1156                                 * Note, that this identification only provides
1157                                 * information regarding the symbol that is
1158                                 * visible from elfdump(1) -y.  The association
1159                                 * of a symbol to its capabilities is derived
1160                                 * from a .SUNW_capinfo entry.
1161                                 */
1162                                if (syminfo) {
1163                                        syminfo[dynsym_ndx].si_flags |=
1164                                            SYMINFO_FLG_CAP;
1165                                }

1167                                dynsym_ndx++;
1168                        }
1169                }
1170        }

1172        if (ofl->ofl_hashbkts) {
1173                qsort(sorted_syms + ofl->ofl_scopecnt + ofl->ofl_elimcnt,
1174                    ofl->ofl_globcnt, sizeof (Sym_s_list),
1175                    (int (*)(const void *, const void *))sym_hash_compare);
1176        }

1178        for (ssndx = 0; ssndx < (ofl->ofl_elimcnt + ofl->ofl_scopecnt +
```

```
1179            ofl->ofl_globcnt); ssndx++) {
1180                const char      *name;
1181                Sym             *sym;
1182                Sym_aux         *sap;
1183                Half            spec;
1184                int             local = 0, dynlocal = 0, enter_in_symtab;
1185                Gotndx          *gnp;
1186                Word            sectndx;

1188                sdp = sorted_syms[ssndx].sl_sdp;
1189                sectndx = 0;

1191                if (symtab)
1192                        enter_in_symtab = 1;
1193                else
1194                        enter_in_symtab = 0;

1196                /*
1197                 * Assign a got offset if necessary.
1198                 */
1199                if ((ld_targ.t_mr.mr_assign_got != NULL) &&
1200                    (*ld_targ.t_mr.mr_assign_got)(ofl, sdp) == S_ERROR)
1201                        return ((Addr)S_ERROR);

1203                if (DBG_ENABLED) {
1204                        Aliste  idx2;

1206                        for (ALIST_TRAVERSE(sdp->sd_GOTndxs, idx2, gnp)) {
1207                                gottable->gt_sym = sdp;
1208                                gottable->gt_gndx.gn_gotndx = gnp->gn_gotndx;
1209                                gottable->gt_gndx.gn_addend = gnp->gn_addend;
1210                                gottable++;
1211                        }

1213                        if (sdp->sd_aux && sdp->sd_aux->sa_PLTGOTndx) {
1214                                gottable->gt_sym = sdp;
1215                                gottable->gt_gndx.gn_gotndx =
1216                                    sdp->sd_aux->sa_PLTGOTndx;
1217                                gottable++;
1218                        }
1219                }

1221                /*
1222                 * If this symbol has been marked as being reduced to local
1223                 * scope then it will have to be placed in the scoped portion
1224                 * of the .symtab.  Retain the appropriate index for use in
1225                 * version symbol indexing and relocation.
1226                 */
1227                if (SYM_IS_HIDDEN(sdp) && (flags & FLG_OF_PROCRED)) {
1228                        local = 1;
1229                        if (!(sdp->sd_flags & FLG_SY_ELIM) && !dynsym)
1230                                sdp->sd_symndx = scopesym_ndx;
1231                        else
1232                                sdp->sd_symndx = 0;

1234                        if (sdp->sd_flags & FLG_SY_ELIM) {
1235                                enter_in_symtab = 0;
1236                        } else if (ldynsym && sdp->sd_sym->st_name &&
1237                            ldynsym_symtype[
1238                            ELF_ST_TYPE(sdp->sd_sym->st_info)]) {
1239                                dynlocal = 1;
1240                        }
1241                } else {
1242                        sdp->sd_symndx = *symndx;
1243                }
```



```
1245                /*
1246                 * Copy basic symbol and string information.
1247                 */
1248                name = sdp->sd_name;
1249                sap = sdp->sd_aux;

1251                /*
1252                 * If we require to record version symbol indexes, update the
1253                 * associated version symbol information for all defined
1254                 * symbols.  If a version definition is required any zero value
1255                 * symbol indexes would have been flagged as undefined symbol
1256                 * errors, however if we're just scoping these need to fall into
1257                 * the base of global symbols.
1258                 */
1259                if (sdp->sd_symndx && versym) {
1260                        Half    vndx = 0;

1262                        if (sdp->sd_flags & FLG_SY_MVTOCOMM) {
1263                                vndx = VER_NDX_GLOBAL;
1264                        } else if (sdp->sd_ref == REF_REL_NEED) {
1265                                vndx = sap->sa_overndx;

1267                                if ((vndx == 0) &&
1268                                    (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
1269                                        if (SYM_IS_HIDDEN(sdp))
1270                                                vndx = VER_NDX_LOCAL;
1271                                        else
1272                                                vndx = VER_NDX_GLOBAL;
1273                                }
1274                        } else if ((sdp->sd_ref == REF_DYN_NEED) &&
1275                            (sap->sa_dverndx > 0) &&
1276                            (sap->sa_dverndx <= sdp->sd_file->ifl_vercnt) &&
1277                            (sdp->sd_file->ifl_verndx != NULL)) {
1278                                /* Use index of verneed record */
1279                                vndx = sdp->sd_file->ifl_verndx
1280                                    [sap->sa_dverndx].vi_overndx;
1281                        }
1282                        versym[sdp->sd_symndx] = vndx;
1283                }

1285                /*
1286                 * If we are creating the .syminfo section then set per symbol
1287                 * flags here.
1288                 */
1289                if (sdp->sd_symndx && syminfo &&
1290                    !(sdp->sd_flags & FLG_SY_NOTAVAIL)) {
1291                        int     ndx = sdp->sd_symndx;
1292                        APlist  **alpp = &(ofl->ofl_symdtent);

1294                        if (sdp->sd_flags & FLG_SY_MVTOCOMM)
1295                                /*
1296                                 * Identify a copy relocation symbol.
1297                                 */
1298                                syminfo[ndx].si_flags |= SYMINFO_FLG_COPY;

1300                        if (sdp->sd_ref == REF_DYN_NEED) {
1301                                /*
1302                                 * A reference is bound to a needed dependency.
1303                                 * Save the syminfo entry, so that when the
1304                                 * .dynamic section has been updated, a
1305                                 * DT_NEEDED entry can be associated
1306                                 * (see update_osyminfo()).
1307                                 */
1308                                if (aplist_append(alpp, sdp,
1309                                    AL_CNT_OFL_SYMINFOSYMS) == NULL)
1310                                        return (0);
```

```
1312                                    /*
1313                                     * Flag that the symbol has a direct association
1314                                     * with the external reference (this is an old
1315                                     * tagging, that has no real effect by itself).
1316                                     */
1317                                    syminfo[ndx].si_flags |= SYMINFO_FLG_DIRECT;

1319                                    /*
1320                                     * Flag any lazy or deferred reference.
1321                                     */
1322                                    if (sdp->sd_flags & FLG_SY_LAZYLD)
1323                                            syminfo[ndx].si_flags |=
1324                                                SYMINFO_FLG_LAZYLOAD;
1325                                    if (sdp->sd_flags & FLG_SY_DEFERRED)
1326                                            syminfo[ndx].si_flags |=
1327                                                SYMINFO_FLG_DEFERRED;

1329                                    /*
1330                                     * Enable direct symbol bindings if:
1331                                     *
1332                                     *  -   Symbol was identified with the DIRECT
1333                                     *      keyword in a mapfile.
1334                                     *
1335                                     *  -   Symbol reference has been bound to a
1336                                     *      dependency which was specified as
1337                                     *      requiring direct bindings with -zdirect.
1338                                     *
1339                                     *  -   All symbol references are required to
1340                                     *      use direct bindings via -Bdirect.
1341                                     */
1342                                    if (sdp->sd_flags & FLG_SY_DIR)
1343                                            syminfo[ndx].si_flags |=
1344                                                SYMINFO_FLG_DIRECTBIND;

1346                            } else if ((sdp->sd_flags & FLG_SY_EXTERN) &&
1347                                (sdp->sd_sym->st_shndx == SHN_UNDEF)) {
1348                                    /*
1349                                     * If this symbol has been explicitly defined
1350                                     * as external, and remains unresolved, mark
1351                                     * it as external.
1352                                     */
1353                                    syminfo[ndx].si_boundto = SYMINFO_BT_EXTERN;

1355                            } else if ((sdp->sd_flags & FLG_SY_PARENT) &&
1356                                (sdp->sd_sym->st_shndx == SHN_UNDEF)) {
1357                                    /*
1358                                     * If this symbol has been explicitly defined
1359                                     * to be a reference to a parent object,
1360                                     * indicate whether a direct binding should be
1361                                     * established.
1362                                     */
1363                                    syminfo[ndx].si_flags |= SYMINFO_FLG_DIRECT;
1364                                    syminfo[ndx].si_boundto = SYMINFO_BT_PARENT;
1365                                    if (sdp->sd_flags & FLG_SY_DIR)
1366                                            syminfo[ndx].si_flags |=
1367                                                SYMINFO_FLG_DIRECTBIND;

1369                            } else if (sdp->sd_flags & FLG_SY_STDFLTR) {
1370                                    /*
1371                                     * A filter definition.  Although this symbol
1372                                     * can only be a stub, it might be necessary to
1373                                     * prevent external direct bindings.
1374                                     */
1375                                    syminfo[ndx].si_flags |= SYMINFO_FLG_FILTER;
1376                                    if (sdp->sd_flags & FLG_SY_NDIR)
```



```
1377                                            syminfo[ndx].si_flags |=
1378                                                SYMINFO_FLG_NOEXTDIRECT;

1380                            } else if (sdp->sd_flags & FLG_SY_AUXFLTR) {
1381                                    /*
1382                                     * An auxiliary filter definition.  By nature,
1383                                     * this definition is direct, in that should the
1384                                     * filtee lookup fail, we'll fall back to this
1385                                     * object.  It may still be necessary to
1386                                     * prevent external direct bindings.
1387                                     */
1388                                    syminfo[ndx].si_flags |= SYMINFO_FLG_AUXILIARY;
1389                                    if (sdp->sd_flags & FLG_SY_NDIR)
1390                                            syminfo[ndx].si_flags |=
1391                                                SYMINFO_FLG_NOEXTDIRECT;

1393                            } else if ((sdp->sd_ref == REF_REL_NEED) &&
1394                                (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
1395                                    /*
1396                                     * This definition exists within the object
1397                                     * being created.  Provide a default boundto
1398                                     * definition, which may be overridden later.
1399                                     */
1400                                    syminfo[ndx].si_boundto = SYMINFO_BT_NONE;

1402                                    /*
1403                                     * Indicate whether it is necessary to prevent
1404                                     * external direct bindings.
1405                                     */
1406                                    if (sdp->sd_flags & FLG_SY_NDIR) {
1407                                            syminfo[ndx].si_flags |=
1408                                                SYMINFO_FLG_NOEXTDIRECT;
1409                                    }

1411                                    /*
1412                                     * Indicate that this symbol is acting as an
1413                                     * individual interposer.
1414                                     */
1415                                    if (sdp->sd_flags & FLG_SY_INTPOSE) {
1416                                            syminfo[ndx].si_flags |=
1417                                                SYMINFO_FLG_INTERPOSE;
1418                                    }

1420                                    /*
1421                                     * Indicate that this symbol is deferred, and
1422                                     * hence should not be bound to during BIND_NOW
1423                                     * relocations.
1424                                     */
1425                                    if (sdp->sd_flags & FLG_SY_DEFERRED) {
1426                                            syminfo[ndx].si_flags |=
1427                                                SYMINFO_FLG_DEFERRED;
1428                                    }

1430                                    /*
1431                                     * If external bindings are allowed, indicate
1432                                     * the binding, and a direct binding if
1433                                     * necessary.
1434                                     */
1435                                    if ((sdp->sd_flags & FLG_SY_NDIR) == 0) {
1436                                            syminfo[ndx].si_flags |=
1437                                                SYMINFO_FLG_DIRECT;

1439                                            if (sdp->sd_flags & FLG_SY_DIR)
1440                                                    syminfo[ndx].si_flags |=
1441                                                        SYMINFO_FLG_DIRECTBIND;
```

```
1443                                    /*
1444                                     * Provide a default boundto definition,
1445                                     * which may be overridden later.
1446                                     */
1447                                    syminfo[ndx].si_boundto =
1448                                        SYMINFO_BT_SELF;
1449                            }

1451                            /*
1452                             * Indicate that this is a capabilities symbol.
1453                             * Note, that this identification only provides
1454                             * information regarding the symbol that is
1455                             * visible from elfdump(1) -y.  The association
1456                             * of a symbol to its capabilities is derived
1457                             * from a .SUNW_capinfo entry.
1458                             */
1459                            if ((sdp->sd_flags & FLG_SY_CAP) &&
1460                                ofl->ofl_oscapinfo) {
1461                                syminfo[ndx].si_flags |=
1462                                    SYMINFO_FLG_CAP;
1463                            }
1464                    }
1465            }

1467            /*
1468             * Note that the 'sym' value is reset to be one of the new
1469             * symbol table entries.  This symbol will be updated further
1470             * depending on the type of the symbol.  Process the .symtab
1471             * first, followed by the .dynsym, thus the 'sym' value will
1472             * remain as the .dynsym value when the .dynsym is present.
1473             * This ensures that any versioning symbols st_name value will
1474             * be appropriate for the string table used by version
1475             * entries.
1476             */
1477            if (enter_in_symtab) {
1478                    Word    _symndx;

1480                    if (local)
1481                            _symndx = scopesym_ndx;
1482                    else
1483                            _symndx = symtab_ndx;

1485                    symtab[_symndx] = *sdp->sd_sym;
1486                    sdp->sd_sym = sym = &symtab[_symndx];
1487                    (void) st_setstring(strtab, name, &stoff);
1488                    sym->st_name = stoff;
1489            }
1490            if (dynlocal) {
1491                    ldynsym[ldynscopesym_ndx] = *sdp->sd_sym;
1492                    sdp->sd_sym = sym = &ldynsym[ldynscopesym_ndx];
1493                    (void) st_setstring(dynstr, name, &stoff);
1494                    ldynsym[ldynscopesym_ndx].st_name = stoff;
1495                    /* Add it to sort section if it qualifies */
1496                    ADD_TO_DYNSORT(sdp, sym, ELF_ST_TYPE(sym->st_info),
1497                        ldynscopesym_ndx);
1498            }

1500            if (dynsym && !local) {
1501                    dynsym[dynsym_ndx] = *sdp->sd_sym;

1503                    /*
1504                     * Provided this isn't an unnamed register symbol,
1505                     * update the symbols name and hash value.
1506                     */
1507                    if (((sdp->sd_flags & FLG_SY_REGSYM) == 0) ||
1508                        dynsym[dynsym_ndx].st_name) {
```



```
1509                            (void) st_setstring(dynstr, name, &stoff);
1510                            dynsym[dynsym_ndx].st_name = stoff;

1512                            if (stoff) {
1513                                    Word    hashval, _hashndx;

1515                                    hashval =
1516                                        sap->sa_hash % ofl->ofl_hashbkts;

1518                                    /* LINTED */
1519                                    if (_hashndx = hashbkt[hashval]) {
1520                                            while (hashchain[_hashndx]) {
1521                                                    _hashndx =
1522                                                        hashchain[_hashndx];
1523                                            }
1524                                            hashchain[_hashndx] =
1525                                                sdp->sd_symndx;
1526                                    } else {
1527                                            hashbkt[hashval] =
1528                                                sdp->sd_symndx;
1529                                    }
1530                            }
1531                    }
1532                    sdp->sd_sym = sym = &dynsym[dynsym_ndx];

1534                    /*
1535                     * Add it to sort section if it qualifies.
1536                     * The indexes in that section are relative to the
1537                     * the adjacent SUNW_ldynsym/dymsym pair, so we
1538                     * add the number of items in SUNW_ldynsym to the
1539                     * dynsym index.
1540                     */
1541                    ADD_TO_DYNSORT(sdp, sym, ELF_ST_TYPE(sym->st_info),
1542                        ldynsym_cnt + dynsym_ndx);
1543            }

1545            if (!enter_in_symtab && (!dynsym || (local && !dynlocal))) {
1546                    if (!(sdp->sd_flags & FLG_SY_UPREQD))
1547                            continue;
1548                    sym = sdp->sd_sym;
1549            } else
1550                    sdp->sd_flags &= ~FLG_SY_CLEAN;

1552            /*
1553             * If we have a weak data symbol for which we need the real
1554             * symbol also, save this processing until later.
1555             *
1556             * The exception to this is if the weak/strong have PLT's
1557             * assigned to them.  In that case we don't do the post-weak
1558             * processing because the PLT's must be maintained so that we
1559             * can do 'interpositioning' on both of the symbols.
1560             */
1561            if ((sap->sa_linkndx) &&
1562                (ELF_ST_BIND(sym->st_info) == STB_WEAK) &&
1563                (!sap->sa_PLTndx)) {
1564                    Sym_desc        *_sdp;

1566                    _sdp = sdp->sd_file->ifl_oldndx[sap->sa_linkndx];

1568                    if (_sdp->sd_ref != REF_DYN_SEEN) {
1569                            Wk_desc wk;

1571                            if (enter_in_symtab) {
1572                                    if (local) {
1573                                            wk.wk_symtab =
1574                                                &symtab[scopesym_ndx];
```

```
1575                                        scopesym_ndx++;
1576                                } else {
1577                                        wk.wk_symtab =
1578                                            &symtab[symtab_ndx];
1579                                        symtab_ndx++;
1580                                }
1581                        } else {
1582                                wk.wk_symtab = NULL;
1583                        }
1584                        if (dynsym) {
1585                                if (!local) {
1586                                        wk.wk_dynsym =
1587                                            &dynsym[dynsym_ndx];
1588                                        dynsym_ndx++;
1589                                } else if (dynlocal) {
1590                                        wk.wk_dynsym =
1591                                            &ldynsym[ldynscopesym_ndx];
1592                                        ldynscopesym_ndx++;
1593                                }
1594                        } else {
1595                                wk.wk_dynsym = NULL;
1596                        }
1597                        wk.wk_weak = sdp;
1598                        wk.wk_alias = _sdp;

1600                        if (alist_append(&weak, &wk,
1601                            sizeof (Wk_desc), AL_CNT_WEAK) == NULL)
1602                                return ((Addr)S_ERROR);

1604                        continue;
1605                }
1606        }

1608        DBG_CALL(Dbg_syms_old(ofl, sdp));

1610        spec = NULL;
1611        /*
1612         * assign new symbol value.
1613         */
1614        sectndx = sdp->sd_shndx;
1615        if (sectndx == SHN_UNDEF) {
1616                if (((sdp->sd_flags & FLG_SY_REGSYM) == 0) &&
1617                    (sym->st_value != 0)) {
1618                        ld_eprintf(ofl, ERR_WARNING,
1619                            MSG_INTL(MSG_SYM_NOTNULL),
1620                            demangle(name), sdp->sd_file->ifl_name);
1621                }

1623                /*
1624                 * Undefined weak global, if we are generating a static
1625                 * executable, output as an absolute zero.  Otherwise
1626                 * leave it as is, ld.so.1 will skip symbols of this
1627                 * type (this technique allows applications and
1628                 * libraries to test for the existence of a symbol as an
1629                 * indication of the presence or absence of certain
1630                 * functionality).
1631                 */
1632                if (OFL_IS_STATIC_EXEC(ofl) &&
1633                    (ELF_ST_BIND(sym->st_info) == STB_WEAK)) {
1634                        sdp->sd_flags |= FLG_SY_SPECSEC;
1635                        sdp->sd_shndx = sectndx = SHN_ABS;
1636                }
1637        } else if ((sdp->sd_flags & FLG_SY_SPECSEC) &&
1638            (sectndx == SHN_COMMON)) {
1639                /* COMMONs have already been processed */
1640                /* EMPTY */
```



```
1641                        ;
1642                } else {
1643                        if ((sdp->sd_flags & FLG_SY_SPECSEC) &&
1644                            (sectndx == SHN_ABS))
1645                                spec = sdp->sd_aux->sa_symspec;

1647                        /* LINTED */
1648                        if (sdp->sd_flags & FLG_SY_COMMEXP) {
1649                                /*
1650                                 * This is (or was) a COMMON symbol which was
1651                                 * processed above - no processing
1652                                 * required here.
1653                                 */
1654                                ;
1655                        } else if (sdp->sd_ref == REF_DYN_NEED) {
1656                                uchar_t type, bind;

1658                                sectndx = SHN_UNDEF;
1659                                sym->st_value = 0;
1660                                sym->st_size = 0;

1662                                /*
1663                                 * Make sure this undefined symbol is returned
1664                                 * to the same binding as was defined in the
1665                                 * original relocatable object reference.
1666                                 */
1667                                type = ELF_ST_TYPE(sym-> st_info);
1668                                if (sdp->sd_flags & FLG_SY_GLOBREF)
1669                                        bind = STB_GLOBAL;
1670                                else
1671                                        bind = STB_WEAK;

1673                                sym->st_info = ELF_ST_INFO(bind, type);

1675                        } else if (((sdp->sd_flags & FLG_SY_SPECSEC) == 0) &&
1676                            (sdp->sd_ref == REF_REL_NEED)) {
1677                                osp = sdp->sd_isc->is_osdesc;
1678                                /* LINTED */
1679                                sectndx = elf_ndxscn(osp->os_scn);

1681                                /*
1682                                 * In an executable, the new symbol value is the
1683                                 * old value (offset into defining section) plus
1684                                 * virtual address of defining section.  In a
1685                                 * relocatable, the new value is the old value
1686                                 * plus the displacement of the section within
1687                                 * the file.
1688                                 */
1689                                /* LINTED */
1690                                sym->st_value +=
1691                                    (Off)_elf_getxoff(sdp->sd_isc->is_indata);

1693                                if (!(flags & FLG_OF_RELOBJ)) {
1694                                        sym->st_value += osp->os_shdr->sh_addr;
1695                                        /*
1696                                         * TLS symbols are relative to
1697                                         * the TLS segment.
1698                                         */
1699                                        if ((ELF_ST_TYPE(sym->st_info) ==
1700                                            STT_TLS) && (ofl->ofl_tlsphdr))
1701                                                sym->st_value -=
1702                                                    ofl->ofl_tlsphdr->p_vaddr;
1703                                }
1704                        }
1705                }
```

```
1707                 if (spec) {
1708                         switch (spec) {
1709                         case SDAUX_ID_ETEXT:
1710                                 sym->st_value = etext;
1711                                 sectndx = etext_ndx;
1712                                 if (etext_abs)
1713                                         sdp->sd_flags |= FLG_SY_SPECSEC;
1714                                 else
1715                                         sdp->sd_flags &= ~FLG_SY_SPECSEC;
1716                                 break;
1717                         case SDAUX_ID_EDATA:
1718                                 sym->st_value = edata;
1719                                 sectndx = edata_ndx;
1720                                 if (edata_abs)
1721                                         sdp->sd_flags |= FLG_SY_SPECSEC;
1722                                 else
1723                                         sdp->sd_flags &= ~FLG_SY_SPECSEC;
1724                                 break;
1725                         case SDAUX_ID_END:
1726                                 sym->st_value = end;
1727                                 sectndx = end_ndx;
1728                                 if (end_abs)
1729                                         sdp->sd_flags |= FLG_SY_SPECSEC;
1730                                 else
1731                                         sdp->sd_flags &= ~FLG_SY_SPECSEC;
1732                                 break;
1733                         case SDAUX_ID_START:
1734                                 sym->st_value = start;
1735                                 sectndx = start_ndx;
1736                                 sdp->sd_flags &= ~FLG_SY_SPECSEC;
1737                                 break;
1738                         case SDAUX_ID_DYN:
1739                                 if (flags & FLG_OF_DYNAMIC) {
1740                                         sym->st_value = ofl->
1741                                             ofl_osdynamic->os_shdr->sh_addr;
1742                                         /* LINTED */
1743                                         sectndx = elf_ndxscn(
1744                                             ofl->ofl_osdynamic->os_scn);
1745                                         sdp->sd_flags &= ~FLG_SY_SPECSEC;
1746                                 }
1747                                 break;
1748                         case SDAUX_ID_PLT:
1749                                 if (ofl->ofl_osplt) {
1750                                         sym->st_value = ofl->
1751                                             ofl_osplt->os_shdr->sh_addr;
1752                                         /* LINTED */
1753                                         sectndx = elf_ndxscn(
1754                                             ofl->ofl_osplt->os_scn);
1755                                         sdp->sd_flags &= ~FLG_SY_SPECSEC;
1756                                 }
1757                                 break;
1758                         case SDAUX_ID_GOT:
1759                                 /*
1760                                  * Symbol bias for negative growing tables is
1761                                  * stored in symbol's value during
1762                                  * allocate_got().
1763                                  */
1764                                 sym->st_value += ofl->
1765                                     ofl_osgot->os_shdr->sh_addr;
1766                                 /* LINTED */
1767                                 sectndx = elf_ndxscn(ofl->
1768                                     ofl_osgot->os_scn);
1769                                 sdp->sd_flags &= ~FLG_SY_SPECSEC;
1770                                 break;
1771                         case SDAUX_ID_SECBOUND_START:
1772                                 sym->st_value = sap->sa_boundsec->
```



```
1773                                     os_shdr->sh_addr;
1774                                 sectndx = elf_ndxscn(sap->sa_boundsec->os_scn);
1775                                 sdp->sd_flags &= ~FLG_SY_SPECSEC;
1776                                 break;
1777                         case SDAUX_ID_SECBOUND_STOP:
1778                                 sym->st_value = sap->sa_boundsec->
1779                                     os_shdr->sh_addr +
1780                                     sap->sa_boundsec->os_shdr->sh_size;
1781                                 sectndx = elf_ndxscn(sap->sa_boundsec->os_scn);
1782                                 sdp->sd_flags &= ~FLG_SY_SPECSEC;
1783                                 break;
1784 #endif /* ! codereview */
1785                         default:
1786                                 /* NOTHING */
1787                                 ;
1788                         }
1789                 }
1790
1791                 /*
1792                  * If a plt index has been assigned to an undefined function,
1793                  * update the symbols value to the appropriate .plt address.
1794                  */
1795                 if ((flags & FLG_OF_DYNAMIC) && (flags & FLG_OF_EXEC) &&
1796                     (sdp->sd_file) &&
1797                     (sdp->sd_file->ifl_ehdr->e_type == ET_DYN) &&
1798                     (ELF_ST_TYPE(sym->st_info) == STT_FUNC) &&
1799                     !(flags & FLG_OF_BFLAG)) {
1800                         if (sap->sa_PLTndx)
1801                                 sym->st_value =
1802                                     (*ld_targ.t_mr.mr_calc_plt_addr)(sdp, ofl);
1803                 }
1804
1805                 /*
1806                  * Finish updating the symbols.
1807                  */
1808
1809                 /*
1810                  * Sym Update: if scoped local - set local binding
1811                  */
1812                 if (local)
1813                         sym->st_info = ELF_ST_INFO(STB_LOCAL,
1814                             ELF_ST_TYPE(sym->st_info));
1815
1816                 /*
1817                  * Sym Updated: If both the .symtab and .dynsym
1818                  * are present then we've actually updated the information in
1819                  * the .dynsym, therefore copy this same information to the
1820                  * .symtab entry.
1821                  */
1822                 sdp->sd_shndx = sectndx;
1823                 if (enter_in_symtab && dynsym && (!local || dynlocal)) {
1824                         Word _symndx = dynlocal ? scopesym_ndx : symtab_ndx;
1825
1826                         symtab[_symndx].st_value = sym->st_value;
1827                         symtab[_symndx].st_size = sym->st_size;
1828                         symtab[_symndx].st_info = sym->st_info;
1829                         symtab[_symndx].st_other = sym->st_other;
1830                 }
1831
1832                 if (enter_in_symtab) {
1833                         Word    _symndx;
1834
1835                         if (local)
1836                                 _symndx = scopesym_ndx++;
1837                         else
1838                                 _symndx = symtab_ndx++;
```

```
1839                        if (((sdp->sd_flags & FLG_SY_SPECSEC) == 0) &&
1840                            (sectndx >= SHN_LORESERVE)) {
1841                                assert(symshndx != NULL);
1842                                symshndx[_symndx] = sectndx;
1843                                symtab[_symndx].st_shndx = SHN_XINDEX;
1844                        } else {
1845                                /* LINTED */
1846                                symtab[_symndx].st_shndx = (Half)sectndx;
1847                        }
1848                }

1850                if (dynsym && (!local || dynlocal)) {
1851                        /*
1852                         * dynsym and ldynsym are distinct tables, so
1853                         * we use indirection to access the right one
1854                         * and the related extended section index array.
1855                         */
1856                        Word    _symndx;
1857                        Sym     *_dynsym;
1858                        Word    *_dynshndx;

1860                        if (!local) {
1861                                _symndx = dynsym_ndx++;
1862                                _dynsym = dynsym;
1863                                _dynshndx = dynshndx;
1864                        } else {
1865                                _symndx = ldynscopesym_ndx++;
1866                                _dynsym = ldynsym;
1867                                _dynshndx = ldynshndx;
1868                        }
1869                        if (((sdp->sd_flags & FLG_SY_SPECSEC) == 0) &&
1870                            (sectndx >= SHN_LORESERVE)) {
1871                                assert(_dynshndx != NULL);
1872                                _dynshndx[_symndx] = sectndx;
1873                                _dynsym[_symndx].st_shndx = SHN_XINDEX;
1874                        } else {
1875                                /* LINTED */
1876                                _dynsym[_symndx].st_shndx = (Half)sectndx;
1877                        }
1878                }

1880                DBG_CALL(Dbg_syms_new(ofl, sym, sdp));
1881        }

1883        /*
1884         * Now that all the symbols have been processed update any weak symbols
1885         * information (ie. copy all information except 'st_name').  As both
1886         * symbols will be represented in the output, return the weak symbol to
1887         * its correct type.
1888         */
1889        for (ALIST_TRAVERSE(weak, idx1, wkp)) {
1890                Sym_desc        *sdp, *_sdp;
1891                Sym             *sym, *_sym, *__sym;
1892                uchar_t         bind;

1894                sdp = wkp->wk_weak;
1895                _sdp = wkp->wk_alias;
1896                _sym = __sym = _sdp->sd_sym;

1898                sdp->sd_flags |= FLG_SY_WEAKDEF;

1900                /*
1901                 * If the symbol definition has been scoped then assign it to
1902                 * be local, otherwise if it's from a shared object then we need
1903                 * to maintain the binding of the original reference.
1904                 */
```



```
1905                if (SYM_IS_HIDDEN(sdp)) {
1906                        if (flags & FLG_OF_PROCRED)
1907                                bind = STB_LOCAL;
1908                        else
1909                                bind = STB_WEAK;
1910                } else if ((sdp->sd_ref == REF_DYN_NEED) &&
1911                    (sdp->sd_flags & FLG_SY_GLOBREF))
1912                        bind = STB_GLOBAL;
1913                else
1914                        bind = STB_WEAK;

1916                DBG_CALL(Dbg_syms_old(ofl, sdp));
1917                if ((sym = wkp->wk_symtab) != NULL) {
1918                        sym->st_value = _sym->st_value;
1919                        sym->st_size = _sym->st_size;
1920                        sym->st_other = _sym->st_other;
1921                        sym->st_shndx = _sym->st_shndx;
1922                        sym->st_info = ELF_ST_INFO(bind,
1923                            ELF_ST_TYPE(sym->st_info));
1924                        __sym = sym;
1925                }
1926                if ((sym = wkp->wk_dynsym) != NULL) {
1927                        sym->st_value = _sym->st_value;
1928                        sym->st_size = _sym->st_size;
1929                        sym->st_other = _sym->st_other;
1930                        sym->st_shndx = _sym->st_shndx;
1931                        sym->st_info = ELF_ST_INFO(bind,
1932                            ELF_ST_TYPE(sym->st_info));
1933                        __sym = sym;
1934                }
1935                DBG_CALL(Dbg_syms_new(ofl, __sym, sdp));
1936        }

1938        /*
1939         * Now display GOT debugging information if required.
1940         */
1941        DBG_CALL(Dbg_got_display(ofl, 0, 0,
1942            ld_targ.t_m.m_got_xnumber, ld_targ.t_m.m_got_entsize));

1944        /*
1945         * Update the section headers information. sh_info is
1946         * supposed to contain the offset at which the first
1947         * global symbol resides in the symbol table, while
1948         * sh_link contains the section index of the associated
1949         * string table.
1950         */
1951        if (symtab) {
1952                Shdr    *shdr = ofl->ofl_ossymtab->os_shdr;

1954                shdr->sh_info = symtab_gbl_bndx;
1955                /* LINTED */
1956                shdr->sh_link = (Word)elf_ndxscn(ofl->ofl_osstrtab->os_scn);
1957                if (symshndx)
1958                        ofl->ofl_ossymshndx->os_shdr->sh_link =
1959                            (Word)elf_ndxscn(ofl->ofl_ossymtab->os_scn);

1961                /*
1962                 * Ensure that the expected number of symbols
1963                 * were entered into the right spots:
1964                 *      - Scoped symbols in the right range
1965                 *      - Globals start at the right spot
1966                 *              (correct number of locals entered)
1967                 *      - The table is exactly filled
1968                 *              (correct number of globals entered)
1969                 */
1970                assert((scopesym_bndx + ofl->ofl_scopecnt) == scopesym_ndx);
```

```
1971                assert(shdr->sh_info == SYMTAB_LOC_CNT(ofl));
1972                assert((shdr->sh_info + ofl->ofl_globcnt) == symtab_ndx);
1973        }
1974        if (dynsym) {
1975                Shdr    *shdr = ofl->ofl_osdynsym->os_shdr;

1977                shdr->sh_info = DYNSYM_LOC_CNT(ofl);
1978                /* LINTED */
1979                shdr->sh_link = (Word)elf_ndxscn(ofl->ofl_osdynstr->os_scn);

1981                ofl->ofl_oshash->os_shdr->sh_link =
1982                    /* LINTED */
1983                    (Word)elf_ndxscn(ofl->ofl_osdynsym->os_scn);
1984                if (dynshndx) {
1985                        shdr = ofl->ofl_osdynshndx->os_shdr;
1986                        shdr->sh_link =
1987                            (Word)elf_ndxscn(ofl->ofl_osdynsym->os_scn);
1988                }
1989        }
1990        if (ldynsym) {
1991                Shdr    *shdr = ofl->ofl_osldynsym->os_shdr;

1993                /* ldynsym has no globals, so give index one past the end */
1994                shdr->sh_info = ldynsym_ndx;

1996                /*
1997                 * The ldynsym and dynsym must be adjacent. The
1998                 * idea is that rtld should be able to start with
1999                 * the ldynsym and march straight through the end
2000                 * of dynsym, seeing them as a single symbol table,
2001                 * despite the fact that they are in distinct sections.
2002                 * Ensure that this happened correctly.
2003                 *
2004                 * Note that I use ldynsym_ndx here instead of the
2005                 * computation I used to set the section size
2006                 * (found in ldynsym_cnt). The two will agree, unless
2007                 * we somehow miscounted symbols or failed to insert them
2008                 * all. Using ldynsym_ndx here catches that error in
2009                 * addition to checking for adjacency.
2010                 */
2011                assert(dynsym == (ldynsym + ldynsym_ndx));


2014                /* LINTED */
2015                shdr->sh_link = (Word)elf_ndxscn(ofl->ofl_osdynstr->os_scn);

2017                if (ldynshndx) {
2018                        shdr = ofl->ofl_osldynshndx->os_shdr;
2019                        shdr->sh_link =
2020                            (Word)elf_ndxscn(ofl->ofl_osldynsym->os_scn);
2021                }

2023                /*
2024                 * The presence of .SUNW_ldynsym means that there may be
2025                 * associated sort sections, one for regular symbols
2026                 * and the other for TLS. Each sort section needs the
2027                 * following done:
2028                 *      - Section header link references .SUNW_ldynsym
2029                 *      - Should have received the expected # of items
2030                 *      - Sorted by increasing address
2031                 */
2032                if (ofl->ofl_osdynsymsort) {    /* .SUNW_dynsymsort */
2033                        ofl->ofl_osdynsymsort->os_shdr->sh_link =
2034                            (Word)elf_ndxscn(ofl->ofl_osldynsym->os_scn);
2035                        assert(ofl->ofl_dynsymsortcnt == dynsymsort_ndx);
```



```
2037                        if (dynsymsort_ndx > 1) {
2038                                dynsort_compare_syms = ldynsym;
2039                                qsort(dynsymsort, dynsymsort_ndx,
2040                                    sizeof (*dynsymsort), dynsort_compare);
2041                                dynsort_dupwarn(ofl, ldynsym,
2042                                    st_getstrbuf(dynstr),
2043                                    dynsymsort, dynsymsort_ndx,
2044                                    MSG_ORIG(MSG_SCN_DYNSYMSORT));
2045                        }
2046                }
2047                if (ofl->ofl_osdyntlssort) {    /* .SUNW_dyntlssort */
2048                        ofl->ofl_osdyntlssort->os_shdr->sh_link =
2049                            (Word)elf_ndxscn(ofl->ofl_osldynsym->os_scn);
2050                        assert(ofl->ofl_dyntlssortcnt == dyntlssort_ndx);

2052                        if (dyntlssort_ndx > 1) {
2053                                dynsort_compare_syms = ldynsym;
2054                                qsort(dyntlssort, dyntlssort_ndx,
2055                                    sizeof (*dyntlssort), dynsort_compare);
2056                                dynsort_dupwarn(ofl, ldynsym,
2057                                    st_getstrbuf(dynstr),
2058                                    dyntlssort, dyntlssort_ndx,
2059                                    MSG_ORIG(MSG_SCN_DYNTLSSORT));
2060                        }
2061                }
2062        }

2064        /*
2065         * Used by ld.so.1 only.
2066         */
2067        return (etext);

2069 #undef ADD_TO_DYNSORT
2070 }

2072 /*
2073  * Build the dynamic section.
2074  *
2075  * This routine must be maintained in parallel with make_dynamic()
2076  * in sections.c
2077  */
2078 static int
2079 update_odynamic(Ofl_desc *ofl)
2080 {
2081        Aliste          idx;
2082        Ifl_desc        *ifl;
2083        Sym_desc        *sdp;
2084        Shdr            *shdr;
2085        Dyn             *_dyn = (Dyn *)ofl->ofl_osdynamic->os_outdata->d_buf;
2086        Dyn             *dyn;
2087        Os_desc         *symosp, *strosp;
2088        Str_tbl         *strtbl;
2089        size_t          stoff;
2090        ofl_flag_t      flags = ofl->ofl_flags;
2091        int             not_relobj = !(flags & FLG_OF_RELOBJ);
2092        Word            cnt;

2094        /*
2095         * Relocatable objects can be built with -r and -dy to trigger the
2096         * creation of a .dynamic section.  This model is used to create kernel
2097         * device drivers.  The .dynamic section provides a subset of userland
2098         * .dynamic entries, typically entries such as DT_NEEDED and DT_RUNPATH.
2099         *
2100         * Within a dynamic object, any .dynamic string references are to the
2101         * .dynstr table.  Within a relocatable object, these strings can reside
2102         * within the .strtab.
```

```
2103          */
2104         if (OFL_IS_STATIC_OBJ(ofl)) {
2105                 symosp = ofl->ofl_ossymtab;
2106                 strosp = ofl->ofl_osstrtab;
2107                 strtbl = ofl->ofl_strtab;
2108         } else {
2109                 symosp = ofl->ofl_osdynsym;
2110                 strosp = ofl->ofl_osdynstr;
2111                 strtbl = ofl->ofl_dynstrtab;
2112         }

2114         /* LINTED */
2115         ofl->ofl_osdynamic->os_shdr->sh_link = (Word)elf_ndxscn(strosp->os_scn);

2117         dyn = _dyn;

2119         for (APLIST_TRAVERSE(ofl->ofl_sos, idx, ifl)) {
2120                 if ((ifl->ifl_flags &
2121                     (FLG_IF_IGNORE | FLG_IF_DEPREQD)) == FLG_IF_IGNORE)
2122                         continue;

2124                 /*
2125                  * Create and set up the DT_POSFLAG_1 entry here if required.
2126                  */
2127                 if ((ifl->ifl_flags & MSK_IF_POSFLAG1) &&
2128                     (ifl->ifl_flags & FLG_IF_NEEDED) && not_relobj) {
2129                         dyn->d_tag = DT_POSFLAG_1;
2130                         if (ifl->ifl_flags & FLG_IF_LAZYLD)
2131                                 dyn->d_un.d_val = DF_P1_LAZYLOAD;
2132                         if (ifl->ifl_flags & FLG_IF_GRPPRM)
2133                                 dyn->d_un.d_val |= DF_P1_GROUPPERM;
2134                         if (ifl->ifl_flags & FLG_IF_DEFERRED)
2135                                 dyn->d_un.d_val |= DF_P1_DEFERRED;
2136                         dyn++;
2137                 }

2139                 if (ifl->ifl_flags & (FLG_IF_NEEDED | FLG_IF_NEEDSTR))
2140                         dyn->d_tag = DT_NEEDED;
2141                 else
2142                         continue;

2144                 (void) st_setstring(strtbl, ifl->ifl_soname, &stoff);
2145                 dyn->d_un.d_val = stoff;
2146                 /* LINTED */
2147                 ifl->ifl_neededndx = (Half)(((uintptr_t)dyn - (uintptr_t)_dyn) /
2148                     sizeof (Dyn));
2149                 dyn++;
2150         }

2152         if (not_relobj) {
2153                 if (ofl->ofl_dtsfltrs != NULL) {
2154                         Dfltr_desc      *dftp;

2156                         for (ALIST_TRAVERSE(ofl->ofl_dtsfltrs, idx, dftp)) {
2157                                 if (dftp->dft_flag == FLG_SY_AUXFLTR)
2158                                         dyn->d_tag = DT_SUNW_AUXILIARY;
2159                                 else
2160                                         dyn->d_tag = DT_SUNW_FILTER;

2162                                 (void) st_setstring(strtbl, dftp->dft_str,
2163                                     &stoff);
2164                                 dyn->d_un.d_val = stoff;
2165                                 dftp->dft_ndx = (Half)(((uintptr_t)dyn -
2166                                     (uintptr_t)_dyn) / sizeof (Dyn));
2167                                 dyn++;
2168                         }
```



```
2169                 }
2170                 if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_INIT_U),
2171                     SYM_NOHASH, 0, ofl)) != NULL) &&
2172                     (sdp->sd_ref == REF_REL_NEED) &&
2173                     (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
2174                         dyn->d_tag = DT_INIT;
2175                         dyn->d_un.d_ptr = sdp->sd_sym->st_value;
2176                         dyn++;
2177                 }
2178                 if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_FINI_U),
2179                     SYM_NOHASH, 0, ofl)) != NULL) &&
2180                     (sdp->sd_ref == REF_REL_NEED) &&
2181                     (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
2182                         dyn->d_tag = DT_FINI;
2183                         dyn->d_un.d_ptr = sdp->sd_sym->st_value;
2184                         dyn++;
2185                 }
2186                 if (ofl->ofl_soname) {
2187                         dyn->d_tag = DT_SONAME;
2188                         (void) st_setstring(strtbl, ofl->ofl_soname, &stoff);
2189                         dyn->d_un.d_val = stoff;
2190                         dyn++;
2191                 }
2192                 if (ofl->ofl_filtees) {
2193                         if (flags & FLG_OF_AUX) {
2194                                 dyn->d_tag = DT_AUXILIARY;
2195                         } else {
2196                                 dyn->d_tag = DT_FILTER;
2197                         }
2198                         (void) st_setstring(strtbl, ofl->ofl_filtees, &stoff);
2199                         dyn->d_un.d_val = stoff;
2200                         dyn++;
2201                 }
2202         }

2204         if (ofl->ofl_rpath) {
2205                 (void) st_setstring(strtbl, ofl->ofl_rpath, &stoff);
2206                 dyn->d_tag = DT_RUNPATH;
2207                 dyn->d_un.d_val = stoff;
2208                 dyn++;
2209                 dyn->d_tag = DT_RPATH;
2210                 dyn->d_un.d_val = stoff;
2211                 dyn++;
2212         }

2214         if (not_relobj) {
2215                 Aliste  idx;
2216                 Sg_desc *sgp;

2218                 if (ofl->ofl_config) {
2219                         dyn->d_tag = DT_CONFIG;
2220                         (void) st_setstring(strtbl, ofl->ofl_config, &stoff);
2221                         dyn->d_un.d_val = stoff;
2222                         dyn++;
2223                 }
2224                 if (ofl->ofl_depaudit) {
2225                         dyn->d_tag = DT_DEPAUDIT;
2226                         (void) st_setstring(strtbl, ofl->ofl_depaudit, &stoff);
2227                         dyn->d_un.d_val = stoff;
2228                         dyn++;
2229                 }
2230                 if (ofl->ofl_audit) {
2231                         dyn->d_tag = DT_AUDIT;
2232                         (void) st_setstring(strtbl, ofl->ofl_audit, &stoff);
2233                         dyn->d_un.d_val = stoff;
2234                         dyn++;
```

```
2235                }

2237                dyn->d_tag = DT_HASH;
2238                dyn->d_un.d_ptr = ofl->ofl_oshash->os_shdr->sh_addr;
2239                dyn++;

2241                shdr = strosp->os_shdr;
2242                dyn->d_tag = DT_STRTAB;
2243                dyn->d_un.d_ptr = shdr->sh_addr;
2244                dyn++;

2246                dyn->d_tag = DT_STRSZ;
2247                dyn->d_un.d_ptr = shdr->sh_size;
2248                dyn++;

2250                /*
2251                 * Note, the shdr is set and used in the ofl->ofl_osldynsym case
2252                 * that follows.
2253                 */
2254                shdr = symosp->os_shdr;
2255                dyn->d_tag = DT_SYMTAB;
2256                dyn->d_un.d_ptr = shdr->sh_addr;
2257                dyn++;

2259                dyn->d_tag = DT_SYMENT;
2260                dyn->d_un.d_ptr = shdr->sh_entsize;
2261                dyn++;

2263                if (ofl->ofl_osldynsym) {
2264                        Shdr    *lshdr = ofl->ofl_osldynsym->os_shdr;

2266                        /*
2267                         * We have arranged for the .SUNW_ldynsym data to be
2268                         * immediately in front of the .dynsym data.
2269                         * This means that you could start at the top
2270                         * of .SUNW_ldynsym and see the data for both tables
2271                         * without a break. This is the view we want to
2272                         * provide for DT_SUNW_SYMTAB, which is why we
2273                         * add the lengths together.
2274                         */
2275                        dyn->d_tag = DT_SUNW_SYMTAB;
2276                        dyn->d_un.d_ptr = lshdr->sh_addr;
2277                        dyn++;

2279                        dyn->d_tag = DT_SUNW_SYMSZ;
2280                        dyn->d_un.d_val = lshdr->sh_size + shdr->sh_size;
2281                        dyn++;
2282                }

2284                if (ofl->ofl_osdynsymsort || ofl->ofl_osdyntlssort) {
2285                        dyn->d_tag = DT_SUNW_SORTENT;
2286                        dyn->d_un.d_val = sizeof (Word);
2287                        dyn++;
2288                }

2290                if (ofl->ofl_osdynsymsort) {
2291                        shdr = ofl->ofl_osdynsymsort->os_shdr;

2293                        dyn->d_tag = DT_SUNW_SYMSORT;
2294                        dyn->d_un.d_ptr = shdr->sh_addr;
2295                        dyn++;

2297                        dyn->d_tag = DT_SUNW_SYMSORTSZ;
2298                        dyn->d_un.d_val = shdr->sh_size;
2299                        dyn++;
2300                }
```



```
2302                if (ofl->ofl_osdyntlssort) {
2303                        shdr = ofl->ofl_osdyntlssort->os_shdr;

2305                        dyn->d_tag = DT_SUNW_TLSSORT;
2306                        dyn->d_un.d_ptr = shdr->sh_addr;
2307                        dyn++;

2309                        dyn->d_tag = DT_SUNW_TLSSORTSZ;
2310                        dyn->d_un.d_val = shdr->sh_size;
2311                        dyn++;
2312                }

2314                /*
2315                 * Reserve the DT_CHECKSUM entry.  Its value will be filled in
2316                 * after the complete image is built.
2317                 */
2318                dyn->d_tag = DT_CHECKSUM;
2319                ofl->ofl_checksum = &dyn->d_un.d_val;
2320                dyn++;

2322                /*
2323                 * Versioning sections: DT_VERDEF and DT_VERNEED.
2324                 *
2325                 * The Solaris ld does not produce DT_VERSYM, but the GNU ld
2326                 * does, in order to support their style of versioning, which
2327                 * differs from ours:
2328                 *
2329                 *      - The top bit of the 16-bit Versym index is
2330                 *              not part of the version, but is interpreted
2331                 *              as a "hidden bit".
2332                 *
2333                 *      - External (SHN_UNDEF) symbols can have non-zero
2334                 *              Versym values, which specify versions in
2335                 *              referenced objects, via the Verneed section.
2336                 *
2337                 *      - The vna_other field of the Vernaux structures
2338                 *              found in the Verneed section are not zero as
2339                 *              with Solaris, but instead contain the version
2340                 *              index to be used by Versym indices to reference
2341                 *              the given external version.
2342                 *
2343                 * The Solaris ld, rtld, and elfdump programs all interpret the
2344                 * presence of DT_VERSYM as meaning that GNU versioning rules
2345                 * apply to the given file. If DT_VERSYM is not present,
2346                 * then Solaris versioning rules apply. If we should ever need
2347                 * to change our ld so that it does issue DT_VERSYM, then
2348                 * this rule for detecting GNU versioning will no longer work.
2349                 * In that case, we will have to invent a way to explicitly
2350                 * specify the style of versioning in use, perhaps via a
2351                 * new dynamic entry named something like DT_SUNW_VERSIONSTYLE,
2352                 * where the d_un.d_val value specifies which style is to be
2353                 * used.
2354                 */
2355                if ((flags & (FLG_OF_VERDEF | FLG_OF_NOVERSEC)) ==
2356                    FLG_OF_VERDEF) {
2357                        shdr = ofl->ofl_osverdef->os_shdr;

2359                        dyn->d_tag = DT_VERDEF;
2360                        dyn->d_un.d_ptr = shdr->sh_addr;
2361                        dyn++;
2362                        dyn->d_tag = DT_VERDEFNUM;
2363                        dyn->d_un.d_ptr = shdr->sh_info;
2364                        dyn++;
2365                }
2366                if ((flags & (FLG_OF_VERNEED | FLG_OF_NOVERSEC)) ==
```

```
2367                    FLG_OF_VERNEED) {
2368                shdr = ofl->ofl_osverneed->os_shdr;

2370                dyn->d_tag = DT_VERNEED;
2371                dyn->d_un.d_ptr = shdr->sh_addr;
2372                dyn++;
2373                dyn->d_tag = DT_VERNEEDNUM;
2374                dyn->d_un.d_ptr = shdr->sh_info;
2375                dyn++;
2376        }

2378        if ((flags & FLG_OF_COMREL) && ofl->ofl_relocrelcnt) {
2379                dyn->d_tag = ld_targ.t_m.m_rel_dt_count;
2380                dyn->d_un.d_val = ofl->ofl_relocrelcnt;
2381                dyn++;
2382        }
2383        if (flags & FLG_OF_TEXTREL) {
2384                /*
2385                 * Only the presence of this entry is used in this
2386                 * implementation, not the value stored.
2387                 */
2388                dyn->d_tag = DT_TEXTREL;
2389                dyn->d_un.d_val = 0;
2390                dyn++;
2391        }

2393        if (ofl->ofl_osfiniarray) {
2394                shdr = ofl->ofl_osfiniarray->os_shdr;

2396                dyn->d_tag = DT_FINI_ARRAY;
2397                dyn->d_un.d_ptr = shdr->sh_addr;
2398                dyn++;

2400                dyn->d_tag = DT_FINI_ARRAYSZ;
2401                dyn->d_un.d_val = shdr->sh_size;
2402                dyn++;
2403        }

2405        if (ofl->ofl_osinitarray) {
2406                shdr = ofl->ofl_osinitarray->os_shdr;

2408                dyn->d_tag = DT_INIT_ARRAY;
2409                dyn->d_un.d_ptr = shdr->sh_addr;
2410                dyn++;

2412                dyn->d_tag = DT_INIT_ARRAYSZ;
2413                dyn->d_un.d_val = shdr->sh_size;
2414                dyn++;
2415        }

2417        if (ofl->ofl_ospreinitarray) {
2418                shdr = ofl->ofl_ospreinitarray->os_shdr;

2420                dyn->d_tag = DT_PREINIT_ARRAY;
2421                dyn->d_un.d_ptr = shdr->sh_addr;
2422                dyn++;

2424                dyn->d_tag = DT_PREINIT_ARRAYSZ;
2425                dyn->d_un.d_val = shdr->sh_size;
2426                dyn++;
2427        }

2429        if (ofl->ofl_pltcnt) {
2430                shdr = ofl->ofl_osplt->os_relosdesc->os_shdr;

2432                dyn->d_tag = DT_PLTRELSZ;
```



```
2433                dyn->d_un.d_ptr = shdr->sh_size;
2434                dyn++;
2435                dyn->d_tag = DT_PLTREL;
2436                dyn->d_un.d_ptr = ld_targ.t_m.m_rel_dt_type;
2437                dyn++;
2438                dyn->d_tag = DT_JMPREL;
2439                dyn->d_un.d_ptr = shdr->sh_addr;
2440                dyn++;
2441        }
2442        if (ofl->ofl_pltpad) {
2443                shdr = ofl->ofl_osplt->os_shdr;

2445                dyn->d_tag = DT_PLTPAD;
2446                if (ofl->ofl_pltcnt) {
2447                        dyn->d_un.d_ptr = shdr->sh_addr +
2448                            ld_targ.t_m.m_plt_reservsz +
2449                            ofl->ofl_pltcnt * ld_targ.t_m.m_plt_entsize;
2450                } else
2451                        dyn->d_un.d_ptr = shdr->sh_addr;
2452                dyn++;
2453                dyn->d_tag = DT_PLTPADSZ;
2454                dyn->d_un.d_val = ofl->ofl_pltpad *
2455                    ld_targ.t_m.m_plt_entsize;
2456                dyn++;
2457        }
2458        if (ofl->ofl_relocsz) {
2459                shdr = ofl->ofl_osrelhead->os_shdr;

2461                dyn->d_tag = ld_targ.t_m.m_rel_dt_type;
2462                dyn->d_un.d_ptr = shdr->sh_addr;
2463                dyn++;
2464                dyn->d_tag = ld_targ.t_m.m_rel_dt_size;
2465                dyn->d_un.d_ptr = ofl->ofl_relocsz;
2466                dyn++;
2467                dyn->d_tag = ld_targ.t_m.m_rel_dt_ent;
2468                if (shdr->sh_type == SHT_REL)
2469                        dyn->d_un.d_ptr = sizeof (Rel);
2470                else
2471                        dyn->d_un.d_ptr = sizeof (Rela);
2472                dyn++;
2473        }
2474        if (ofl->ofl_ossyminfo) {
2475                shdr = ofl->ofl_ossyminfo->os_shdr;

2477                dyn->d_tag = DT_SYMINFO;
2478                dyn->d_un.d_ptr = shdr->sh_addr;
2479                dyn++;
2480                dyn->d_tag = DT_SYMINSZ;
2481                dyn->d_un.d_val = shdr->sh_size;
2482                dyn++;
2483                dyn->d_tag = DT_SYMINENT;
2484                dyn->d_un.d_val = sizeof (Syminfo);
2485                dyn++;
2486        }
2487        if (ofl->ofl_osmove) {
2488                shdr = ofl->ofl_osmove->os_shdr;

2490                dyn->d_tag = DT_MOVETAB;
2491                dyn->d_un.d_val = shdr->sh_addr;
2492                dyn++;
2493                dyn->d_tag = DT_MOVESZ;
2494                dyn->d_un.d_val = shdr->sh_size;
2495                dyn++;
2496                dyn->d_tag = DT_MOVEENT;
2497                dyn->d_un.d_val = shdr->sh_entsize;
2498                dyn++;
```

```
2499                }
2500                if (ofl->ofl_regsymcnt) {
2501                        int     ndx;
2503                        for (ndx = 0; ndx < ofl->ofl_regsymsno; ndx++) {
2504                                if ((sdp = ofl->ofl_regsyms[ndx]) == NULL)
2505                                        continue;
2507                                dyn->d_tag = ld_targ.t_m.m_dt_register;
2508                                dyn->d_un.d_val = sdp->sd_symndx;
2509                                dyn++;
2510                        }
2511                }
2513                for (APLIST_TRAVERSE(ofl->ofl_rtldinfo, idx, sdp)) {
2514                        dyn->d_tag = DT_SUNW_RTLDINF;
2515                        dyn->d_un.d_ptr = sdp->sd_sym->st_value;
2516                        dyn++;
2517                }
2519                if (((sgp = ofl->ofl_osdynamic->os_sgdesc) != NULL) &&
2520                    (sgp->sg_phdr.p_flags & PF_W) && ofl->ofl_osinterp) {
2521                        dyn->d_tag = DT_DEBUG;
2522                        dyn->d_un.d_ptr = 0;
2523                        dyn++;
2524                }
2526                if (ofl->ofl_oscap) {
2527                        dyn->d_tag = DT_SUNW_CAP;
2528                        dyn->d_un.d_val = ofl->ofl_oscap->os_shdr->sh_addr;
2529                        dyn++;
2530                }
2531                if (ofl->ofl_oscapinfo) {
2532                        dyn->d_tag = DT_SUNW_CAPINFO;
2533                        dyn->d_un.d_val = ofl->ofl_oscapinfo->os_shdr->sh_addr;
2534                        dyn++;
2535                }
2536                if (ofl->ofl_oscapchain) {
2537                        shdr = ofl->ofl_oscapchain->os_shdr;
2539                        dyn->d_tag = DT_SUNW_CAPCHAIN;
2540                        dyn->d_un.d_val = shdr->sh_addr;
2541                        dyn++;
2542                        dyn->d_tag = DT_SUNW_CAPCHAINSZ;
2543                        dyn->d_un.d_val = shdr->sh_size;
2544                        dyn++;
2545                        dyn->d_tag = DT_SUNW_CAPCHAINENT;
2546                        dyn->d_un.d_val = shdr->sh_entsize;
2547                        dyn++;
2548                }
2550                if (ofl->ofl_aslr != 0) {
2551                        dyn->d_tag = DT_SUNW_ASLR;
2552                        dyn->d_un.d_val = (ofl->ofl_aslr == 1);
2553                        dyn++;
2554                }
2556                if (flags & FLG_OF_SYMBOLIC) {
2557                        dyn->d_tag = DT_SYMBOLIC;
2558                        dyn->d_un.d_val = 0;
2559                        dyn++;
2560                }
2561        }
2563        dyn->d_tag = DT_FLAGS;
2564        dyn->d_un.d_val = ofl->ofl_dtflags;
```



```
2565        dyn++;
2567        /*
2568         * If -Bdirect was specified, but some NODIRECT symbols were specified
2569         * via a mapfile, or -znodirect was used on the command line, then
2570         * clear the DF_1_DIRECT flag.  The resultant object will use per-symbol
2571         * direct bindings rather than be enabled for global direct bindings.
2572         *
2573         * If any no-direct bindings exist within this object, set the
2574         * DF_1_NODIRECT flag.  ld(1) recognizes this flag when processing
2575         * dependencies, and performs extra work to ensure that no direct
2576         * bindings are established to the no-direct symbols that exist
2577         * within these dependencies.
2578         */
2579        if (ofl->ofl_flags1 & FLG_OF1_NGLBDIR)
2580                ofl->ofl_dtflags_1 &= ~DF_1_DIRECT;
2581        if (ofl->ofl_flags1 & FLG_OF1_NDIRECT)
2582                ofl->ofl_dtflags_1 |= DF_1_NODIRECT;
2584        dyn->d_tag = DT_FLAGS_1;
2585        dyn->d_un.d_val = ofl->ofl_dtflags_1;
2586        dyn++;
2588        dyn->d_tag = DT_SUNW_STRPAD;
2589        dyn->d_un.d_val = DYNSTR_EXTRA_PAD;
2590        dyn++;
2592        dyn->d_tag = DT_SUNW_LDMACH;
2593        dyn->d_un.d_val = ld_sunw_ldmach();
2594        dyn++;
2596        (*ld_targ.t_mr.mr_mach_update_odynamic)(ofl, &dyn);
2598        for (cnt = 1 + DYNAMIC_EXTRA_ELTS; cnt--; dyn++) {
2599                dyn->d_tag = DT_NULL;
2600                dyn->d_un.d_val = 0;
2601        }
2603        /*
2604         * Ensure that we wrote the right number of entries. If not, we either
2605         * miscounted in make_dynamic(), or we did something wrong in this
2606         * function.
2607         */
2608        assert((ofl->ofl_osdynamic->os_shdr->sh_size /
2609            ofl->ofl_osdynamic->os_shdr->sh_entsize) ==
2610            ((uintptr_t)dyn - (uintptr_t)_dyn) / sizeof (*dyn));
2612        return (1);
2613 }
2615 /*
2616  * Build the version definition section
2617  */
2618 static int
2619 update_overdef(Ofl_desc *ofl)
2620 {
2621        Aliste          idx1;
2622        Ver_desc        *vdp, *_vdp;
2623        Verdef          *vdf, *_vdf;
2624        int             num = 0;
2625        Os_desc         *strosp;
2626        Str_tbl         *strtbl;
2628        /*
2629         * Determine which string table to use.
2630         */
```

```
2631         if (OFL_IS_STATIC_OBJ(ofl)) {
2632                 strtbl = ofl->ofl_strtab;
2633                 strosp = ofl->ofl_osstrtab;
2634         } else {
2635                 strtbl = ofl->ofl_dynstrtab;
2636                 strosp = ofl->ofl_osdynstr;
2637         }

2639         /*
2640          * Traverse the version descriptors and update the version structures
2641          * to point to the dynstr name in preparation for building the version
2642          * section structure.
2643          */
2644         for (APLIST_TRAVERSE(ofl->ofl_verdesc, idx1, vdp)) {
2645                 Sym_desc        *sdp;

2647                 if (vdp->vd_flags & VER_FLG_BASE) {
2648                         const char      *name = vdp->vd_name;
2649                         size_t          stoff;

2651                         /*
2652                          * Create a new string table entry to represent the base
2653                          * version name (there is no corresponding symbol for
2654                          * this).
2655                          */
2656                         (void) st_setstring(strtbl, name, &stoff);
2657                         /* LINTED */
2658                         vdp->vd_name = (const char *)stoff;
2659                 } else {
2660                         sdp = ld_sym_find(vdp->vd_name, vdp->vd_hash, 0, ofl);
2661                         /* LINTED */
2662                         vdp->vd_name = (const char *)
2663                             (uintptr_t)sdp->sd_sym->st_name;
2664                 }
2665         }

2667         _vdf = vdf = (Verdef *)ofl->ofl_osverdef->os_outdata->d_buf;

2669         /*
2670          * Traverse the version descriptors and update the version section to
2671          * reflect each version and its associated dependencies.
2672          */
2673         for (APLIST_TRAVERSE(ofl->ofl_verdesc, idx1, vdp)) {
2674                 Aliste          idx2;
2675                 Half            cnt = 1;
2676                 Verdaux         *vdap, *_vdap;

2678                 _vdap = vdap = (Verdaux *)(vdf + 1);

2680                 vdf->vd_version = VER_DEF_CURRENT;
2681                 vdf->vd_flags   = vdp->vd_flags & MSK_VER_USER;
2682                 vdf->vd_ndx     = vdp->vd_ndx;
2683                 vdf->vd_hash    = vdp->vd_hash;

2685                 /* LINTED */
2686                 vdap->vda_name = (uintptr_t)vdp->vd_name;
2687                 vdap++;
2688                 /* LINTED */
2689                 _vdap->vda_next = (Word)((uintptr_t)vdap - (uintptr_t)_vdap);

2691                 /*
2692                  * Traverse this versions dependency list generating the
2693                  * appropriate version dependency entries.
2694                  */
2695                 for (APLIST_TRAVERSE(vdp->vd_deps, idx2, _vdp)) {
2696                         /* LINTED */
```



```
2697                         vdap->vda_name = (uintptr_t)_vdp->vd_name;
2698                         _vdap = vdap;
2699                         vdap++, cnt++;
2700                         /* LINTED */
2701                         _vdap->vda_next = (Word)((uintptr_t)vdap -
2702                             (uintptr_t)_vdap);
2703                 }
2704                 _vdap->vda_next = 0;

2706                 /*
2707                  * Record the versions auxiliary array offset and the associated
2708                  * dependency count.
2709                  */
2710                 /* LINTED */
2711                 vdf->vd_aux = (Word)((uintptr_t)(vdf + 1) - (uintptr_t)vdf);
2712                 vdf->vd_cnt = cnt;

2714                 /*
2715                  * Record the next versions offset and update the version
2716                  * pointer.  Remember the previous version offset as the very
2717                  * last structures next pointer should be null.
2718                  */
2719                 _vdf = vdf;
2720                 vdf = (Verdef *)vdap, num++;
2721                 /* LINTED */
2722                 _vdf->vd_next = (Word)((uintptr_t)vdf - (uintptr_t)_vdf);
2723         }
2724         _vdf->vd_next = 0;

2726         /*
2727          * Record the string table association with the version definition
2728          * section, and the symbol table associated with the version symbol
2729          * table (the actual contents of the version symbol table are filled
2730          * in during symbol update).
2731          */
2732         /* LINTED */
2733         ofl->ofl_osverdef->os_shdr->sh_link = (Word)elf_ndxscn(strosp->os_scn);

2735         /*
2736          * The version definition sections ‘info’ field is used to indicate the
2737          * number of entries in this section.
2738          */
2739         ofl->ofl_osverdef->os_shdr->sh_info = num;

2741         return (1);
2742 }

2744 /*
2745  * Finish the version symbol index section
2746  */
2747 static void
2748 update_oversym(Ofl_desc *ofl)
2749 {
2750         Os_desc         *osp;

2752         /*
2753          * Record the symbol table associated with the version symbol table.
2754          * The contents of the version symbol table are filled in during
2755          * symbol update.
2756          */
2757         if (OFL_IS_STATIC_OBJ(ofl))
2758                 osp = ofl->ofl_ossymtab;
2759         else
2760                 osp = ofl->ofl_osdynsym;

2762         /* LINTED */
```

```
2763         ofl->ofl_osversym->os_shdr->sh_link = (Word)elf_ndxscn(osp->os_scn);
2764 }

2766 /*
2767  * Build the version needed section
2768  */
2769 static int
2770 update_overneed(Ofl_desc *ofl)
2771 {
2772         Aliste          idx1;
2773         Ifl_desc        *ifl;
2774         Verneed         *vnd, *_vnd;
2775         Os_desc         *strosp;
2776         Str_tbl         *strtbl;
2777         Word            num = 0;

2779         _vnd = vnd = (Verneed *)ofl->ofl_osverneed->os_outdata->d_buf;

2781         /*
2782          * Determine which string table is appropriate.
2783          */
2784         if (OFL_IS_STATIC_OBJ(ofl)) {
2785                 strosp = ofl->ofl_osstrtab;
2786                 strtbl = ofl->ofl_strtab;
2787         } else {
2788                 strosp = ofl->ofl_osdynstr;
2789                 strtbl = ofl->ofl_dynstrtab;
2790         }

2792         /*
2793          * Traverse the shared object list looking for dependencies that have
2794          * versions defined within them.
2795          */
2796         for (APLIST_TRAVERSE(ofl->ofl_sos, idx1, ifl)) {
2797                 Half            _cnt;
2798                 Word            cnt = 0;
2799                 Vernaux         *_vnap, *vnap;
2800                 size_t          stoff;

2802                 if (!(ifl->ifl_flags & FLG_IF_VERNEED))
2803                         continue;

2805                 vnd->vn_version = VER_NEED_CURRENT;

2807                 (void) st_setstring(strtbl, ifl->ifl_soname, &stoff);
2808                 vnd->vn_file = stoff;

2810                 _vnap = vnap = (Vernaux *)(vnd + 1);

2812                 /*
2813                  * Traverse the version index list recording
2814                  * each version as a needed dependency.
2815                  */
2816                 for (_cnt = 0; _cnt <= ifl->ifl_vercnt; _cnt++) {
2817                         Ver_index       *vip = &ifl->ifl_verndx[_cnt];

2819                         if (vip->vi_flags & FLG_VER_REFER) {
2820                                 (void) st_setstring(strtbl, vip->vi_name,
2821                                     &stoff);
2822                                 vnap->vna_name = stoff;

2824                                 if (vip->vi_desc) {
2825                                         vnap->vna_hash = vip->vi_desc->vd_hash;
2826                                         vnap->vna_flags =
2827                                             vip->vi_desc->vd_flags;
2828                                 } else {
```



```
2829                                         vnap->vna_hash = 0;
2830                                         vnap->vna_flags = 0;
2831                                 }
2832                                 vnap->vna_other = vip->vi_overndx;

2834                                 /*
2835                                  * If version A inherits version B, then
2836                                  * B is implicit in A. It suffices for ld.so.1
2837                                  * to verify A at runtime and skip B. The
2838                                  * version normalization process sets the INFO
2839                                  * flag for the versions we want ld.so.1 to
2840                                  * skip.
2841                                  */
2842                                 if (vip->vi_flags & VER_FLG_INFO)
2843                                         vnap->vna_flags |= VER_FLG_INFO;

2845                                 _vnap = vnap;
2846                                 vnap++, cnt++;
2847                                 _vnap->vna_next =
2848                                     /* LINTED */
2849                                     (Word)((uintptr_t)vnap - (uintptr_t)_vnap);
2850                         }
2851                 }

2853                 _vnap->vna_next = 0;

2855                 /*
2856                  * Record the versions auxiliary array offset and
2857                  * the associated dependency count.
2858                  */
2859                 /* LINTED */
2860                 vnd->vn_aux = (Word)((uintptr_t)(vnd + 1) - (uintptr_t)vnd);
2861                 /* LINTED */
2862                 vnd->vn_cnt = (Half)cnt;

2864                 /*
2865                  * Record the next versions offset and update the version
2866                  * pointer.  Remember the previous version offset as the very
2867                  * last structures next pointer should be null.
2868                  */
2869                 _vnd = vnd;
2870                 vnd = (Verneed *)vnap, num++;
2871                 /* LINTED */
2872                 _vnd->vn_next = (Word)((uintptr_t)vnd - (uintptr_t)_vnd);
2873         }
2874         _vnd->vn_next = 0;

2876         /*
2877          * Use sh_link to record the associated string table section, and
2878          * sh_info to indicate the number of entries contained in the section.
2879          */
2880         /* LINTED */
2881         ofl->ofl_osverneed->os_shdr->sh_link = (Word)elf_ndxscn(strosp->os_scn);
2882         ofl->ofl_osverneed->os_shdr->sh_info = num;

2884         return (1);
2885 }

2887 /*
2888  * Update syminfo section.
2889  */
2890 static uintptr_t
2891 update_osyminfo(Ofl_desc *ofl)
2892 {
2893         Os_desc         *symosp, *infosp = ofl->ofl_ossyminfo;
2894         Syminfo         *sip = infosp->os_outdata->d_buf;
```

```
2895         Shdr            *shdr = infosp->os_shdr;
2896         char            *strtab;
2897         Aliste          idx;
2898         Sym_desc        *sdp;
2899         Sfltr_desc      *sftp;

2901         if (ofl->ofl_flags & FLG_OF_RELOBJ) {
2902                 symosp = ofl->ofl_ossymtab;
2903                 strtab = ofl->ofl_osstrtab->os_outdata->d_buf;
2904         } else {
2905                 symosp = ofl->ofl_osdynsym;
2906                 strtab = ofl->ofl_osdynstr->os_outdata->d_buf;
2907         }

2909         /* LINTED */
2910         infosp->os_shdr->sh_link = (Word)elf_ndxscn(symosp->os_scn);
2911         if (ofl->ofl_osdynamic)
2912                 infosp->os_shdr->sh_info =
2913                     /* LINTED */
2914                     (Word)elf_ndxscn(ofl->ofl_osdynamic->os_scn);

2916         /*
2917          * Update any references with the index into the dynamic table.
2918          */
2919         for (APLIST_TRAVERSE(ofl->ofl_symdtent, idx, sdp))
2920                 sip[sdp->sd_symndx].si_boundto = sdp->sd_file->ifl_neededndx;

2922         /*
2923          * Update any filtee references with the index into the dynamic table.
2924          */
2925         for (ALIST_TRAVERSE(ofl->ofl_symfltrs, idx, sftp)) {
2926                 Dfltr_desc      *dftp;

2928                 dftp = alist_item(ofl->ofl_dtsfltrs, sftp->sft_idx);
2929                 sip[sftp->sft_sdp->sd_symndx].si_boundto = dftp->dft_ndx;
2930         }

2932         /*
2933          * Display debugging information about section.
2934          */
2935         DBG_CALL(Dbg_syminfo_title(ofl->ofl_lml));
2936         if (DBG_ENABLED) {
2937                 Word    _cnt, cnt = shdr->sh_size / shdr->sh_entsize;
2938                 Sym     *symtab = symosp->os_outdata->d_buf;
2939                 Dyn     *dyn;

2941                 if (ofl->ofl_osdynamic)
2942                         dyn = ofl->ofl_osdynamic->os_outdata->d_buf;
2943                 else
2944                         dyn = NULL;

2946                 for (_cnt = 1; _cnt < cnt; _cnt++) {
2947                         if (sip[_cnt].si_flags || sip[_cnt].si_boundto)
2948                                 /* LINTED */
2949                                 DBG_CALL(Dbg_syminfo_entry(ofl->ofl_lml, _cnt,
2950                                     &sip[_cnt], &symtab[_cnt], strtab, dyn));
2951                 }
2952         }
2953         return (1);
2954 }

2956 /*
2957  * Build the output elf header.
2958  */
2959 static uintptr_t
2960 update_oehdr(Ofl_desc * ofl)
```



```
2961 {
2962         Ehdr    *ehdr = ofl->ofl_nehdr;

2964         /*
2965          * If an entry point symbol has already been established (refer
2966          * sym_validate()) simply update the elf header entry point with the
2967          * symbols value.  If no entry point is defined it will have been filled
2968          * with the start address of the first section within the text segment
2969          * (refer update_outfile()).
2970          */
2971         if (ofl->ofl_entry)
2972                 ehdr->e_entry =
2973                     ((Sym_desc *)(ofl->ofl_entry))->sd_sym->st_value;

2975         ehdr->e_ident[EI_DATA] = ld_targ.t_m.m_data;
2976         ehdr->e_version = ofl->ofl_dehdr->e_version;

2978         /*
2979          * When generating a relocatable object under -z symbolcap, set the
2980          * e_machine to be generic, and remove any e_flags.  Input relocatable
2981          * objects may identify alternative e_machine (m.machplus) and e_flags
2982          * values.  However, the functions within the created output object
2983          * are selected at runtime using the capabilities mechanism, which
2984          * supersedes the e-machine and e_flags information.  Therefore,
2985          * e_machine and e_flag values are not propagated to the output object,
2986          * as these values might prevent the kernel from loading the object
2987          * before the runtime linker gets control.
2988          */
2989         if (ofl->ofl_flags & FLG_OF_OTOSCAP) {
2990                 ehdr->e_machine = ld_targ.t_m.m_mach;
2991                 ehdr->e_flags = 0;
2992         } else {
2993                 /*
2994                  * Note. it may be necessary to update the e_flags field in the
2995                  * machine dependent section.
2996                  */
2997                 ehdr->e_machine = ofl->ofl_dehdr->e_machine;
2998                 ehdr->e_flags = ofl->ofl_dehdr->e_flags;

3000                 if (ehdr->e_machine != ld_targ.t_m.m_mach) {
3001                         if (ehdr->e_machine != ld_targ.t_m.m_machplus)
3002                                 return (S_ERROR);
3003                         if ((ehdr->e_flags & ld_targ.t_m.m_flagsplus) == 0)
3004                                 return (S_ERROR);
3005                 }
3006         }

3008         if (ofl->ofl_flags & FLG_OF_SHAROBJ)
3009                 ehdr->e_type = ET_DYN;
3010         else if (ofl->ofl_flags & FLG_OF_RELOBJ)
3011                 ehdr->e_type = ET_REL;
3012         else
3013                 ehdr->e_type = ET_EXEC;

3015         return (1);
3016 }

3018 /*
3019  * Perform move table expansion.
3020  */
3021 static void
3022 expand_move(Ofl_desc *ofl, Sym_desc *sdp, Move *mvp)
3023 {
3024         Os_desc         *osp;
3025         uchar_t         *taddr, *taddr0;
3026         Sxword          offset;
```

```
3027         Half            cnt;
3028         uint_t          stride;

3030         osp = ofl->ofl_isparexpn->is_osdesc;
3031         offset = sdp->sd_sym->st_value - osp->os_shdr->sh_addr;

3033         taddr0 = taddr = osp->os_outdata->d_buf;
3034         taddr += offset;
3035         taddr = taddr + mvp->m_poffset;

3037         for (cnt = 0; cnt < mvp->m_repeat; cnt++) {
3038                 /* LINTED */
3039                 DBG_CALL(Dbg_move_expand(ofl->ofl_lml, mvp,
3040                     (Addr)(taddr - taddr0)));
3041                 stride = (uint_t)mvp->m_stride + 1;

3043                 /*
3044                  * Update the target address based upon the move entry size.
3045                  * This size was validated in ld_process_move().
3046                  */
3047                 /* LINTED */
3048                 switch (ELF_M_SIZE(mvp->m_info)) {
3049                 case 1:
3050                         /* LINTED */
3051                         *taddr = (uchar_t)mvp->m_value;
3052                         taddr += stride;
3053                         break;
3054                 case 2:
3055                         /* LINTED */
3056                         *((Half *)taddr) = (Half)mvp->m_value;
3057                         taddr += 2 * stride;
3058                         break;
3059                 case 4:
3060                         /* LINTED */
3061                         *((Word *)taddr) = (Word)mvp->m_value;
3062                         taddr += 4 * stride;
3063                         break;
3064                 case 8:
3065                         /* LINTED */
3066                         *((u_longlong_t *)taddr) = mvp->m_value;
3067                         taddr += 8 * stride;
3068                         break;
3069                 }
3070         }
3071 }

3073 /*
3074  * Update Move sections.
3075  */
3076 static void
3077 update_move(Ofl_desc *ofl)
3078 {
3079         Word            ndx = 0;
3080         ofl_flag_t      flags = ofl->ofl_flags;
3081         Move            *omvp;
3082         Aliste          idx1;
3083         Sym_desc        *sdp;

3085         /*
3086          * Determine the index of the symbol table that will be referenced by
3087          * the Move section.
3088          */
3089         if (OFL_ALLOW_DYNSYM(ofl))
3090                 /* LINTED */
3091                 ndx = (Word) elf_ndxscn(ofl->ofl_osdynsym->os_scn);
3092         else if (!(flags & FLG_OF_STRIP) || (flags & FLG_OF_RELOBJ))
```



```
3093                 /* LINTED */
3094                 ndx = (Word) elf_ndxscn(ofl->ofl_ossymtab->os_scn);

3096         /*
3097          * Update sh_link of the Move section, and point to the new Move data.
3098          */
3099         if (ofl->ofl_osmove) {
3100                 ofl->ofl_osmove->os_shdr->sh_link = ndx;
3101                 omvp = (Move *)ofl->ofl_osmove->os_outdata->d_buf;
3102         }

3104         /*
3105          * Update symbol entry index
3106          */
3107         for (APLIST_TRAVERSE(ofl->ofl_parsyms, idx1, sdp)) {
3108                 Aliste          idx2;
3109                 Mv_desc         *mdp;

3111                 /*
3112                  * Expand move table
3113                  */
3114                 if (sdp->sd_flags & FLG_SY_PAREXPN) {
3115                         const char      *str;

3117                         if (flags & FLG_OF_STATIC)
3118                                 str = MSG_INTL(MSG_PSYM_EXPREASON1);
3119                         else if (ofl->ofl_flags1 & FLG_OF1_NOPARTI)
3120                                 str = MSG_INTL(MSG_PSYM_EXPREASON2);
3121                         else
3122                                 str = MSG_INTL(MSG_PSYM_EXPREASON3);

3124                         DBG_CALL(Dbg_move_parexpn(ofl->ofl_lml,
3125                             sdp->sd_name, str));

3127                         for (ALIST_TRAVERSE(sdp->sd_move, idx2, mdp)) {
3128                                 DBG_CALL(Dbg_move_entry1(ofl->ofl_lml, 0,
3129                                     mdp->md_move, sdp));
3130                                 expand_move(ofl, sdp, mdp->md_move);
3131                         }
3132                         continue;
3133                 }

3135                 /*
3136                  * Process move table
3137                  */
3138                 DBG_CALL(Dbg_move_outmove(ofl->ofl_lml, sdp->sd_name));

3140                 for (ALIST_TRAVERSE(sdp->sd_move, idx2, mdp)) {
3141                         Move    *imvp;
3142                         int     idx = 1;
3143                         Sym     *sym;

3145                         imvp = mdp->md_move;
3146                         sym = sdp->sd_sym;

3148                         DBG_CALL(Dbg_move_entry1(ofl->ofl_lml, 1, imvp, sdp));

3150                         *omvp = *imvp;
3151                         if ((flags & FLG_OF_RELOBJ) == 0) {
3152                                 if (ELF_ST_BIND(sym->st_info) == STB_LOCAL) {
3153                                         Os_desc *osp = sdp->sd_isc->is_osdesc;
3154                                         Word    ndx = osp->os_identndx;

3156                                         omvp->m_info =
3157                                             /* LINTED */
3158                                             ELF_M_INFO(ndx, imvp->m_info);
```

```
3160                                    if (ELF_ST_TYPE(sym->st_info) !=
3161                                        STT_SECTION) {
3162                                            omvp->m_poffset =
3163                                                sym->st_value -
3164                                                osp->os_shdr->sh_addr +
3165                                                imvp->m_poffset;
3166                                    }
3167                            } else {
3168                                    omvp->m_info =
3169                                        /* LINTED */
3170                                        ELF_M_INFO(sdp->sd_symndx,
3171                                        imvp->m_info);
3172                            }
3173                    } else {
3174                            Boolean         isredloc = FALSE;
3175
3176                            if ((ELF_ST_BIND(sym->st_info) == STB_LOCAL) &&
3177                                (ofl->ofl_flags & FLG_OF_REDLSYM))
3178                                    isredloc = TRUE;
3179
3180                            if (isredloc && !(sdp->sd_move)) {
3181                                    Os_desc *osp = sdp->sd_isc->is_osdesc;
3182                                    Word    ndx = osp->os_identndx;
3183
3184                                    omvp->m_info =
3185                                        /* LINTED */
3186                                        ELF_M_INFO(ndx, imvp->m_info);
3187
3188                                    omvp->m_poffset += sym->st_value;
3189                            } else {
3190                                    if (isredloc)
3191                                            DBG_CALL(Dbg_syms_reduce(ofl,
3192                                                DBG_SYM_REDUCE_RETAIN,
3193                                                sdp, idx,
3194                                                ofl->ofl_osmove->os_name));
3195
3196                                    omvp->m_info =
3197                                        /* LINTED */
3198                                        ELF_M_INFO(sdp->sd_symndx,
3199                                        imvp->m_info);
3200                            }
3201                    }
3202
3203                    DBG_CALL(Dbg_move_entry1(ofl->ofl_lml, 0, omvp, sdp));
3204                    omvp++;
3205                    idx++;
3206            }
3207    }
3208 }
3209
3210 /*
3211  * Scan through the SHT_GROUP output sections.  Update their sh_link/sh_info
3212  * fields as well as the section contents.
3213  */
3214 static uintptr_t
3215 update_ogroup(Ofl_desc *ofl)
3216 {
3217         Aliste          idx;
3218         Os_desc         *osp;
3219         uintptr_t       error = 0;
3220
3221         for (APLIST_TRAVERSE(ofl->ofl_osgroups, idx, osp)) {
3222                 Is_desc         *isp;
3223                 Ifl_desc        *ifl;
3224                 Shdr            *shdr = osp->os_shdr;
```



```
3225                 Sym_desc        *sdp;
3226                 Xword           i, grpcnt;
3227                 Word            *gdata;
3228
3229                 /*
3230                  * Since input GROUP sections always create unique
3231                  * output GROUP sections - we know there is only one
3232                  * item on the list.
3233                  */
3234                 isp = ld_os_first_isdesc(osp);
3235
3236                 ifl = isp->is_file;
3237                 sdp = ifl->ifl_oldndx[isp->is_shdr->sh_info];
3238                 shdr->sh_link = (Word)elf_ndxscn(ofl->ofl_ossymtab->os_scn);
3239                 shdr->sh_info = sdp->sd_symndx;
3240
3241                 /*
3242                  * Scan through the group data section and update
3243                  * all of the links to new values.
3244                  */
3245                 grpcnt = shdr->sh_size / shdr->sh_entsize;
3246                 gdata = (Word *)osp->os_outdata->d_buf;
3247
3248                 for (i = 1; i < grpcnt; i++) {
3249                         Os_desc *_osp;
3250                         Is_desc *_isp = ifl->ifl_isdesc[gdata[i]];
3251
3252                         /*
3253                          * If the referenced section didn't make it to the
3254                          * output file - just zero out the entry.
3255                          */
3256                         if ((_osp = _isp->is_osdesc) == NULL)
3257                                 gdata[i] = 0;
3258                         else
3259                                 gdata[i] = (Word)elf_ndxscn(_osp->os_scn);
3260                 }
3261         }
3262         return (error);
3263 }
3264
3265 static void
3266 update_ostrtab(Os_desc *osp, Str_tbl *stp, uint_t extra)
3267 {
3268         Elf_Data        *data;
3269
3270         if (osp == NULL)
3271                 return;
3272
3273         data = osp->os_outdata;
3274         assert(data->d_size == (st_getstrtab_sz(stp) + extra));
3275         (void) st_setstrbuf(stp, data->d_buf, data->d_size - extra);
3276         /* If leaving an extra hole at the end, zero it */
3277         if (extra > 0)
3278                 (void) memset((char *)data->d_buf + data->d_size - extra,
3279                     0x0, extra);
3280 }
3281
3282 /*
3283  * Update capabilities information.
3284  *
3285  * If string table capabilities exist, then the associated string must be
3286  * translated into an offset into the string table.
3287  */
3288 static void
3289 update_oscap(Ofl_desc *ofl)
3290 {
```

```
3291         Os_desc         *strosp, *cosp;
3292         Cap             *cap;
3293         Str_tbl         *strtbl;
3294         Capstr          *capstr;
3295         size_t          stoff;
3296         Aliste          idx1;

3298         /*
3299          * Determine which symbol table or string table is appropriate.
3300          */
3301         if (OFL_IS_STATIC_OBJ(ofl)) {
3302                 strosp = ofl->ofl_osstrtab;
3303                 strtbl = ofl->ofl_strtab;
3304         } else {
3305                 strosp = ofl->ofl_osdynstr;
3306                 strtbl = ofl->ofl_dynstrtab;
3307         }

3309         /*
3310          * If symbol capabilities exist, set the sh_link field of the .SUNW_cap
3311          * section to the .SUNW_capinfo section.
3312          */
3313         if (ofl->ofl_oscapinfo) {
3314                 cosp = ofl->ofl_oscap;
3315                 cosp->os_shdr->sh_link =
3316                     (Word)elf_ndxscn(ofl->ofl_oscapinfo->os_scn);
3317         }

3319         /*
3320          * If there are capability strings to process, set the sh_info
3321          * field of the .SUNW_cap section to the associated string table, and
3322          * proceed to process any CA_SUNW_PLAT entries.
3323          */
3324         if ((ofl->ofl_flags & FLG_OF_CAPSTRS) == 0)
3325                 return;

3327         cosp = ofl->ofl_oscap;
3328         cosp->os_shdr->sh_info = (Word)elf_ndxscn(strosp->os_scn);

3330         cap = ofl->ofl_oscap->os_outdata->d_buf;

3332         /*
3333          * Determine whether an object capability identifier, or object
3334          * machine/platform capabilities exists.
3335          */
3336         capstr = &ofl->ofl_ocapset.oc_id;
3337         if (capstr->cs_str) {
3338                 (void) st_setstring(strtbl, capstr->cs_str, &stoff);
3339                 cap[capstr->cs_ndx].c_un.c_ptr = stoff;
3340         }
3341         for (ALIST_TRAVERSE(ofl->ofl_ocapset.oc_plat.cl_val, idx1, capstr)) {
3342                 (void) st_setstring(strtbl, capstr->cs_str, &stoff);
3343                 cap[capstr->cs_ndx].c_un.c_ptr = stoff;
3344         }
3345         for (ALIST_TRAVERSE(ofl->ofl_ocapset.oc_mach.cl_val, idx1, capstr)) {
3346                 (void) st_setstring(strtbl, capstr->cs_str, &stoff);
3347                 cap[capstr->cs_ndx].c_un.c_ptr = stoff;
3348         }

3350         /*
3351          * Determine any symbol capability identifiers, or machine/platform
3352          * capabilities.
3353          */
3354         if (ofl->ofl_capgroups) {
3355                 Cap_group       *cgp;
```



```
3357                 for (APLIST_TRAVERSE(ofl->ofl_capgroups, idx1, cgp)) {
3358                         Objcapset       *ocapset = &cgp->cg_set;
3359                         Aliste          idx2;

3361                         capstr = &ocapset->oc_id;
3362                         if (capstr->cs_str) {
3363                                 (void) st_setstring(strtbl, capstr->cs_str,
3364                                     &stoff);
3365                                 cap[capstr->cs_ndx].c_un.c_ptr = stoff;
3366                         }
3367                         for (ALIST_TRAVERSE(ocapset->oc_plat.cl_val, idx2,
3368                             capstr)) {
3369                                 (void) st_setstring(strtbl, capstr->cs_str,
3370                                     &stoff);
3371                                 cap[capstr->cs_ndx].c_un.c_ptr = stoff;
3372                         }
3373                         for (ALIST_TRAVERSE(ocapset->oc_mach.cl_val, idx2,
3374                             capstr)) {
3375                                 (void) st_setstring(strtbl, capstr->cs_str,
3376                                     &stoff);
3377                                 cap[capstr->cs_ndx].c_un.c_ptr = stoff;
3378                         }
3379                 }
3380         }
3381 }

3383 /*
3384  * Update the .SUNW_capinfo, and possibly the .SUNW_capchain sections.
3385  */
3386 static void
3387 update_oscapinfo(Ofl_desc *ofl)
3388 {
3389         Os_desc         *symosp, *ciosp, *ccosp = NULL;
3390         Capinfo         *ocapinfo;
3391         Capchain        *ocapchain;
3392         Cap_avlnode     *cav;
3393         Word            chainndx = 0;

3395         /*
3396          * Determine which symbol table is appropriate.
3397          */
3398         if (OFL_IS_STATIC_OBJ(ofl))
3399                 symosp = ofl->ofl_ossymtab;
3400         else
3401                 symosp = ofl->ofl_osdynsym;

3403         /*
3404          * Update the .SUNW_capinfo sh_link to point to the appropriate symbol
3405          * table section.  If we're creating a dynamic object, the
3406          * .SUNW_capinfo sh_info is updated to point to the .SUNW_capchain
3407          * section.
3408          */
3409         ciosp = ofl->ofl_oscapinfo;
3410         ciosp->os_shdr->sh_link = (Word)elf_ndxscn(symosp->os_scn);

3412         if (OFL_IS_STATIC_OBJ(ofl) == 0) {
3413                 ccosp = ofl->ofl_oscapchain;
3414                 ciosp->os_shdr->sh_info = (Word)elf_ndxscn(ccosp->os_scn);
3415         }

3417         /*
3418          * Establish the data for each section.  The first element of each
3419          * section defines the section's version number.
3420          */
3421         ocapinfo = ciosp->os_outdata->d_buf;
3422         ocapinfo[0] = CAPINFO_CURRENT;
```

```
3423         if (ccosp) {
3424                 ocapchain = ccosp->os_outdata->d_buf;
3425                 ocapchain[chainndx++] = CAPCHAIN_CURRENT;
3426         }
3427
3428         /*
3429          * Traverse all capabilities families.  Each member has a .SUNW_capinfo
3430          * assignment.  The .SUNW_capinfo entry differs for relocatable objects
3431          * and dynamic objects.
3432          *
3433          * Relocatable objects:
3434          *                      ELF_C_GROUP             ELF_C_SYM
3435          *
3436          * Family lead:         CAPINFO_SUNW_GLOB       lead symbol index
3437          * Family lead alias:   CAPINFO_SUNW_GLOB       lead symbol index
3438          * Family member:       .SUNW_cap index         lead symbol index
3439          *
3440          * Dynamic objects:
3441          *                      ELF_C_GROUP             ELF_C_SYM
3442          *
3443          * Family lead:         CAPINFO_SUNW_GLOB       .SUNW_capchain index
3444          * Family lead alias:   CAPINFO_SUNW_GLOB       .SUNW_capchain index
3445          * Family member:       .SUNW_cap index         lead symbol index
3446          *
3447          * The ELF_C_GROUP field identifies a capabilities symbol.  Lead
3448          * capability symbols, and lead capability aliases are identified by
3449          * a CAPINFO_SUNW_GLOB group identifier.  For family members, the
3450          * ELF_C_GROUP provides an index to the associate capabilities group
3451          * (i.e, an index into the SUNW_cap section that defines a group).
3452          *
3453          * For relocatable objects, the ELF_C_SYM field identifies the lead
3454          * capability symbol.  For the lead symbol itself, the .SUNW_capinfo
3455          * index is the same as the ELF_C_SYM value.  For lead alias symbols,
3456          * the .SUNW_capinfo index differs from the ELF_C_SYM value.  This
3457          * differentiation of CAPINFO_SUNW_GLOB symbols allows ld(1) to
3458          * identify, and propagate lead alias symbols.  For example, the lead
3459          * capability symbol memcpy() would have the ELF_C_SYM for memcpy(),
3460          * and the lead alias _memcpy() would also have the ELF_C_SYM for
3461          * memcpy().
3462          *
3463          * For dynamic objects, both a lead capability symbol, and alias symbol
3464          * would have a ELF_C_SYM value that represents the same capability
3465          * chain index.  The capability chain allows ld.so.1 to traverse a
3466          * family chain for a given lead symbol, and select the most appropriate
3467          * family member.  The .SUNW_capchain array contains a series of symbol
3468          * indexes for each family member:
3469          *
3470          *    chaincap[n]  chaincap[n + 1]  chaincap[n + 2]  chaincap[n + x]
3471          *      foo() ndx    foo%x() ndx        foo%y() ndx     0
3472          *
3473          * For family members, the ELF_C_SYM value associates the capability
3474          * members with their family lead symbol.  This association, although
3475          * unused within a dynamic object, allows ld(1) to identify, and
3476          * propagate family members when processing relocatable objects.
3477          */
3478         for (cav = avl_first(ofl->ofl_capfamilies); cav;
3479             cav = AVL_NEXT(ofl->ofl_capfamilies, cav)) {
3480                 Cap_sym         *csp;
3481                 Aliste          idx;
3482                 Sym_desc        *asdp, *lsdp = cav->cn_symavlnode.sav_sdp;
3483
3484                 if (ccosp) {
3485                         /*
3486                          * For a dynamic object, identify this lead symbol, and
3487                          * point it to the head of a capability chain.  Set the
3488                          * head of the capability chain to the same lead symbol.
```



```
3489                          */
3490                         ocapinfo[lsdp->sd_symndx] =
3491                             ELF_C_INFO(chainndx, CAPINFO_SUNW_GLOB);
3492                         ocapchain[chainndx] = lsdp->sd_symndx;
3493                 } else {
3494                         /*
3495                          * For a relocatable object, identify this lead symbol,
3496                          * and set the lead symbol index to itself.
3497                          */
3498                         ocapinfo[lsdp->sd_symndx] =
3499                             ELF_C_INFO(lsdp->sd_symndx, CAPINFO_SUNW_GLOB);
3500                 }
3501
3502                 /*
3503                  * Gather any lead symbol aliases.
3504                  */
3505                 for (APLIST_TRAVERSE(cav->cn_aliases, idx, asdp)) {
3506                         if (ccosp) {
3507                                 /*
3508                                  * For a dynamic object, identify this lead
3509                                  * alias symbol, and point it to the same
3510                                  * capability chain index as the lead symbol.
3511                                  */
3512                                 ocapinfo[asdp->sd_symndx] =
3513                                     ELF_C_INFO(chainndx, CAPINFO_SUNW_GLOB);
3514                         } else {
3515                                 /*
3516                                  * For a relocatable object, identify this lead
3517                                  * alias symbol, and set the lead symbol index
3518                                  * to the lead symbol.
3519                                  */
3520                                 ocapinfo[asdp->sd_symndx] =
3521                                     ELF_C_INFO(lsdp->sd_symndx,
3522                                     CAPINFO_SUNW_GLOB);
3523                         }
3524                 }
3525
3526                 chainndx++;
3527
3528                 /*
3529                  * Gather the family members.
3530                  */
3531                 for (APLIST_TRAVERSE(cav->cn_members, idx, csp)) {
3532                         Sym_desc        *msdp = csp->cs_sdp;
3533
3534                         /*
3535                          * Identify the members capability group, and the lead
3536                          * symbol of the family this symbol is a member of.
3537                          */
3538                         ocapinfo[msdp->sd_symndx] =
3539                             ELF_C_INFO(lsdp->sd_symndx, csp->cs_group->cg_ndx);
3540                         if (ccosp) {
3541                                 /*
3542                                  * For a dynamic object, set the next capability
3543                                  * chain to point to this family member.
3544                                  */
3545                                 ocapchain[chainndx++] = msdp->sd_symndx;
3546                         }
3547                 }
3548
3549                 /*
3550                  * Any chain of family members is terminated with a 0 element.
3551                  */
3552                 if (ccosp)
3553                         ocapchain[chainndx++] = 0;
3554         }
```

```
3555 }
3556
3557 /*
3558  * Translate the shdr->sh_{link, info} from its input section value to that
3559  * of the corresponding shdr->sh_{link, info} output section value.
3560  */
3561 static Word
3562 translate_link(Ofl_desc *ofl, Os_desc *osp, Word link, const char *msg)
3563 {
3564         Is_desc         *isp;
3565         Ifl_desc        *ifl;
3566
3567         /*
3568          * Don't translate the special section numbers.
3569          */
3570         if (link >= SHN_LORESERVE)
3571                 return (link);
3572
3573         /*
3574          * Does this output section translate back to an input file.  If not
3575          * then there is no translation to do.  In this case we will assume that
3576          * if sh_link has a value, it's the right value.
3577          */
3578         isp = ld_os_first_isdesc(osp);
3579         if ((ifl = isp->is_file) == NULL)
3580                 return (link);
3581
3582         /*
3583          * Sanity check to make sure that the sh_{link, info} value
3584          * is within range for the input file.
3585          */
3586         if (link >= ifl->ifl_shnum) {
3587                 ld_eprintf(ofl, ERR_WARNING, msg, ifl->ifl_name,
3588                     EC_WORD(isp->is_scnndx), isp->is_name, EC_XWORD(link));
3589                 return (link);
3590         }
3591
3592         /*
3593          * Follow the link to the input section.
3594          */
3595         if ((isp = ifl->ifl_isdesc[link]) == NULL)
3596                 return (0);
3597         if ((osp = isp->is_osdesc) == NULL)
3598                 return (0);
3599
3600         /* LINTED */
3601         return ((Word)elf_ndxscn(osp->os_scn));
3602 }
3603
3604 /*
3605  * Having created all of the necessary sections, segments, and associated
3606  * headers, fill in the program headers and update any other data in the
3607  * output image.  Some general rules:
3608  *
3609  *  -   If an interpreter is required always generate a PT_PHDR entry as
3610  *      well.  It is this entry that triggers the kernel into passing the
3611  *      interpreter an aux vector instead of just a file descriptor.
3612  *
3613  *  -   When generating an image that will be interpreted (ie. a dynamic
3614  *      executable, a shared object, or a static executable that has been
3615  *      provided with an interpreter - weird, but possible), make the initial
3616  *      loadable segment include both the ehdr and phdr[].  Both of these
3617  *      tables are used by the interpreter therefore it seems more intuitive
3618  *      to explicitly defined them as part of the mapped image rather than
3619  *      relying on page rounding by the interpreter to allow their access.
3620  *
```



```
3621  *  -   When generating a static image that does not require an interpreter
3622  *      have the first loadable segment indicate the address of the first
3623  *      .section as the start address (things like /kernel/unix and ufsboot
3624  *      expect this behavior).
3625  */
3626 uintptr_t
3627 ld_update_outfile(Ofl_desc *ofl)
3628 {
3629         Addr            size, etext, vaddr;
3630         Sg_desc         *sgp;
3631         Sg_desc         *dtracesgp = NULL, *capsgp = NULL, *intpsgp = NULL;
3632         Os_desc         *osp;
3633         int             phdrndx = 0, segndx = -1, secndx, intppndx, intpsndx;
3634         int             dtracepndx, dtracesndx, cappndx, capsndx;
3635         Ehdr            *ehdr = ofl->ofl_nehdr;
3636         Shdr            *hshdr;
3637         Phdr            *_phdr = NULL;
3638         Word            phdrsz = (ehdr->e_phnum * ehdr->e_phentsize), shscnndx;
3639         ofl_flag_t      flags = ofl->ofl_flags;
3640         Word            ehdrsz = ehdr->e_ehsize;
3641         Boolean         nobits;
3642         Off             offset;
3643         Aliste          idx1;
3644
3645         /*
3646          * Initialize the starting address for the first segment.  Executables
3647          * have different starting addresses depending upon the target ABI,
3648          * where as shared objects have a starting address of 0.  If this is
3649          * a 64-bit executable that is being constructed to run in a restricted
3650          * address space, use an alternative origin that will provide more free
3651          * address space for the the eventual process.
3652          */
3653         if (ofl->ofl_flags & FLG_OF_EXEC) {
3654 #if     defined(_ELF64)
3655                 if (ofl->ofl_ocapset.oc_sf_1.cm_val & SF1_SUNW_ADDR32)
3656                         vaddr = ld_targ.t_m.m_segm_aorigin;
3657                 else
3658 #endif
3659                         vaddr = ld_targ.t_m.m_segm_origin;
3660         } else
3661                 vaddr = 0;
3662
3663         /*
3664          * Loop through the segment descriptors and pick out what we need.
3665          */
3666         DBG_CALL(Dbg_seg_title(ofl->ofl_lml));
3667         for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
3668                 Phdr            *phdr = &(sgp->sg_phdr);
3669                 Xword           p_align;
3670                 Aliste          idx2;
3671                 Sym_desc        *sdp;
3672
3673                 segndx++;
3674
3675                 /*
3676                  * If an interpreter is required generate a PT_INTERP and
3677                  * PT_PHDR program header entry.  The PT_PHDR entry describes
3678                  * the program header table itself.  This information will be
3679                  * passed via the aux vector to the interpreter (ld.so.1).
3680                  * The program header array is actually part of the first
3681                  * loadable segment (and the PT_PHDR entry is the first entry),
3682                  * therefore its virtual address isn't known until the first
3683                  * loadable segment is processed.
3684                  */
3685                 if (phdr->p_type == PT_PHDR) {
3686                         if (ofl->ofl_osinterp) {
```

```
3687                                phdr->p_offset = ehdr->e_phoff;
3688                                phdr->p_filesz = phdr->p_memsz = phdrsz;

3690                                DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3691                                ofl->ofl_phdr[phdrndx++] = *phdr;
3692                        }
3693                        continue;
3694                }
3695                if (phdr->p_type == PT_INTERP) {
3696                        if (ofl->ofl_osinterp) {
3697                                intpsgp = sgp;
3698                                intpsndx = segndx;
3699                                intppndx = phdrndx++;
3700                        }
3701                        continue;
3702                }

3704                /*
3705                 * If we are creating a PT_SUNWDTRACE segment, remember where
3706                 * the program header is.  The header values are assigned after
3707                 * update_osym() has completed and the symbol table addresses
3708                 * have been updated.
3709                 */
3710                if (phdr->p_type == PT_SUNWDTRACE) {
3711                        if (ofl->ofl_dtracesym &&
3712                            ((flags & FLG_OF_RELOBJ) == 0)) {
3713                                dtracesgp = sgp;
3714                                dtracesndx = segndx;
3715                                dtracepndx = phdrndx++;
3716                        }
3717                        continue;
3718                }

3720                /*
3721                 * If a hardware/software capabilities section is required,
3722                 * generate the PT_SUNWCAP header.  Note, as this comes before
3723                 * the first loadable segment, we don't yet know its real
3724                 * virtual address.  This is updated later.
3725                 */
3726                if (phdr->p_type == PT_SUNWCAP) {
3727                        if (ofl->ofl_oscap && (ofl->ofl_flags & FLG_OF_PTCAP) &&
3728                            ((flags & FLG_OF_RELOBJ) == 0)) {
3729                                capsgp = sgp;
3730                                capsndx = segndx;
3731                                cappndx = phdrndx++;
3732                        }
3733                        continue;
3734                }

3736                /*
3737                 * As the dynamic program header occurs after the loadable
3738                 * headers in the segment descriptor table, all the address
3739                 * information for the .dynamic output section will have been
3740                 * figured out by now.
3741                 */
3742                if (phdr->p_type == PT_DYNAMIC) {
3743                        if (OFL_ALLOW_DYNSYM(ofl)) {
3744                                Shdr    *shdr = ofl->ofl_osdynamic->os_shdr;

3746                                phdr->p_vaddr = shdr->sh_addr;
3747                                phdr->p_offset = shdr->sh_offset;
3748                                phdr->p_filesz = shdr->sh_size;
3749                                phdr->p_flags = ld_targ.t_m.m_dataseg_perm;

3751                                DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3752                                ofl->ofl_phdr[phdrndx++] = *phdr;
```



```
3753                        }
3754                        continue;
3755                }

3757                /*
3758                 * As the unwind (.eh_frame_hdr) program header occurs after
3759                 * the loadable headers in the segment descriptor table, all
3760                 * the address information for the .eh_frame output section
3761                 * will have been figured out by now.
3762                 */
3763                if (phdr->p_type == PT_SUNW_UNWIND) {
3764                        Shdr        *shdr;

3766                        if (ofl->ofl_unwindhdr == NULL)
3767                                continue;

3769                        shdr = ofl->ofl_unwindhdr->os_shdr;

3771                        phdr->p_flags = PF_R;
3772                        phdr->p_vaddr = shdr->sh_addr;
3773                        phdr->p_memsz = shdr->sh_size;
3774                        phdr->p_filesz = shdr->sh_size;
3775                        phdr->p_offset = shdr->sh_offset;
3776                        phdr->p_align = shdr->sh_addralign;
3777                        phdr->p_paddr = 0;
3778                        ofl->ofl_phdr[phdrndx++] = *phdr;
3779                        continue;
3780                }

3782                /*
3783                 * The sunwstack program is used to convey non-default
3784                 * flags for the process stack. Only emit it if it would
3785                 * change the default.
3786                 */
3787                if (phdr->p_type == PT_SUNWSTACK) {
3788                        if (((flags & FLG_OF_RELOBJ) == 0) &&
3789                            ((sgp->sg_flags & FLG_SG_DISABLED) == 0))
3790                                ofl->ofl_phdr[phdrndx++] = *phdr;
3791                        continue;
3792                }

3794                /*
3795                 * As the TLS program header occurs after the loadable
3796                 * headers in the segment descriptor table, all the address
3797                 * information for the .tls output section will have been
3798                 * figured out by now.
3799                 */
3800                if (phdr->p_type == PT_TLS) {
3801                        Os_desc         *tlsosp;
3802                        Shdr            *lastfileshdr = NULL;
3803                        Shdr            *firstshdr = NULL, *lastshdr;
3804                        Aliste          idx;

3806                        if (ofl->ofl_ostlsseg == NULL)
3807                                continue;

3809                        /*
3810                         * Scan the output sections that have contributed TLS.
3811                         * Remember the first and last so as to determine the
3812                         * TLS memory size requirement.  Remember the last
3813                         * progbits section to determine the TLS data
3814                         * contribution, which determines the TLS program
3815                         * header filesz.
3816                         */
3817                        for (APLIST_TRAVERSE(ofl->ofl_ostlsseg, idx, tlsosp)) {
3818                                Shdr    *tlsshdr = tlsosp->os_shdr;
```

```
3820                            if (firstshdr == NULL)
3821                                    firstshdr = tlsshdr;
3822                            if (tlsshdr->sh_type != SHT_NOBITS)
3823                                    lastfileshdr = tlsshdr;
3824                            lastshdr = tlsshdr;
3825                    }

3827                    phdr->p_flags = PF_R | PF_W;
3828                    phdr->p_vaddr = firstshdr->sh_addr;
3829                    phdr->p_offset = firstshdr->sh_offset;
3830                    phdr->p_align = firstshdr->sh_addralign;

3832                    /*
3833                     * Determine the initialized TLS data size.  This
3834                     * address range is from the start of the TLS segment
3835                     * to the end of the last piece of initialized data.
3836                     */
3837                    if (lastfileshdr)
3838                            phdr->p_filesz = lastfileshdr->sh_offset +
3839                                lastfileshdr->sh_size - phdr->p_offset;
3840                    else
3841                            phdr->p_filesz = 0;

3843                    /*
3844                     * Determine the total TLS memory size.  This includes
3845                     * all TLS data and TLS uninitialized data.  This
3846                     * address range is from the start of the TLS segment
3847                     * to the memory address of the last piece of
3848                     * uninitialized data.
3849                     */
3850                    phdr->p_memsz = lastshdr->sh_addr +
3851                        lastshdr->sh_size - phdr->p_vaddr;

3853                    DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3854                    ofl->ofl_phdr[phdrndx] = *phdr;
3855                    ofl->ofl_tlsphdr = &ofl->ofl_phdr[phdrndx++];
3856                    continue;
3857            }

3859            /*
3860             * If this is an empty segment declaration, it will occur after
3861             * all other loadable segments.  As empty segments can be
3862             * defined with fixed addresses, make sure that no loadable
3863             * segments overlap.  This might occur as the object evolves
3864             * and the loadable segments grow, thus encroaching upon an
3865             * existing segment reservation.
3866             *
3867             * Segments are only created for dynamic objects, thus this
3868             * checking can be skipped when building a relocatable object.
3869             */
3870            if (!(flags & FLG_OF_RELOBJ) &&
3871                (sgp->sg_flags & FLG_SG_EMPTY)) {
3872                    int     i;
3873                    Addr    v_e;

3875                    vaddr = phdr->p_vaddr;
3876                    phdr->p_memsz = sgp->sg_length;
3877                    DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3878                    ofl->ofl_phdr[phdrndx++] = *phdr;

3880                    if (phdr->p_type != PT_LOAD)
3881                            continue;

3883                    v_e = vaddr + phdr->p_memsz;
```



```
3885                    /*
3886                     * Check overlaps
3887                     */
3888                    for (i = 0; i < phdrndx - 1; i++) {
3889                            Addr    p_s = (ofl->ofl_phdr[i]).p_vaddr;
3890                            Addr    p_e;

3892                            if ((ofl->ofl_phdr[i]).p_type != PT_LOAD)
3893                                    continue;

3895                            p_e = p_s + (ofl->ofl_phdr[i]).p_memsz;
3896                            if (((p_s <= vaddr) && (p_e > vaddr)) ||
3897                                ((vaddr <= p_s) && (v_e > p_s)))
3898                                    ld_eprintf(ofl, ERR_WARNING,
3899                                        MSG_INTL(MSG_UPD_SEGOVERLAP),
3900                                        ofl->ofl_name, EC_ADDR(p_e),
3901                                        sgp->sg_name, EC_ADDR(vaddr));
3902                    }
3903                    continue;
3904            }

3906            /*
3907             * Having processed any of the special program headers any
3908             * remaining headers will be built to express individual
3909             * segments.  Segments are only built if they have output
3910             * section descriptors associated with them (ie. some form of
3911             * input section has been matched to this segment).
3912             */
3913            if (sgp->sg_osdescs == NULL)
3914                    continue;

3916            /*
3917             * Determine the segments offset and size from the section
3918             * information provided from elf_update().
3919             * Allow for multiple NOBITS sections.
3920             */
3921            osp = sgp->sg_osdescs->apl_data[0];
3922            hshdr = osp->os_shdr;

3924            phdr->p_filesz = 0;
3925            phdr->p_memsz = 0;
3926            phdr->p_offset = offset = hshdr->sh_offset;

3928            nobits = ((hshdr->sh_type == SHT_NOBITS) &&
3929                ((sgp->sg_flags & FLG_SG_PHREQ) == 0));

3931            for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
3932                    Shdr    *shdr = osp->os_shdr;

3934                    p_align = 0;
3935                    if (shdr->sh_addralign > p_align)
3936                            p_align = shdr->sh_addralign;

3938                    offset = (Off)S_ROUND(offset, shdr->sh_addralign);
3939                    offset += shdr->sh_size;

3941                    if (shdr->sh_type != SHT_NOBITS) {
3942                            if (nobits) {
3943                                    ld_eprintf(ofl, ERR_FATAL,
3944                                        MSG_INTL(MSG_UPD_NOBITS));
3945                                    return (S_ERROR);
3946                            }
3947                            phdr->p_filesz = offset - phdr->p_offset;
3948                    } else if ((sgp->sg_flags & FLG_SG_PHREQ) == 0)
3949                            nobits = TRUE;
3950            }
```

```
3951                phdr->p_memsz = offset - hshdr->sh_offset;
3952
3953                /*
3954                 * If this is the first loadable segment of a dynamic object,
3955                 * or an interpreter has been specified (a static object built
3956                 * with an interpreter will still be given a PT_HDR entry), then
3957                 * compensate for the elf header and program header array.  Both
3958                 * of these are actually part of the loadable segment as they
3959                 * may be inspected by the interpreter.  Adjust the segments
3960                 * size and offset accordingly.
3961                 */
3962                if ((_phdr == NULL) && (phdr->p_type == PT_LOAD) &&
3963                    ((ofl->ofl_osinterp) || (flags & FLG_OF_DYNAMIC)) &&
3964                    (!(ofl->ofl_dtflags_1 & DF_1_NOHDR))) {
3965                        size = (Addr)S_ROUND((phdrsz + ehdrsz),
3966                            hshdr->sh_addralign);
3967                        phdr->p_offset -= size;
3968                        phdr->p_filesz += size;
3969                        phdr->p_memsz += size;
3970                }
3971
3972                /*
3973                 * If segment size symbols are required (specified via a
3974                 * mapfile) update their value.
3975                 */
3976                for (APLIST_TRAVERSE(sgp->sg_sizesym, idx2, sdp))
3977                        sdp->sd_sym->st_value = phdr->p_memsz;
3978
3979                /*
3980                 * If no file content has been assigned to this segment (it
3981                 * only contains no-bits sections), then reset the offset for
3982                 * consistency.
3983                 */
3984                if (phdr->p_filesz == 0)
3985                        phdr->p_offset = 0;
3986
3987                /*
3988                 * If a virtual address has been specified for this segment
3989                 * from a mapfile use it and make sure the previous segment
3990                 * does not run into this segment.
3991                 */
3992                if (phdr->p_type == PT_LOAD) {
3993                        if ((sgp->sg_flags & FLG_SG_P_VADDR)) {
3994                                if (_phdr && (vaddr > phdr->p_vaddr) &&
3995                                    (phdr->p_type == PT_LOAD))
3996                                        ld_eprintf(ofl, ERR_WARNING,
3997                                            MSG_INTL(MSG_UPD_SEGOVERLAP),
3998                                            ofl->ofl_name, EC_ADDR(vaddr),
3999                                            sgp->sg_name,
4000                                            EC_ADDR(phdr->p_vaddr));
4001                                vaddr = phdr->p_vaddr;
4002                                phdr->p_align = 0;
4003                        } else {
4004                                vaddr = phdr->p_vaddr =
4005                                    (Addr)S_ROUND(vaddr, phdr->p_align);
4006                        }
4007                }
4008
4009                /*
4010                 * Adjust the address offset and p_align if needed.
4011                 */
4012                if (((sgp->sg_flags & FLG_SG_P_VADDR) == 0) &&
4013                    ((ofl->ofl_dtflags_1 & DF_1_NOHDR) == 0)) {
4014                        if (phdr->p_align != 0)
4015                                vaddr += phdr->p_offset % phdr->p_align;
4016                        else
```



```
4017                                vaddr += phdr->p_offset;
4018                        phdr->p_vaddr = vaddr;
4019                }
4020
4021                /*
4022                 * If an interpreter is required set the virtual address of the
4023                 * PT_PHDR program header now that we know the virtual address
4024                 * of the loadable segment that contains it.  Update the
4025                 * PT_SUNWCAP header similarly.
4026                 */
4027                if ((_phdr == NULL) && (phdr->p_type == PT_LOAD)) {
4028                        _phdr = phdr;
4029
4030                        if ((ofl->ofl_dtflags_1 & DF_1_NOHDR) == 0) {
4031                                if (ofl->ofl_osinterp)
4032                                        ofl->ofl_phdr[0].p_vaddr =
4033                                            vaddr + ehdrsz;
4034
4035                                /*
4036                                 * Finally, if we're creating a dynamic object
4037                                 * (or a static object in which an interpreter
4038                                 * is specified) update the vaddr to reflect
4039                                 * the address of the first section within this
4040                                 * segment.
4041                                 */
4042                                if ((ofl->ofl_osinterp) ||
4043                                    (flags & FLG_OF_DYNAMIC))
4044                                        vaddr += size;
4045                        } else {
4046                                /*
4047                                 * If the DF_1_NOHDR flag was set, and an
4048                                 * interpreter is being generated, the PT_PHDR
4049                                 * will not be part of any loadable segment.
4050                                 */
4051                                if (ofl->ofl_osinterp) {
4052                                        ofl->ofl_phdr[0].p_vaddr = 0;
4053                                        ofl->ofl_phdr[0].p_memsz = 0;
4054                                        ofl->ofl_phdr[0].p_flags = 0;
4055                                }
4056                        }
4057                }
4058
4059                /*
4060                 * Ensure the ELF entry point defaults to zero.  Typically, this
4061                 * value is overridden in update_oehdr() to one of the standard
4062                 * entry points.  Historically, this default was set to the
4063                 * address of first executable section, but this has since been
4064                 * found to be more confusing than it is helpful.
4065                 */
4066                ehdr->e_entry = 0;
4067
4068                DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
4069
4070                /*
4071                 * Traverse the output section descriptors for this segment so
4072                 * that we can update the section headers addresses.  We've
4073                 * calculated the virtual address of the initial section within
4074                 * this segment, so each successive section can be calculated
4075                 * based on their offsets from each other.
4076                 */
4077                secndx = 0;
4078                hshdr = 0;
4079                for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
4080                        Shdr    *shdr = osp->os_shdr;
4081
4082                        if (shdr->sh_link)
```

```
4083                                shdr->sh_link = translate_link(ofl, osp,
4084                                    shdr->sh_link, MSG_INTL(MSG_FIL_INVSHLINK));
4085
4086                        if (shdr->sh_info && (shdr->sh_flags & SHF_INFO_LINK))
4087                                shdr->sh_info = translate_link(ofl, osp,
4088                                    shdr->sh_info, MSG_INTL(MSG_FIL_INVSHINFO));
4089
4090                        if (!(flags & FLG_OF_RELOBJ) &&
4091                            (phdr->p_type == PT_LOAD)) {
4092                                if (hshdr)
4093                                        vaddr += (shdr->sh_offset -
4094                                            hshdr->sh_offset);
4095
4096                                shdr->sh_addr = vaddr;
4097                                hshdr = shdr;
4098                        }
4099
4100                        DBG_CALL(Dbg_seg_os(ofl, osp, secndx));
4101                        secndx++;
4102                }
4103
4104                /*
4105                 * Establish the virtual address of the end of the last section
4106                 * in this segment so that the next segments offset can be
4107                 * calculated from this.
4108                 */
4109                if (hshdr)
4110                        vaddr += hshdr->sh_size;
4111
4112                /*
4113                 * Output sections for this segment complete.  Adjust the
4114                 * virtual offset for the last sections size, and make sure we
4115                 * haven't exceeded any maximum segment length specification.
4116                 */
4117                if ((sgp->sg_length != 0) && (sgp->sg_length < phdr->p_memsz)) {
4118                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_UPD_LARGSIZE),
4119                            ofl->ofl_name, sgp->sg_name,
4120                            EC_XWORD(phdr->p_memsz), EC_XWORD(sgp->sg_length));
4121                        return (S_ERROR);
4122                }
4123
4124                if (phdr->p_type == PT_NOTE) {
4125                        phdr->p_vaddr = 0;
4126                        phdr->p_paddr = 0;
4127                        phdr->p_align = 0;
4128                        phdr->p_memsz = 0;
4129                }
4130
4131                if ((phdr->p_type != PT_NULL) && !(flags & FLG_OF_RELOBJ))
4132                        ofl->ofl_phdr[phdrndx++] = *phdr;
4133        }
4134
4135        /*
4136         * Update any new output sections.  When building the initial output
4137         * image, a number of sections were created but left uninitialized (eg.
4138         * .dynsym, .dynstr, .symtab, .symtab, etc.).  Here we update these
4139         * sections with the appropriate data.  Other sections may still be
4140         * modified via reloc_process().
4141         *
4142         * Copy the interpreter name into the .interp section.
4143         */
4144        if (ofl->ofl_interp)
4145                (void) strcpy((char *)ofl->ofl_osinterp->os_outdata->d_buf,
4146                    ofl->ofl_interp);
4147
4148        /*
```



```
4149         * Update the .shstrtab, .strtab and .dynstr sections.
4150         */
4151        update_ostrtab(ofl->ofl_osshstrtab, ofl->ofl_shdrsttab, 0);
4152        update_ostrtab(ofl->ofl_osstrtab, ofl->ofl_strtab, 0);
4153        update_ostrtab(ofl->ofl_osdynstr, ofl->ofl_dynstrtab, DYNSTR_EXTRA_PAD);
4154
4155        /*
4156         * Build any output symbol tables, the symbols information is copied
4157         * and updated into the new output image.
4158         */
4159        if ((etext = update_osym(ofl)) == (Addr)S_ERROR)
4160                return (S_ERROR);
4161
4162        /*
4163         * If we have an PT_INTERP phdr, update it now from the associated
4164         * section information.
4165         */
4166        if (intpsgp) {
4167                Phdr    *phdr = &(intpsgp->sg_phdr);
4168                Shdr    *shdr = ofl->ofl_osinterp->os_shdr;
4169
4170                phdr->p_vaddr = shdr->sh_addr;
4171                phdr->p_offset = shdr->sh_offset;
4172                phdr->p_memsz = phdr->p_filesz = shdr->sh_size;
4173                phdr->p_flags = PF_R;
4174
4175                DBG_CALL(Dbg_seg_entry(ofl, intpsndx, intpsgp));
4176                ofl->ofl_phdr[intppndx] = *phdr;
4177        }
4178
4179        /*
4180         * If we have a PT_SUNWDTRACE phdr, update it now with the address of
4181         * the symbol.  It's only now been updated via update_sym().
4182         */
4183        if (dtracesgp) {
4184                Phdr            *aphdr, *phdr = &(dtracesgp->sg_phdr);
4185                Sym_desc        *sdp = ofl->ofl_dtracesym;
4186
4187                phdr->p_vaddr = sdp->sd_sym->st_value;
4188                phdr->p_memsz = sdp->sd_sym->st_size;
4189
4190                /*
4191                 * Take permissions from the segment that the symbol is
4192                 * associated with.
4193                 */
4194                aphdr = &sdp->sd_isc->is_osdesc->os_sgdesc->sg_phdr;
4195                assert(aphdr);
4196                phdr->p_flags = aphdr->p_flags;
4197
4198                DBG_CALL(Dbg_seg_entry(ofl, dtracesndx, dtracesgp));
4199                ofl->ofl_phdr[dtracepndx] = *phdr;
4200        }
4201
4202        /*
4203         * If we have a PT_SUNWCAP phdr, update it now from the associated
4204         * section information.
4205         */
4206        if (capsgp) {
4207                Phdr    *phdr = &(capsgp->sg_phdr);
4208                Shdr    *shdr = ofl->ofl_oscap->os_shdr;
4209
4210                phdr->p_vaddr = shdr->sh_addr;
4211                phdr->p_offset = shdr->sh_offset;
4212                phdr->p_memsz = phdr->p_filesz = shdr->sh_size;
4213                phdr->p_flags = PF_R;
```

```
4215                DBG_CALL(Dbg_seg_entry(ofl, capsndx, capsgp));
4216                ofl->ofl_phdr[cappndx] = *phdr;
4217        }

4219        /*
4220         * Update the GROUP sections.
4221         */
4222        if (update_ogroup(ofl) == S_ERROR)
4223                return (S_ERROR);

4225        /*
4226         * Update Move Table.
4227         */
4228        if (ofl->ofl_osmove || ofl->ofl_isparexpn)
4229                update_move(ofl);

4231        /*
4232         * Build any output headers, version information, dynamic structure and
4233         * syminfo structure.
4234         */
4235        if (update_oehdr(ofl) == S_ERROR)
4236                return (S_ERROR);
4237        if (!(flags & FLG_OF_NOVERSEC)) {
4238                if ((flags & FLG_OF_VERDEF) &&
4239                    (update_overdef(ofl) == S_ERROR))
4240                        return (S_ERROR);
4241                if ((flags & FLG_OF_VERNEED) &&
4242                    (update_overneed(ofl) == S_ERROR))
4243                        return (S_ERROR);
4244                if (flags & (FLG_OF_VERNEED | FLG_OF_VERDEF))
4245                        update_oversym(ofl);
4246        }
4247        if (flags & FLG_OF_DYNAMIC) {
4248                if (update_odynamic(ofl) == S_ERROR)
4249                        return (S_ERROR);
4250        }
4251        if (ofl->ofl_ossyminfo) {
4252                if (update_osyminfo(ofl) == S_ERROR)
4253                        return (S_ERROR);
4254        }

4256        /*
4257         * Update capabilities information if required.
4258         */
4259        if (ofl->ofl_oscap)
4260                update_oscap(ofl);
4261        if (ofl->ofl_oscapinfo)
4262                update_oscapinfo(ofl);

4264        /*
4265         * Sanity test: the first and last data byte of a string table
4266         * must be NULL.
4267         */
4268        assert((ofl->ofl_osshstrtab == NULL) ||
4269            (*((char *)ofl->ofl_osshstrtab->os_outdata->d_buf) == '\0'));
4270        assert((ofl->ofl_osshstrtab == NULL) ||
4271            (*(((char *)ofl->ofl_osshstrtab->os_outdata->d_buf) +
4272            ofl->ofl_osshstrtab->os_outdata->d_size - 1) == '\0'));

4274        assert((ofl->ofl_osstrtab == NULL) ||
4275            (*((char *)ofl->ofl_osstrtab->os_outdata->d_buf) == '\0'));
4276        assert((ofl->ofl_osstrtab == NULL) ||
4277            (*(((char *)ofl->ofl_osstrtab->os_outdata->d_buf) +
4278            ofl->ofl_osstrtab->os_outdata->d_size - 1) == '\0'));

4280        assert((ofl->ofl_osdynstr == NULL) ||
```



```
4281            (*((char *)ofl->ofl_osdynstr->os_outdata->d_buf) == '\0'));
4282        assert((ofl->ofl_osdynstr == NULL) ||
4283            (*(((char *)ofl->ofl_osdynstr->os_outdata->d_buf) +
4284            ofl->ofl_osdynstr->os_outdata->d_size - DYNSTR_EXTRA_PAD - 1) ==
4285            '\0'));

4287        /*
4288         * Emit Strtab diagnostics.
4289         */
4290        DBG_CALL(Dbg_sec_strtab(ofl->ofl_lml, ofl->ofl_osshstrtab,
4291            ofl->ofl_shdrsttab));
4292        DBG_CALL(Dbg_sec_strtab(ofl->ofl_lml, ofl->ofl_osstrtab,
4293            ofl->ofl_strtab));
4294        DBG_CALL(Dbg_sec_strtab(ofl->ofl_lml, ofl->ofl_osdynstr,
4295            ofl->ofl_dynstrtab));

4297        /*
4298         * Initialize the section headers string table index within the elf
4299         * header.
4300         */
4301        /* LINTED */
4302        if ((shscnndx = elf_ndxscn(ofl->ofl_osshstrtab->os_scn)) <
4303            SHN_LORESERVE) {
4304                ofl->ofl_nehdr->e_shstrndx =
4305                    /* LINTED */
4306                    (Half)shscnndx;
4307        } else {
4308                /*
4309                 * If the STRTAB section index doesn't fit into
4310                 * e_shstrndx, then we store it in 'shdr[0].st_link'.
4311                 */
4312                Elf_Scn *scn;
4313                Shdr    *shdr0;

4315                if ((scn = elf_getscn(ofl->ofl_elf, 0)) == NULL) {
4316                        ld_eprintf(ofl, ERR_ELF, MSG_INTL(MSG_ELF_GETSCN),
4317                            ofl->ofl_name);
4318                        return (S_ERROR);
4319                }
4320                if ((shdr0 = elf_getshdr(scn)) == NULL) {
4321                        ld_eprintf(ofl, ERR_ELF, MSG_INTL(MSG_ELF_GETSHDR),
4322                            ofl->ofl_name);
4323                        return (S_ERROR);
4324                }
4325                ofl->ofl_nehdr->e_shstrndx = SHN_XINDEX;
4326                shdr0->sh_link = shscnndx;
4327        }

4329        return ((uintptr_t)etext);
4330 }
```

```
**********************************************************
   88729 Mon Feb 11 00:23:20 2019
new/usr/src/cmd/sgs/packages/common/SUNWonld-README
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

```
#
# Copyright (c) 1996, 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
#
# Note: The contents of this file are used to determine the versioning
#       information for the SGS toolset. The number of CRs listed in
#       this file must grow monotonically, or the SGS version will
#       move backwards, causing a great deal of confusion. As such,
#       CRs must never be removed from this file. See
#       libconv/common/bld_vernote.ksh, and bug#4519569 for more
#       details on SGS versioning.
#
-------------------------------------------------------------------------------
SUNWonld - link-editors development package.
-------------------------------------------------------------------------------

  The SUNWonld package is an internal development package containing the
  link-editors and some related tools.  All components live in the OSNET
  source base, but not all components are delivered as part of the normal
  OSNET consolidation.  The intent of this package is to provide access
  to new features/bugfixes before they become generally available.

  General link-editor information can be found:

    http://linkers.central/
    http://linkers.sfbay/       (also known as linkers.eng)

  Comments and Questions:

    Contact Rod Evans, Ali Bahrami, and/or Seizo Sakurai.

  Warnings:

    The postremove script for this package employs /usr/sbin/static/mv,
    and thus, besides the common core dependencies, this package also
    has a dependency on the SUNWsutl package.

  Patches:

    If the patch has been made official, you'll find it in:
```



```
        http://sunsolve.east/cgi/show.pl?target=patches/os-patches

    If it hasn't been released, the patch will be in:

        /net/sunsoftpatch/patches/temporary

    Note, any patches logged here refer to the temporary ("T") name, as we
    never know when they're made official, and although we try to keep all
    patch information up-to-date the real status of any patch can be
    determined from:

        http://sunsoftpatch.eng

    If it has been obsoleted, the patch will be in:

        /net/on${RELEASE}-patch/on${RELEASE}/patches/${MACH}/obsolete


  History:

    Note, starting after Solaris 10, letter codes in parenthesis may
    be found following the bug synopsis. Their meanings are as follows:

        (D) A documentation change accompanies the implementation change.
        (P) A packaging change accompanies the implementation change.

    In all cases, see the implementation bug report for details.

    The following bug fixes exist in the OSNET consolidation workspace
    from which this package is created:

---------
Solaris 8
---------
Bugid   Risk Synopsis
================================================================================
4225937 i386 linker emits sparc specific warning messages
4215164 shf_order flag handling broken by fix for 4194028.
4215587 using ld and the -r option on solaris 7 with compiler option -xarch=v9
        causes link errors.
4234657 103627-08 breaks purify 4.2 (plt padding should not be enabled for
        32-bit)
4235241 dbx no longer gets dlclose notification.
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-05  (never released)
        Solaris/SunOS 5.7_x86           patch 106951-05  (never released)
        Solaris/SunOS 5.6_sparc         patch 107733-02  (never released)
        Solaris/SunOS 5.6_x86           patch 107734-02
--------------------------------------------------------------------------------
4248290 inetd dumps core upon bootup - failure in dlclose() logic.
4238071 dlopen() leaks while descriptors under low memory conditions
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-06
        Solaris/SunOS 5.7_x86           patch 106951-06
        Solaris/SunOS 5.6_sparc         patch 107733-03  (never released)
        Solaris/SunOS 5.6_x86           patch 107734-03
--------------------------------------------------------------------------------
4267980 INITFIRST flag of the shard object could be ignored.
--------------------------------------------------------------------------------
All the above changes plus:
    4238973 fix for 4121152 affects linking of Ada objects
    4158744 patch 103627-02 causes core when RPATH has blank entry and
            dlopen/dlclose is used
are incorporated in the following patches:
```

```
        Solaris/SunOS 5.5.1_sparc       patch 103627-12  (never released)
        Solaris/SunOS 5.5.1_x86         patch 103628-11
-------------------------------------------------------------------------------
4256518 miscalculated calloc() during dlclose/tsorting can result in segv
4254171 DT_SPARC_REGISTER has invalid value associated with it.
-------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-07
        Solaris/SunOS 5.7_x86           patch 106951-07
        Solaris/SunOS 5.6_sparc         patch 107733-04  (never released)
        Solaris/SunOS 5.6_x86           patch 107734-04
-------------------------------------------------------------------------------
4293159 ld needs to combine sections with and without SHF_ORDERED flag(comdat)
4292238 linking a library which has a static char ptr invokes mprotect() call
-------------------------------------------------------------------------------
All the above changes except for:
    4256518 miscalculated calloc() during dlclose/tsorting can result in segv
    4254171 DT_SPARC_REGISTER has invalid value associated with it.
plus:
    4238973 fix for 4121152 affects linking of Ada objects
    4158744 patch 103627-02 causes core when RPATH has blank entry and
            dlopen/dlclose is used
are incorporated in the following patches:
        Solaris/SunOS 5.5.1_sparc       patch 103627-13
        Solaris/SunOS 5.5.1_x86         patch 103628-12
-------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-08
        Solaris/SunOS 5.7_x86           patch 106951-08
        Solaris/SunOS 5.6_sparc         patch 107733-05
        Solaris/SunOS 5.6_x86           patch 107734-05
-------------------------------------------------------------------------------
4295613 COMMON symbol resolution can be incorrect
-------------------------------------------------------------------------------
All the above changes plus:
    4238973 fix for 4121152 affects linking of Ada objects
    4158744 patch 103627-02 causes core when RPATH has blank entry and
            dlopen/dlclose is used
are incorporated in the following patches:
        Solaris/SunOS 5.5.1_sparc       patch 103627-14
        Solaris/SunOS 5.5.1_x86         patch 103628-13
-------------------------------------------------------------------------------
All the above changes plus:
    4351197 nfs performance problem by 103627-13
are incorporated in the following patches:
        Solaris/SunOS 5.5.1_sparc       patch 103627-15
        Solaris/SunOS 5.5.1_x86         patch 103628-14
-------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-09
        Solaris/SunOS 5.7_x86           patch 106951-09
        Solaris/SunOS 5.6_sparc         patch 107733-06
        Solaris/SunOS 5.6_x86           patch 107734-06
-------------------------------------------------------------------------------
4158971 increase the default segment alignment for i386 to 64k
4064994 Add an $ISALIST token to those understood by the dynamic linker
xxxxxxx ia64 common code putback
4239308 LD_DEBUG busted for sparc machines
4239008 Support MAP_ANON
4238494 link-auditing extensions required
4232239 R_SPARC_LOX10 truncates field
4231722 R_SPARC_UA* relocations are busted
4235514 R_SPARC_OLO10 relocation fails
4244025 sgsmsg update
4239281 need to support SECREL relocations for ia64
4253751 ia64 linker must support PT_IA_64_UNWIND tables
```



```
4259254 dlmopen mistakenly closes fd 0 (stdin) under certain error conditions
4260872 libelf hangs when libthread present
4224569 linker core dumping when profiling specified
4270937 need mechanism to suppress ld.so.1's use of a default search path.
1050476 ld.so to permit configuration of search path
4273654 filtee processing using $ISALIST could be optimized
4271860 get MERCED cruft out of elf.h
4248991 Dynamic loader (via PLT) corrupts register G4
4275754 cannot mmap file: Resource temporarily unavailable
4277689 The linker can not handle relocation against MOVE tabl
4270766 atexit processing required on dlclose().
4279229 Add a "release" token to those understood by the dynamic linker
4215433 ld can bus error when insufficient disc space exists for output file
4285571 Pssst, want some free disk space?  ld's miscalculating.
4286236 ar gives confusing "bad format" error with a null .stab section
4286838 ld.so.1 can't handle a no-bits segment
4287364 ld.so.1 runtime configuration cleanup
4289573 disable linking of ia64 binaries for Solaris8
4293966 crle(1)'s default directories should be supplied
-------------------------------------------------------------------------------

-----------------------------------
Solaris 8 600 (1st Q-update - s28u1)
-----------------------------------
Bugid   Risk Synopsis
===============================================================================
4309212 dlsym can't find symbol
4311226 rejection of preloading in secure apps is inconsistent
4312449 dlclose: invalid deletion of dependency can occur using RTLD_GLOBAL
-------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-01
        Solaris/SunOS 5.8_x86           patch 109148-01
        Solaris/SunOS 5.7_sparc         patch 106950-10
        Solaris/SunOS 5.7_x86           patch 106951-10
        Solaris/SunOS 5.6_sparc         patch 107733-07
        Solaris/SunOS 5.6_x86           patch 107734-07
-------------------------------------------------------------------------------

-----------------------------------
Solaris 8 900 (2nd Q-update - s28u2)
-----------------------------------
Bugid   Risk Synopsis
===============================================================================
4324775 non-PIC code & -zcombreloc don't mix very well...
4327653 run-time linker should preload tables it will process (madvise)
4324324 shared object code can be referenced before .init has fired
4321634 .init firing of multiple INITFIRST objects can fail
-------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-03
        Solaris/SunOS 5.8_x86           patch 109148-03
        Solaris/SunOS 5.7_sparc         patch 106950-11
        Solaris/SunOS 5.7_x86           patch 106951-11
        Solaris/SunOS 5.6_sparc         patch 107733-08
        Solaris/SunOS 5.6_x86           patch 107734-08
-------------------------------------------------------------------------------
4338812 crle(1) omits entries in the directory cache
4341496 RFE: provide a static version of /usr/bin/crle
4340878 rtld should treat $ORIGIN like LD_LIBRARY_PATH in security issues
-------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-04
        Solaris/SunOS 5.8_x86           patch 109148-04
        Solaris/SunOS 5.7_sparc         patch 106950-12
        Solaris/SunOS 5.7_x86           patch 106951-12
```

```
258 ------------------------------------------------------------------------------
259 4349563 auxiliary filter error handling regression introduced in 4165487
260 4355795 ldd -r now gives "displacement relocated" warnings
261 ------------------------------------------------------------------------------
262 All the above changes are incorporated in the following patches:
263         Solaris/SunOS 5.7_sparc         patch 106950-13
264         Solaris/SunOS 5.7_x86           patch 106951-13
265         Solaris/SunOS 5.6_sparc         patch 107733-09
266         Solaris/SunOS 5.6_x86           patch 107734-09
267 ------------------------------------------------------------------------------
268 4210412 versioning a static executable causes ld to core dump
269 4219652 Linker gives misleading error about not finding main (xarch=v9)
270 4103449 ld command needs a command line flag to force 64-bits
271 4187211 problem with RDISP32 linking in copy-relocated objects
272 4287274 dladdr, dlinfo do not provide the full path name of a shared object
273 4297563 dlclose still does not remove all objects.
274 4250694 rtld_db needs a new auxvec entry
275 4235315 new features for rtld_db (DT_CHECKSUM, dynamic linked .o files
276 4303609 64bit libelf.so.1 does not properly implement elf_hash()
277 4310901 su.static fails when OSNet build with lazy-loading
278 4310324 elf_errno() causes Bus Error(coredump) in 64-bit multithreaded programs
279 4306415 ld core dump
280 4316531 BCP: possible failure with dlclose/_preexec_exit_handlers
281 4313765 LD_BREADTH should be shot
282 4318162 crle uses automatic strings in putenv.
283 4255943 Description of -t option incomplete.
284 4322528 sgs message test infrastucture needs improvement
285 4239213 Want an API to obtain linker's search path
286 4324134 use of extern mapfile directives can contribute unused symbols
287 4322581 ELF data structures could be layed out more efficiently...
288 4040628 Unnecessary section header symbols should be removed from .dynsym
289 4300018 rtld: bindlock should be freed before calling call_fini()
290 4336102 dlclose with non-deletable objects can mishandle dependencies
291 4329785 mixing of SHT_SUNW_COMDAT & SHF_ORDERED causes ld to seg fault
292 4334617 COPY relocations should be produces for references to .bss symbols
293 4248250 relcoation of local ABS symbols incorrect
294 4335801 For complimentary alignments eliminate ld: warning: symbol `ll'
295         has differing a
296 4336980 ld.so.1 relative path processing revisited
297 4243097 dlerror(3DL) is not affected by setlocale(3C).
298 4344528 dump should remove -D and -l usage message
299 xxxxxxx enable LD_ALTEXEC to access alternate link-editor
300 ------------------------------------------------------------------------------
301 All the above changes are incorporated in the following patches:
302         Solaris/SunOS 5.8_sparc         patch 109147-06
303         Solaris/SunOS 5.8_x86           patch 109148-06
304 ------------------------------------------------------------------------------

306 ------------------------------------
307 Solaris 8 101 (3rd Q-update - s28u3)
308 ------------------------------------
309 Bugid   Risk Synopsis
310 ==============================================================================
311 4346144 link-auditing: plt_tracing fails if LA_SYMB_NOPLTENTER given after
312         being bound
313 4346001 The ld should support mapfile syntax to generate PT_SUNWSTACK segment
314 4349137 rtld_db: A third fallback method for locating the linkmap
315 4343417 dladdr interface information inadequate
316 4343801 RFE: crle(1): provide option for updating configuration files
317 4346615 ld.so.1 attempting to open a directory gives: No such device
318 4352233 crle should not honor umask
319 4352330 LD_PRELOAD cannot use absolute path for privileged program
320 4357805 RFE: man page for ld(1) does not document all -z or -B options in
321         Solaris 8 9/00
322 4358751 ld.so.1: LD_XXX environ variables and LD_FLAGS should be synchronized.
323 4358862 link editors should reference "64" symlinks instead of sparcv9 (ia64).
```



```
324 4356879 PLTs could use faster code sequences in some cases
325 4367118 new fast baplt's fail when traversed twice in threaded application
326 4366905 Need a way to determine path to a shared library
327 4351197 nfs performance problem by 103627-13
328 4367405 LD_LIBRARY_PATH_64 not being used
329 4354500 SHF_ORDERED ordered scections does not properly sort sections
330 4369068 ld(1)'s weak symbol processing is inefficient (slow and doesn't scale).
331 ------------------------------------------------------------------------------
332 All the above changes are incorporated in the following patches:
333         Solaris/SunOS 5.8_sparc         patch 109147-07
334         Solaris/SunOS 5.8_x86           patch 109148-07
335         Solaris/SunOS 5.7_sparc         patch 106950-14
336         Solaris/SunOS 5.7_x86           patch 106951-14
337 ------------------------------------------------------------------------------

339 ------------------------------------
340 Solaris 8 701 (5th Q-update - s28u5)
341 ------------------------------------
342 Bugid   Risk Synopsis
343 ==============================================================================
344 4368846 ld(1) fails to version some interfaces given in a mapfile
345 4077245 dump core dump on null pointer.
346 4372554 elfdump should demangle symbols (like nm, dump)
347 4371114 dlclose may unmap a promiscuous object while it's still in use.
348 4204447 elfdump should understand SHN_AFTER/SHN_BEGIN macro
349 4377941 initialization of interposers may not occur
350 4381116 ldd/ld.so.1 could aid in detecting unused dependencies
351 4381783 dlopen/dlclose of a libCrun+libthread can dump core
352 4385402 linker & run-time linker must support gABI ELF updates
353 4394698 ld.so.1 does not process DF_SYMBOLIC - not gABI conforming
354 4394212 the link editor quietly ignores missing support libraries
355 4390308 ld.so.1 should provide more flexibility LD_PRELOAD'ing 32-bit/64-bit
356         objects
357 4401232 crle(1) could provide better flexibility for alternatives
358 4401815 fix misc nits in debugging output...
359 4402861 cleanup /usr/demo/link_audit & /usr/tmp/librtld_db demo source code...
360 4393044 elfdump should allow raw dumping of sections
361 4413168 SHF_ORDERED bit causes linker to generate a separate section
362 ------------------------------------------------------------------------------
363 All the above changes are incorporated in the following patches:
364         Solaris/SunOS 5.8_sparc         patch 109147-08
365         Solaris/SunOS 5.8_x86           patch 109148-08
366 ------------------------------------------------------------------------------
367 4452202 Typos in <sys/link.h>
368 4452220 dump doesn't support RUNPATH
369 ------------------------------------------------------------------------------
370 All the above changes are incorporated in the following patches:
371         Solaris/SunOS 5.8_sparc         patch 109147-09
372         Solaris/SunOS 5.8_x86           patch 109148-09
373 ------------------------------------------------------------------------------

375 ------------------------------------
376 Solaris 8 1001 (6th Q-update - s28u6)
377 ------------------------------------
378 Bugid   Risk Synopsis
379 ==============================================================================
380 4421842 fixups in SHT_GROUP processing required...
381 4450433 problem with liblddbg output on -Dsection,detail when
382         processing SHF_LINK_ORDER
383 ------------------------------------------------------------------------------
384 All the above changes are incorporated in the following patches:
385         Solaris/SunOS 5.8_sparc         patch 109147-10
386         Solaris/SunOS 5.8_x86           patch 109148-10
387         Solaris/SunOS 5.7_sparc         patch 106950-15
388         Solaris/SunOS 5.7_x86           patch 106951-15
389 ------------------------------------------------------------------------------
```

```
4463473 pldd showing wrong output
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-11
        Solaris/SunOS 5.8_x86           patch 109148-11
--------------------------------------------------------------------------------

-----------------------------------
Solaris 8 202 (7th Q-update - s28u7)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4488954 ld.so.1 reuses same buffer to send ummapping range to
        _preexec_exit_handlers()
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-12
        Solaris/SunOS 5.8_x86           patch 109148-12
--------------------------------------------------------------------------------

---------
Solaris 9
---------
Bugid   Risk Synopsis
================================================================================
4505289 incorrect handling of _START_ and _END_
4506164 mcs does not recognize #linkbefore or #linkafter qualifiers
4447560 strip is creating unexecutable files...
4513842 library names not in ld.so string pool cause corefile bugs
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-13
        Solaris/SunOS 5.8_x86           patch 109148-13
        Solaris/SunOS 5.7_sparc         patch 106950-16
        Solaris/SunOS 5.7_x86           patch 106951-16
--------------------------------------------------------------------------------
4291384 ld -M with a mapfile does not properly align Fortran REAL*8 data
4413322 SunOS 5.9 librtld_db doesn't show dlopened ".o" files anymore?
4429371 librtld_db busted on ia32 with SC6.x compilers...
4418274 elfdump dumps core on invalid input
4432224 libelf xlate routines are out of date
4433643 Memory leak using dlopen()/dlclose() in Solaris 8
4446564 ldd/lddstub - core dump conditions
4446115 translating SUNW_move sections is broken
4450225 The rdb command can fall into an infinite loop
4448531 Linker Causes Segmentation Fault
4453241 Regression in 4291384 can result in empty symbol table.
4453398 invalid runpath token can cause ld to spin.
4460230 ld (for OS 5.8 and 5.9) loses error message
4462245 ld.so.1 core dumps when executed directly...
4455802 need more flexibility in establishing a support library for ld
4467068 dyn_plt_entsize not properly initialized in ld.so.1
4468779 elf_plt_trace_write() broken on i386 (link-auditing)
4465871 -zld32 and -zld64 does not work the way it should
4461890 bad shared object created with -zredlocsym
4469400 ld.so.1: is_so_loaded isn't as efficient as we thought...
4469566 lazy loading fallback can reference un-relocated objects
4470493 libelf incorectly translates NOTE sections accross architectures...
4469684 rtld leaks dl_handles and permits on dlopen/dlclose
4475174 ld.so.1 prematurly reports the failure to load a object...
4475514 ld.so.1 can core dump in memory allocation fails (no swap)
4481851 Setting ld.so.1 environment variables globally would be useful
4482035 setting LD_PROFILE & LD_AUDIT causes ping command to issue warnings
        on 5.8
4377735 segment reservations cause sbrk() to fail
4491434 ld.so.1 can leak file-descriptors when loading same named objects
```



```
4289232 some of warning/error/debugging messages from libld.so can be revised
4462748 Linker Portion of TLS Support
4496718 run-time linkers mutex_locks not working with ld_libc interface
4497270 The -zredlocsym option should not eliminate partially initialized local
        symbols
4496963 dumping an object with crle(1) that uses $ORIGIN can loose its
        dependencies
4499413 Sun linker orders of magnitude slower than gnu linker
4461760 lazy loading libXm and libXt can fail.
4469031 The partial initialized (local) symbols for intel platform is not
        working.
4492883 Add link-editor option to multi-pass archives to resolve unsatisfied
        symbols
4503731 linker-related commands misspell "argument"
4503768 whocalls(1) should output messages to stderr, not stdout
4503748 whocalls(1) usage message and manpage could be improved
4503625 nm should be taught about TLS symbols - that they aren't allowed that is
4300120 segment address validation is too simplistic to handle segment
        reservations
4404547 krtld/reloc.h could have better error message, has typos
4270931 R_SPARC_HIX22 relocation is not handled properly
4485320 ld needs to support more the 32768 PLTs
4516434 sotruss can not watch libc_psr.so.1
4213100 sotruss could use more flexible pattern matching
4503457 ld seg fault with comdat
4510264 sections with SHF_TLS can come in different orders...
4518079 link-editor support library unable to modify section header flags
4515913 ld.so.1 can incorrectly decrement external reference counts on dlclose()
4519569 ld -V does not return a interesting value...
4524512 ld.so.1 should allow alternate termination signals
4524767 elfdump dies on bogus sh_name fields...
4524735 ld getopt processing of '-' changed
4521931 subroutine in a shared object as LOCL instead of GLOB
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-14
        Solaris/SunOS 5.8_x86           patch 109148-14
        Solaris/SunOS 5.7_sparc         patch 106950-17
        Solaris/SunOS 5.7_x86           patch 106951-17
--------------------------------------------------------------------------------
4532729 tentative definition of TLS variable causes linker to dump core
4526745 fixup ld error message about duplicate dependencies/needed names
4522999 Solaris linker one order of magnitude slower than GNU linker
4518966 dldump undoes existing relocations with no thought of alignment or size.
4587441 Certain libraries have race conditions when setting error codes
4523798 linker option to align bss to large pagesize alignments.
4524008 ld can improperly set st_size of symbols named "_init" or "_fini"
4619282 ld cannot link a program with the option -sb
4620846 Perl Configure probing broken by ld changes
4621122 multiple ld '-zinitarray=' on a commandline fails
--------------------------------------------------------------------------------
        Solaris/SunOS 5.8_sparc         patch 109147-15
        Solaris/SunOS 5.8_x86           patch 109148-15
        Solaris/SunOS 5.7_sparc         patch 106950-18
        Solaris/SunOS 5.7_x86           patch 106951-18
        Solaris/SunOS 5.6_sparc         patch 107733-10
        Solaris/SunOS 5.6_x86           patch 107734-10
--------------------------------------------------------------------------------
All the above changes plus:
        4616944 ar seg faults when order of object file is reversed.
are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-16
        Solaris/SunOS 5.8_x86           patch 109148-16
--------------------------------------------------------------------------------
All the above changes plus:
        4872634 Large LD_PRELOAD values can cause SEGV of process
```

```
are incorporated in the following patches:
        Solaris/SunOS 5.6_sparc         patch T107733-11
        Solaris/SunOS 5.6_x86           patch T107734-11
------------------------------------------------------------------------------

----------------------------------
Solaris 9 1202 (2nd Q-update - s9u2)
----------------------------------
Bugid   Risk Synopsis
==============================================================================
4546416 add help messages to ld.so mdbmodule
4526752 we should build and ship ld.so's mdb module
4624658 update 386 TLS relocation values
4622472 LA_SYMB_DLSYM not set for la_symbind() invocations
4638070 ldd/ld.so.1 could aid in detecting unreferenced dependencies
        PSARC/2002/096 Detecting unreferenced dependencies with ldd(1)
4633860 Optimization for unused static global variables
        PSARC/2002/113 ld -zignore - section elimination
4642829 ld.so.1 mprotect()'s text segment for weak relocations (it shouldn't)
4621479 'make' in $SRC/cmd/sgs/tools tries to install things in the proto area
4529912 purge ia64 source from sgs
4651709 dlopen(RTLD_NOLOAD) can disable lazy loading
4655066 crle: -u with nonexistent config file doesn't work
4654406 string tables created by the link-editor could be smaller...
        PSARC/2002/160 ld -znocompstrtab - disable string-table compression
4651493 RTLD_NOW can result in binding to an object prior to its init being run.
4662575 linker displacement relocation checking introduces significant
        linker overhead
4533195 ld interposes on malloc()/free() preventing support library from freeing
        memory
4630224 crle get's confused about memory layout of objects...
4664855 crle on application failed with ld.so.1 encountering mmap() returning
        ENOMEM err
4669582 latest dynamic linker causes libthread _init to get skipped
4671493 ld.so.1 inconsistantly assigns PATHNAME() on primary objects
4668517 compile with map.bssalign doesn't copy _iob to bss
------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-01
        Solaris/SunOS 5.8_sparc         patch T109147-17
        Solaris/SunOS 5.8_x86           patch T109148-17
------------------------------------------------------------------------------
4701749 On Solaris 8 + 109147-16 ld crashes when building a dynamic library.
4707808 The ldd command is broken in the latest 2.8 linker patch.
------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-02
        Solaris/SunOS 5.8_sparc         patch T109147-18
        Solaris/SunOS 5.8_x86           patch T109148-18
------------------------------------------------------------------------------
4696204 enable extended section indexes in relocatable objects
        PSARC/2001/332 ELF gABI updates - round II
        PSARC/2002/369 libelf interfaces to support ELF Extended Sections
4706503 linkers need to cope with EF_SPARCV9_PSO/EF_SPARCV9_RMO
4716929 updating of local register symbols in dynamic symtab busted...
4710814 add "official" support for the "symbolic" keyword in linker map-file
        PSARC/2002/439 linker mapfile visibility declarations
------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-03
        Solaris/SunOS 5.8_sparc         patch T109147-19
        Solaris/SunOS 5.8_x86           patch T109148-19
        Solaris/SunOS 5.7_sparc         patch T106950-19
        Solaris/SunOS 5.7_x86           patch T106951-19
------------------------------------------------------------------------------
```



```
----------------------------------
Solaris 9 403 (3nd Q-update - s9u3)
----------------------------------
Bugid   Risk Synopsis
==============================================================================
4731174 strip(1) does not fixup SHT_GROUP data
4733697 -zignore with gcc may exclude C++ exception sections
4733317 R_SPARC_*_HIX22 calculations are wrong with 32bit LD building
        ELF64 binaries
4735165 fatal linker error when compiling C++ programs with -xlinkopt
4736951 The mcs broken when the target file is an archive file
------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch T109147-20
        Solaris/SunOS 5.8_x86           patch T109148-20
        Solaris/SunOS 5.7_sparc         patch T106950-20
        Solaris/SunOS 5.7_x86           patch T106951-20
------------------------------------------------------------------------------
4739660 Threads deadlock in schedlock and dynamic linker lock.
4653148 ld.so.1/libc should unregister its dlclose() exit handler via a fini.
4743413 ld.so.1 doesn't terminate argv with NULL pointer when invoked directly
4746231 linker core-dumps when SECTION relocations are made against discarded
        sections
4730433 ld.so.1 wastes time repeatedly opening dependencies
4744337 missing RD_CONSISTENT event with dlmopen(LD_ID_NEWLM, ...)
4670835 rd_load_objiter can ignore callback's return value
4745932 strip utility doesn't strip out Dwarf2 debug section
4754751 "strip" command doesn't remove comdat stab sections.
4755674 Patch 109147-18 results in coredump.
------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-04
        Solaris/SunOS 5.7_sparc         patch T106950-21
        Solaris/SunOS 5.7_x86           patch T106951-21
------------------------------------------------------------------------------
4772927 strip core dumps on an archive library
4774727 direct-bindings can fail against copy-reloc symbols
------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-05
        Solaris/SunOS 5.9_x86           patch T113986-01
        Solaris/SunOS 5.8_sparc         patch T109147-21
        Solaris/SunOS 5.8_x86           patch T109148-21
        Solaris/SunOS 5.7_sparc         patch T106950-22
        Solaris/SunOS 5.7_x86           patch T106951-22
------------------------------------------------------------------------------

----------------------------------
Solaris 9 803 (4th Q-update - s9u4)
----------------------------------
Bugid   Risk Synopsis
==============================================================================
4730110 ld.so.1 list implementation could scale better
4728822 restrict the objects dlsym() searches.
        PSARC/2002/478 New dlopen(3dl) flag - RTLD_FIRST
4714146 crle: 64-bit secure pathname is incorrect.
4504895 dlclose() does not remove all objects
4698800 Wrong comments in /usr/lib/ld/sparcv9/map.*
4745129 dldump is inconsistent with .dynamic processing errors.
4753066 LD_SIGNAL isn't very useful in a threaded environment
        PSARC/2002/569 New dlinfo(3dl) flag - RTLD_DI_SIGNAL
4765536 crle: symbolic links can confuse alternative object configuration info
4766815 ld -r of object the TLS data fails
4770484 elfdump can not handle stripped archive file
4770494 The ld command gives improper error message handling broken archive
4775738 overwriting output relocation table when 'ld -zignore' is used
```

```
4778247 elfdump -e of core files fails
4779976 elfdump dies on bad relocation entries
4787579 invalid SHT_GROUP entries can cause linker to seg fault
4783869 dlclose: filter closure exhibits hang/failure - introduced with 4504895
4778418 ld.so.1: there be nits out there
4792461 Thread-Local Storage - x86 instruction sequence updates
        PSARC/2002/746 Thread-Local Storage - x86 instruction sequence updates
4461340 sgs: ugly build output while suppressing ia64 (64-bit) build on Intel
4790194 dlopen(..., RTLD_GROUP) has an odd interaction with interposition
4804328 auditing of threaded applications results in deadlock
4806476 building relocatable objects with SHF_EXCLUDE loses relocation
        information
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-06
        Solaris/SunOS 5.9_x86           patch T113986-02
        Solaris/SunOS 5.8_sparc         patch T109147-22
        Solaris/SunOS 5.8_x86           patch T109148-22
--------------------------------------------------------------------------------
4731183 compiler creates .tlsbss section instead of .tbss as documented
4816378 TLS: a tls test case dumps core with C and C++ compilers
4817314 TLS_GD relocations against local symbols do not reference symbol...
4811951 non-default symbol visibility overriden by definition in shared object
4802194 relocation error of mozilla built by K2 compiler
4715815 ld should allow linking with no output file (or /dev/null)
4793721 Need a way to null all code in ISV objects enabling ld performance
        tuning
--------------------------------------------------------------------------------
All the above changes plus:
        4796237 RFE: link-editor became extremely slow with patch 109147-20 and
                static libraries
are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-07
        Solaris/SunOS 5.9_x86           patch T113986-03
        Solaris/SunOS 5.8_sparc         patch T109147-23
        Solaris/SunOS 5.8_x86           patch T109148-23
--------------------------------------------------------------------------------

-----------------------------------
Solaris 9 1203 (5th Q-update - s9u5)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4830584 mmap for the padding region doesn't get freed after dlclose
4831650 ld.so.1 can walk off the end of it's call_init() array...
4831544 ldd using .so modules compiled with FD7 compiler caused a core dump
4834784 Accessing members in a TLS structure causes a core dump in Oracle
4824026 segv when -z combreloc is used with -xlinkopt
4825296 typo in elfdump
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-08
        Solaris/SunOS 5.9_x86           patch T113986-04
        Solaris/SunOS 5.8_sparc         patch T109147-24
        Solaris/SunOS 5.8_x86           patch T109148-24
--------------------------------------------------------------------------------
4470917 Solaris Process Model Unification (link-editor components only)
        PSARC/2002/117 Solaris Process Model Unification
4744411 Bloomberg wants a faster linker.
4811969 64-bit links can be much slower than 32-bit.
4825065 ld(1) should ignore consecutive empty sections.
4838226 unrelocated shared objects may be erroneously collected for init firing
4830889 TLS: testcase coredumps with -xarch=v9 and -g
4845764 filter removal can leave dangling filtee pointer
4811093 apptrace -F libc date core dumps
4826315 Link editors need to be pre- and post- Unified Process Model aware
```



```
4868300 interposing on direct bindings can fail
4872634 Large LD_PRELOAD values can cause SEGV of process
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-09
        Solaris/SunOS 5.9_x86           patch T113986-05
        Solaris/SunOS 5.8_sparc         patch T109147-25
        Solaris/SunOS 5.8_x86           patch T109148-25
--------------------------------------------------------------------------------

-----------------------------------
Solaris 9 404 (6th Q-update - s9u6)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4870260 The elfdump command should produce more warning message on invalid move
        entries.
4865418 empty PT_TLS program headers cause problems in TLS enabled applications
4825151 compiler core dumped with a -mt -xF=%all test
4845829 The runtime linker fails to dlopen() long path name.
4900684 shared libraries with more then 32768 plt's fail for sparc ELF64
4906062 Makefiles under usr/src/cmd/sgs needs to be updated
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-10
        Solaris/SunOS 5.9_x86           patch T113986-06
        Solaris/SunOS 5.8_sparc         patch T109147-26
        Solaris/SunOS 5.8_x86           patch T109148-26
        Solaris/SunOS 5.7_sparc         patch T106950-24
        Solaris/SunOS 5.7_x86           patch T106951-24
--------------------------------------------------------------------------------
4900320 rtld library mapping could be faster
4911775 implement GOTDATA proposal in ld
        PSARC/2003/477 SPARC GOTDATA instruction sequences
4904565 Functionality to ignore relocations against external symbols
4764817 add section types SHT_DEBUG and SHT_DEBUGSTR
        PSARC/2003/510 New ELF DEBUG and ANNOTATE sections
4850703 enable per-symbol direct bindings
4716275 Help required in the link analysis of runtime interfaces
        PSARC/2003/519 Link-editors: Direct Binding Updates
4904573 elfdump may hang when processing archive files
4918310 direct binding from an executable can't be interposed on
4918938 ld.so.1 has become SPARC32PLUS - breaks 4.x binary compatibility
4911796 S1S8 C++: ld dump core when compiled and linked with xlinkopt=1.
4889914 ld crashes with SEGV using -M mapfile under certain conditions
4911936 exception are not catch from shared library with -zignore
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-11
        Solaris/SunOS 5.9_x86           patch T113986-07
        Solaris/SunOS 5.8_sparc         patch T109147-27
        Solaris/SunOS 5.8_x86           patch T109148-27
        Solaris/SunOS 5.7_sparc         patch T106950-25
        Solaris/SunOS 5.7_x86           patch T106951-25
--------------------------------------------------------------------------------
4946992 ld crashes due to huge number of sections (>65,000)
4951840 mcs -c goes into a loop on executable program
4939869 Need additional relocation types for abs34 code model
        PSARC/2003/684 abs34 ELF relocations
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-12
        Solaris/SunOS 5.9_x86           patch T113986-08
        Solaris/SunOS 5.8_sparc         patch T109147-28
        Solaris/SunOS 5.8_x86           patch T109148-28
--------------------------------------------------------------------------------
```

```
 787 ----------------------------------
 788 Solaris 9 904 (7th Q-update - s9u7)
 789 ----------------------------------
 790 Bugid   Risk Synopsis
 791 ================================================================================
 792 4912214 Having multiple of libc.so.1 in a link map causes malloc() to fail
 793 4526878 ld.so.1 should pass MAP_ALIGN flag to give kernel more flexibility
 794 4930997 sgs bld_vernote.ksh script needs to be hardend...
 795 4796286 ld.so.1: scenario for trouble?
 796 4930985 clean up cruft under usr/src/cmd/sgs/tools
 797 4933300 remove references to Ultra-1 in librtld_db demo
 798 4936305 string table compression is much too slow...
 799 4939626 SUNWonld internal package must be updated...
 800 4939565 per-symbol filtering required
 801 4948119 ld(1) -z loadfltr fails with per-symbol filtering
 802 4948427 ld.so.1 gives fatal error when multiple RTLDINFO objects are loaded
 803 4940894 ld core dumps using "-xldscope=symbolic
 804 4955373 per-symbol filtering refinements
 805 4878827 crle(1M) - display post-UPM search paths, and compensate for pre-UPM.
 806 4955802 /usr/ccs/bin/ld dumps core in process_reld()
 807 4964415 elfdump issues wrong relocation error message
 808 4966465 LD_NOAUXFLTR fails when object is both a standard and auxiliary filter
 809 4973865 the link-editor does not scale properly when linking objects with
 810         lots of syms
 811 4975598 SHT_SUNW_ANNOTATE section relocation not resolved
 812 4974828 nss_files nss_compat r_mt tests randomly segfaulting
 813 --------------------------------------------------------------------------------
 814 All the above changes are incorporated in the following patches:
 815         Solaris/SunOS 5.9_sparc         patch T112963-13
 816         Solaris/SunOS 5.9_x86           patch T113986-09
 817 --------------------------------------------------------------------------------
 818 4860508 link-editors should create/promote/verify hardware capabilities
 819 5002160 crle: reservation for dumped objects gets confused by mmaped object
 820 4967869 linking stripped library causes segv in linker
 821 5006657 link-editor doesn't always handle nodirect binding syminfo information
 822 4915901 no way to see ELF information
 823 5021773 ld.so.1 has trouble with objects having more than 2 segments.
 824 --------------------------------------------------------------------------------
 825 All the above changes are incorporated in the following patches:
 826         Solaris/SunOS 5.9_sparc         patch T112963-14
 827         Solaris/SunOS 5.9_x86           patch T113986-10
 828         Solaris/SunOS 5.8_sparc         patch T109147-29
 829         Solaris/SunOS 5.8_x86           patch T109148-29
 830 --------------------------------------------------------------------------------
 831 All the above changes plus:
 832         6850124 dlopen reports "No such file or directory" in spite of ENOMEM
 833                 when mmap fails in anon_map()
 834 are incorporated in the following patches:
 835         Solaris/SunOS 5.9_sparc         patch TXXXXXX-XX
 836         Solaris/SunOS 5.9_x86           patch TXXXXXX-XX
 837 --------------------------------------------------------------------------------

 839 ----------
 840 Solaris 10
 841 ----------
 842 Bugid   Risk Synopsis
 843 ================================================================================
 844 5044797 ld.so.1: secure directory testing is being skipped during filtee
 845         processing
 846 4963676 Remove remaining static libraries
 847 5021541 unnecessary PT_SUNWBSS segment may be created
 848 5031495 elfdump complains about bad symbol entries in core files
 849 5012172 Need error when creating shared object with .o compiled
 850         -xarch=v9 -xcode=abs44
 851 4994738 rd_plt_resolution() resolves ebx-relative PLT entries incorrectly
```



```
 852 5023493 ld -m output with patch 109147-25 missing .o information
 853 --------------------------------------------------------------------------------
 854 All the above changes are incorporated in the following patches:
 855         Solaris/SunOS 5.9_sparc         patch T112963-15
 856         Solaris/SunOS 5.9_x86           patch T113986-11
 857         Solaris/SunOS 5.8_sparc         patch T109147-30
 858         Solaris/SunOS 5.8_x86           patch T109148-30
 859 --------------------------------------------------------------------------------
 860 5071614 109147-29 & -30 break the build of on28-patch on Solaris 8 2/04
 861 5029830 crle: provide for optional alternative dependencies.
 862 5034652 ld.so.1 should save, and print, more error messages
 863 5036561 ld.so.1 outputs non-fatal fatal message about auxiliary filter libraries
 864 5042713 4866170 broke ld.so's ::setenv
 865 5047082 ld can core dump on bad gcc objects
 866 5047612 ld.so.1: secure pathname verification is flawed with filter use
 867 5047235 elfdump can core dump printing PT_INTERP section
 868 4798376 nits in demo code
 869 5041446 gelf_update_*() functions inconsistently return NULL or 0
 870 5032364 M_ID_TLSBSS and M_ID_UNKNOWN have the same value
 871 4707030 Empty LD_PRELOAD_64 doesn't override LD_PRELOAD
 872 4968618 symbolic linkage causes core dump
 873 5062313 dladdr() can cause deadlock in MT apps.
 874 5056867 $ISALIST/$HWCAP expansion should be more flexible.
 875 4918303 0@0.so.1 should not use compiler-supplied crt*.o files
 876 5058415 whocalls cannot take more than 10 arguments
 877 5067518 The fix for 4918303 breaks the build if a new work space is used.
 878 --------------------------------------------------------------------------------
 879 All the above changes are incorporated in the following patches:
 880         Solaris/SunOS 5.9_sparc         patch T112963-16
 881         Solaris/SunOS 5.9_x86           patch T113986-12
 882         Solaris/SunOS 5.8_sparc         patch T109147-31
 883         Solaris/SunOS 5.8_x86           patch T109148-31
 884 --------------------------------------------------------------------------------
 885 5013759 *file* should report hardware/software capabilities (link-editor
 886         components only)
 887 5063580 libldstab: file /tmp/posto..: .stab[.index|.sbfocus] found with no
 888         matching stri
 889 5076838 elfdump(1) is built with a CTF section (the wrong one)
 890 5080344 Hardware capabilities are not enforced for a.out
 891 5079061 RTLD_DEFAULT can be expensive
 892         PSARC/2004/747 New dlsym(3c) Handle - RTLD_PROBE
 893 5064973 allow normal relocs against TLS symbols for some sections
 894 5085792 LD_XXXX_64 should override LD_XXXX
 895 5096272 every executable or library has a .SUNW_dof section
 896 5094135 Bloomberg wants a faster ldd.
 897 5086352 libld.so.3 should be built with a .SUNW_ctf ELF section, ready for CR
 898 5098205 elfdump gives wrong section name for the global offset table
 899 5092414 Linker patch 109147-29 makes Broadvison One-To-One server v4.1
 900         installation fail
 901 5080256 dump(1) doesn't list ELF hardware capabilities
 902 5097347 recursive read lock in gelf_getsym()
 903 --------------------------------------------------------------------------------
 904 All the above changes are incorporated in the following patches:
 905         Solaris/SunOS 5.9_sparc         patch T112963-17
 906         Solaris/SunOS 5.9_x86           patch T113986-13
 907         Solaris/SunOS 5.8_sparc         patch T109147-32
 908         Solaris/SunOS 5.8_x86           patch T109148-32
 909 --------------------------------------------------------------------------------
 910 5106206 ld.so.1 fail to run a Solaris9 program that has libc linked with
 911         -z lazyload
 912 5102601 ON should deliver a 64-bit operating system for Opteron systems
 913         (link-editor components only)
 914 6173852 enable link_auditing technology for amd64
 915 6174599 linker does not create .eh_frame_hdr sections for eh_frame sections
 916         with SHF_LINK_ORDER
 917 6175609 amd64 run-time linker has a corrupted note section
```

```
 6175843 amd64 rdb_demo files not installed
 6182293 ld.so.1 can repeatedly relocate object .plts (RTLD_NOW).
 6183645 ld core dumps when automounter fails
 6178667 ldd list unexpected (file not found) in x86 environment.
 6181928 Need new reloc types R_AMD64_GOTOFF64 and R_AMD64_GOTPC32
 6182884 AMD64: ld coredumps when building a shared library
 6173559 The ld may set incorrect value for sh_addralign under some conditions.
 5105601 ld.so.1 gets a little too enthusiastic with interposition
 6189384 ld.so.1 should accommodate a files dev/inode change (libc loopback mnt)
 6177838 AMD64: linker cannot resolve PLT for 32-bit a.out(s) on amd64-S2 kernel
 6190863 sparc disassembly code should be removed from rdb_demo
 6191488 unwind eh_frame_hdr needs corrected encoding value
 6192490 moe(1) returns /lib/libc.so.1 for optimal expansion of libc HWCAP
         libraries
 6192164 AMD64: introduce dlamd64getunwind interface
         PSARC/2004/747 libc::dlamd64getunwind()
 6195030 libdl has bad version name
 6195521 64-bit moe(1) missed the train
 6198358 AMD64: bad eh_frame_hdr data when C and C++ mixed in a.out
 6204123 ld.so.1: symbol lookup fails even after lazy loading fallback
 6207495 UNIX98/UNIX03 vsx namespace violation DYNL.hdr/misc/dlfcn/T.dlfcn
         14 Failed
 6217285 ctfmerge crashed during full onnv build
 ------------------------------------------------------------------------------

 -----------------------------------
 Solaris 10 106 (1st Q-update - s10u1)
 -----------------------------------
 Bugid   Risk Synopsis
 ==============================================================================
 6209350 Do not include signature section from dynamic dependency library into
         relocatable object
 6212797 The binary compiled on SunOS4.x doesn't run on Solaris8 with Patch
         109147-31
 ------------------------------------------------------------------------------
 All the above changes are incorporated in the following patches:
         Solaris/SunOS 5.9_sparc         patch T112963-18
         Solaris/SunOS 5.9_x86           patch T113986-14
         Solaris/SunOS 5.8_sparc         patch T109147-33
         Solaris/SunOS 5.8_x86           patch T109148-33
 ------------------------------------------------------------------------------
 6219538 112963-17: linker patch causes binary to dump core
 ------------------------------------------------------------------------------
 All the above changes are incorporated in the following patches:
         Solaris/SunOS 5.10_sparc        patch T117461-01
         Solaris/SunOS 5.10_x86          patch T118345-01
         Solaris/SunOS 5.9_sparc         patch T112963-19
         Solaris/SunOS 5.9_x86           patch T113986-15
         Solaris/SunOS 5.8_sparc         patch T109147-34
         Solaris/SunOS 5.8_x86           patch T109148-34
 ------------------------------------------------------------------------------
 6257177 incremental builds of usr/src/cmd/sgs can fail...
 6219651 AMD64: Linker does not issue error for out of range R_AMD64_PC32
 ------------------------------------------------------------------------------
 All the above changes are incorporated in the following patches:
         Solaris/SunOS 5.10_sparc        patch T117461-02
         Solaris/SunOS 5.10_x86          patch T118345-02
         Solaris/SunOS 5.9_sparc         patch T112963-20
         Solaris/SunOS 5.9_x86           patch T113986-16
         Solaris/SunOS 5.8_sparc         patch T109147-35
         Solaris/SunOS 5.8_x86           patch T109148-35
 NOTE: The fix for 6219651 is only applicable for 5.10_x86 platform.
 ------------------------------------------------------------------------------
 5080443 lazy loading failure doesn't clean up after itself (D)
 6226206 ld.so.1 failure when processing single segment hwcap filtee
 6228472 ld.so.1: link-map control list stacking can loose objects
```



```
 6235000 random packages not getting installed in snv_09 and snv_10 -
         rtld/common/malloc.c Assertion
 6219317 Large page support is needed for mapping executables, libraries and
         files (link-editor components only)
 6244897 ld.so.1 can't run apps from commandline
 6251798 moe(1) returns an internal assertion failure message in some
         circumstances
 6251722 ld fails silently with exit 1 status when -z ignore passed
 6254364 ld won't build libgenunix.so with absolute relocations
 6215444 ld.so.1 caches "not there" lazy libraries, foils svc.startd(1M)'s logic
 6222525 dlsym(3C) trusts caller(), which may return wrong results with tail call
         optimization
 6241995 warnings in sgs should be fixed (link-editor components only)
 6258834 direct binding availability should be verified at runtime
 6260361 lari shouldn't count a.out non-zero undefined entries as interesting
 6260780 ldd doesn't recognize LD_NOAUXFLTR
 6266261 Add ld(1) -Bnodirect support (D)
 6261990 invalid e_flags error could be a little more friendly
 6261803 lari(1) should find more events uninteresting (D)
 6267352 libld_malloc provides inadequate alignment
 6268693 SHN_SUNW_IGNORE symbols should be allowed to be mulitiply defined
 6262789 Infosys wants a faster linker
 ------------------------------------------------------------------------------
 All the above changes are incorporated in the following patches:
         Solaris/SunOS 5.10_sparc        patch T117461-03
         Solaris/SunOS 5.10_x86          patch T118345-03
         Solaris/SunOS 5.9_sparc         patch T112963-21
         Solaris/SunOS 5.9_x86           patch T113986-17
         Solaris/SunOS 5.8_sparc         patch T109147-36
         Solaris/SunOS 5.8_x86           patch T109148-36
 ------------------------------------------------------------------------------
 6283601 The usr/src/cmd/sgs/packages/common/copyright contains old information
         legally problematic
 6276905 dlinfo gives inconsistent results (relative vs absolute linkname) (D)
         PSARC/2005/357 dlinfo(3c) RTLD_DI_ARGSINFO
 6284941 excessive link times with many groups/sections
 6280467 dlclose() unmaps shared library before library's _fini() has finished
 6291547 ld.so mishandles LD_AUDIT causing security problems.
 ------------------------------------------------------------------------------
 All the above changes are incorporated in the following patches:
         Solaris/SunOS 5.10_sparc        patch T117461-04
         Solaris/SunOS 5.10_x86          patch T118345-04
         Solaris/SunOS 5.9_sparc         patch T112963-22
         Solaris/SunOS 5.9_x86           patch T113986-18
         Solaris/SunOS 5.8_sparc         patch T109147-37
         Solaris/SunOS 5.8_x86           patch T109148-37
 ------------------------------------------------------------------------------
 6295971 UNIX98/UNIX03 *vsx* DYNL.hdr/misc/dlfcn/T.dlfcn 14 fails, auxv.h syntax
         error
 6299525 .init order failure when processing cycles
 6273855 gcc and sgs/crle don't get along
 6273864 gcc and sgs/libld don't get along
 6273875 gcc and sgs/rtld don't get along
 6272563 gcc and amd64/krtld/doreloc.c don't get along
 6290157 gcc and sgs/librtld_db/rdb_demo don't get along
 6301218 Matlab dumps core on startup when running on 112963-22 (D)
 ------------------------------------------------------------------------------
 All the above changes are incorporated in the following patches:
         Solaris/SunOS 5.10_sparc        patch T117461-06
         Solaris/SunOS 5.10_x86          patch T118345-08
         Solaris/SunOS 5.9_sparc         patch T112963-23
         Solaris/SunOS 5.9_x86           patch T113986-19
         Solaris/SunOS 5.8_sparc         patch T109147-38
         Solaris/SunOS 5.8_x86           patch T109148-38
 ------------------------------------------------------------------------------
 6314115 Checkpoint refuses to start, crashes on start, after application of
```

```
1050         linker patch 112963-22
1051 ------------------------------------------------------------------------------
1052 All the above changes are incorporated in the following patches:
1053         Solaris/SunOS 5.9_sparc         patch T112963-24
1054         Solaris/SunOS 5.9_x86           patch T113986-20
1055         Solaris/SunOS 5.8_sparc         patch T109147-39
1056         Solaris/SunOS 5.8_x86           patch T109148-39
1057 ------------------------------------------------------------------------------
1058 6318306 a dlsym() from a filter should be redirected to an associated filtee
1059 6318401 mis-aligned TLS variable
1060 6324019 ld.so.1: malloc alignment is insufficient for new compilers
1061 6324589 psh coredumps on x86 machines on snv_23
1062 6236594 AMD64: Linker needs to handle the new .lbss section (D)
1063         PSARC 2005/514 AMD64 - large section support
1064 6314743 Linker: incorrect resolution for R_AMD64_GOTPC32
1065 6311865 Linker: x86 medium model; invalid ELF program header
1066 ------------------------------------------------------------------------------
1067 All the above changes are incorporated in the following patches:
1068         Solaris/SunOS 5.10_sparc        patch T117461-07
1069         Solaris/SunOS 5.10_x86          patch T118345-12
1070 ------------------------------------------------------------------------------
1071 6309061 link_audit should use __asm__ with gcc
1072 6310736 gcc and sgs/libld don't get along on SPARC
1073 6329796 Memory leak with iconv_open/iconv_close with patch 109147-33
1074 6332983 s9 linker patches 112963-24/113986-20 causing cluster machines not
1075         to boot
1076 ------------------------------------------------------------------------------
1077 All the above changes are incorporated in the following patches:
1078         Solaris/SunOS 5.10_sparc        patch T117461-08
1079         Solaris/SunOS 5.10_x86          patch T121208-02
1080         Solaris/SunOS 5.9_sparc         patch T112963-25
1081         Solaris/SunOS 5.9_x86           patch T113986-21
1082         Solaris/SunOS 5.8_sparc         patch T109147-40
1083         Solaris/SunOS 5.8_x86           patch T109148-40
1084 ------------------------------------------------------------------------------
1085 6445311 The sparc S8/S9/S10 linker patches which include the fix for the
1086         CR6222525 are hit by the CR6439613.
1087 ------------------------------------------------------------------------------
1088 All the above changes are incorporated in the following patches:
1089         Solaris/SunOS 5.9_sparc         patch T112963-26
1090         Solaris/SunOS 5.8_sparc         patch T109147-41
1091 ------------------------------------------------------------------------------

1093 ------------------------------------
1094 Solaris 10 807 (4th Q-update - s10u4)
1095 ------------------------------------
1096 Bugid   Risk Synopsis
1097 ==============================================================================
1098 6487273 ld.so.1 may open arbitrary locale files when relative path is built
1099         from locale environment vars
1100 6487284 ld.so.1: buffer overflow in doprf() function
1101 ------------------------------------------------------------------------------
1102 All the above changes are incorporated in the following patches:
1103         Solaris/SunOS 5.10_sparc        patch T124922-01
1104         Solaris/SunOS 5.10_x86          patch T124923-01
1105         Solaris/SunOS 5.9_sparc         patch T112963-27
1106         Solaris/SunOS 5.9_x86           patch T113986-22
1107         Solaris/SunOS 5.8_sparc         patch T109147-42
1108         Solaris/SunOS 5.8_x86           patch T109148-41
1109 ------------------------------------------------------------------------------
1110 6477132 ld.so.1: memory leak when running set*id application
1111 ------------------------------------------------------------------------------
1112 All the above changes are incorporated in the following patches:
1113         Solaris/SunOS 5.10_sparc        patch T124922-02
1114         Solaris/SunOS 5.10_x86          patch T124923-02
1115         Solaris/SunOS 5.9_sparc         patch T112963-30
```



```
1116         Solaris/SunOS 5.9_x86           patch T113986-24
1117 ------------------------------------------------------------------------------
1118 6340814 ld.so.1 core dump with HWCAP relocatable object + updated statistics
1119 6307274 crle bug with LD_LIBRARY_PATH
1120 6317969 elfheader limited to 65535 segments (link-editor components only)
1121 6350027 ld.so.1 aborts with assertion failed on amd64
1122 6362044 ld(1) inconsistencies with LD_DEBUG=-Dunused and -zignore
1123 6362047 ld.so.1 dumps core when combining HWCAP and LD_PROFILE
1124 6304206 runtime linker may respect LANG and LC_MESSAGE more than LC_ALL
1125 6363495 Catchup required with Intel relocations
1126 6326497 ld.so not properly processing LD_LIBRARY_PATH ending in :
1127 6307146 mcs dumps core when appending null string to comment section
1128 6371877 LD_PROFILE_64 with gprof does not produce correct results on amd64
1129 6372082 ld -r erroneously creates .got section on i386
1130 6201866 amd64: linker symbol elimination is broken
1131 6372620 printstack() segfaults when called from static function (D)
1132 6380470 32-bit ld(1) incorrectly builds 64-bit relocatable objects
1133 6391407 Insufficient alignment of 32-bit object in archive makes ld segfault
1134         (libelf component only) (D)
1135 6316708 LD_DEBUG should provide a means of identifying/isolating individual
1136         link-map lists (P)
1137 6280209 elfdump cores on memory model 0x3
1138 6197234 elfdump and dump don't handle 64-bit symbols correctly
1139 6398893 Extended section processing needs some work
1140 6397256 ldd dumps core in elf_fix_name
1141 6327926 ld does not set etext symbol correctly for AMD64 medium model (D)
1142 6390410 64-bit LD_PROFILE can fail: relocation error when binding profile plt
1143 6382945 AMD64-GCC: dbx: internal error: dwarf reference attribute out of bounds
1144 6262333 init section of .so dlopened from audit interface not being called
1145 6409613 elf_outsync() should fsync()
1146 6426048 C++ exceptions broken in Nevada for amd64
1147 6429418 ld.so.1: need work-around for Nvidia drivers use of static TLS
1148 6429504 crle(1) shows wrong defaults for non-existent 64-bit config file
1149 6431835 data corruption on x64 in 64-bit mode while LD_PROFILE is in effect
1150 6423051 static TLS support within the link-editors needs a major face lift (D)
1151 6388946 attempting to dlopen a .o file mislabeled as .so fails
1152 6446740 allow mapfile symbol definitions to create backing storage (D)
1153 4986360 linker crash on exec of .so (as opposed to a.out) -- error preferred
1154         instead
1155 6229145 ld: initarray/finiarray processing occurs after got size is determined
1156 6324924 the linker should warn if there's a .init section but not _init
1157 6424132 elfdump inserts extra whitespace in bitmap value display
1158 6449485 ld(1) creates misaligned TLS in binary compiled with -xpg
1159 6424550 Write to unallocated (wua) errors when libraries are built with
1160         -z lazyload
1161 6464235 executing the 64-bit ld(1) should be easy (D)
1162 6465623 need a way of building unix without an interpreter
1163 6467925 ld: section deletion (-z ignore) requires improvement
1164 6357230 specfiles should be nuked (link-editor components only)
1165 ------------------------------------------------------------------------------
1166 All the above changes are incorporated in the following patches:
1167         Solaris/SunOS 5.10_sparc        patch T124922-03
1168         Solaris/SunOS 5.10_x86          patch T124923-03

1170 These patches also include the framework changes for the following bug fixes.
1171 However, the associated feature has not been enabled in Solaris 10 or earlier
1172 releases:

1174 6174390 crle configuration files are inconsistent across platforms (D, P)
1175 6432984 ld(1) output file removal - change default behavior (D)
1176         PSARC/2006/353 ld(1) output file removal - change default behavior
1177 ------------------------------------------------------------------------------

1179 ------------------------------------
1180 Solaris 10 508 (5th Q-update - s10u5)
1181 ------------------------------------
```

```
Bugid   Risk Synopsis
================================================================================
6561987 data vac_conflict faults on lipthread libthread libs in s10.
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T127111-01
        Solaris/SunOS 5.10_x86          patch T127112-01
--------------------------------------------------------------------------------
6501793 GOTOP relocation transition (optimization) fails with offsets > 2^32
6532924 AMD64: Solaris 5.11 55b: SEGV after whocatches
6551627 OGL: SIGSEGV when trying to use OpenGL pipeline with splash screen,
        Solaris/Nvidia only
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T127111-04
        Solaris/SunOS 5.10_x86          patch T127112-04
--------------------------------------------------------------------------------
6479848 Enhancements to the linker support interface needed. (D)
        PSARC/2006/595 link-editor support library interface - ld_open()
6521608 assertion failure in runtime linker related to auditing
6494228 pclose() error when an audit library calls popen() and the main target
        is being run under ldd (D)
6568745 segfault when using LD_DEBUG with bit_audit library when instrumenting
        mozilla (D)
        PSARC/2007/413 Add -zglobalaudit option to ld
6602294 ps_pbrandname breaks apps linked directly against librtld_db
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T127111-07
        Solaris/SunOS 5.10_x86          patch T127112-07
--------------------------------------------------------------------------------

------------------------------------
Solaris 10 908 (6th Q-update - s10u6)
------------------------------------
Bugid   Risk Synopsis
================================================================================
6672544 elf_rtbndr must support non-ABI aligned stacks on amd64
6668050 First trip through PLT does not preserve args in xmm registers
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patch:
        Solaris/SunOS 5.10_x86          patch T137138-01
--------------------------------------------------------------------------------

------------------------------------
Solaris 10 409 (7th Q-update - s10u7)
------------------------------------
Bugid   Risk Synopsis
================================================================================
6629404 ld with -z ignore doesn't scale
6606203 link editor ought to allow creation of >2gb sized objects (P)
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T139574-01
        Solaris/SunOS 5.10_x86          patch T139575-01
--------------------------------------------------------------------------------
6746674 setuid applications do not find libraries any more because trusted
        directories behavior changed (D)
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T139574-02
        Solaris/SunOS 5.10_x86          patch T139575-02
--------------------------------------------------------------------------------
6703683 Can't build VirtualBox on Build 88 or 89
6737579 process_req_lib() in libld consumes file descriptors
6685125 ld/elfdump do not handle ZERO terminator .eh_frame amd64 unwind entry
```



```
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T139574-03
        Solaris/SunOS 5.10_x86          patch T139575-03
--------------------------------------------------------------------------------

------------------------------------
Solaris 10 1009 (8th Q-update - s10u8)
------------------------------------
Bugid   Risk Synopsis
================================================================================
6782597 32-bit ld.so.1 needs to accept objects with large inode number
6805502 The addition of "inline" keywords to sgs code broke the lint
        verification in S10
6807864 ld.so.1 is susceptible to a fatal dlsym()/setlocale() race
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T141692-01
        Solaris/SunOS 5.10_x86          patch T141693-01
NOTE: The fix for 6805502 is only applicable to s10.
--------------------------------------------------------------------------------
6826410 ld needs to sort sections using 32-bit sort keys
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T141771-01
        Solaris/SunOS 5.10_x86          patch T141772-01
NOTE: The fix for 6826410 is also available for s9 in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-33
        Solaris/SunOS 5.9_x86           patch T113986-27
--------------------------------------------------------------------------------
6568447 bcp is broken by 6551627
6599700 librtld_db needs better plugin support
6713830 mdb dumped core reading a gcore
6756048 rd_loadobj_iter() should always invoke brand plugin callback
6786744 32-bit dbx failed with unknown rtld_db.so error on snv_104
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T141444-06
        Solaris/SunOS 5.10_x86          patch T141445-06
--------------------------------------------------------------------------------

------------------------------------
Solaris 10 1005 (9th Q-update - s10u9)
------------------------------------
Bugid   Risk Synopsis
================================================================================
6850124 dlopen reports "No such file or directory" in spite of ENOMEM
        when mmap fails in anon_map()
6826513 ldd gets confused by a crle(1) LD_PRELOAD setting
6684577 ld should propagate SHF_LINK_ORDER flag to ET_REL objects
6524709 executables using /usr/lib/libc.so.1 as the ELF interpreter dump core
        (link-editor components only)
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T143895-01
        Solaris/SunOS 5.10_x86          patch T143896-01
--------------------------------------------------------------------------------

------------------------------------
Solaris 10 XXXX (10th Q-update - s10u10)
------------------------------------
Bugid   Risk Synopsis
================================================================================
6478684 isainfo/cpuid reports pause instruction not supported on amd64
        PSARC/2010/089 Removal of AV_386_PAUSE and AV_386_MON
--------------------------------------------------------------------------------
```

All the above changes are incorporated in the following patches:

Solaris/SunOS 5.10_sparc	patch TXXXXXX-XX
Solaris/SunOS 5.10_x86	patch TXXXXXX-XX

Solaris Nevada (OpenSolaris 2008.05, snv_86)

Bugid	Risk	Synopsis
6409350		BrandZ project integration into Solaris (link-editor components only)
6459189		UNIX03: *VSC* c99 compiler overwrites non-writable file
6423746		add an option to relax the resolution of COMDAT relocs (D)
4934427		runtime linker should load up static symbol names visible to dladdr() (D)
		PSARC 2006/526 SHT_SUNW_LDYNSYM - default local symbol addition
6448719		sys/elf.h could be updated with additional machine and ABI types
6336605		link-editors need to support R_*_SIZE relocations
		PSARC/2006/558 R_*_SIZE relocation support
6475375		symbol search optimization to reduce rescans
6475497		elfdump(1) is misreporting sh_link
6482058		lari(1) could be faster, and handle per-symbol filters better
6482974		defining virtual address of text segment can result in an invalid data segment
6476734		crle(1m) "-l" as described fails system, crle cores trying to fix /a/var/ld/ld.config in failsafe
6487499		link_audit "make clobber" creates and populates proto area
6488141		ld(1) should detect attempt to reference 0-length .bss section
6496718		restricted visibility symbol references should trigger archive extraction
6515970		HWCAP processing doesn’t clean up fmap structure - browser fails to run java applet
6494214		Refinements to symbolic binding, symbol declarations and interposition (D)
		PSARC/2006/714 ld(1) mapfile: symbol interpose definition
6475344		DTrace needs ELF function and data symbols sorted by address (D)
		PSARC/2007/026 ELF symbol sort sections
6518480		ld -melf_i386 doesn’t complain (D)
6519951		bfu is just another word for exit today (RPATH -> RUNPATH conversion bites us) (D)
6521504		ld: hardware capabilities processing from relocatables objects needs hardening.
6518322		Some ELF utilities need updating for .SUNW_ldynsym section (D)
		PSARC/2007/074 -L option for nm(1) to display SHT_SUNW_LDYNSYM symbols
6523787		dlopen() handle gets mistakenly orphaned - results in access to freed memory
6531189		SEGV in dladdr()
6527318		dlopen(name, RTLD_NOLOAD) returns handle for unloaded library
6518359		extern mapfiles references to _init/_fini can create INIT/FINI addresses of 0
6533587		ld.so.1: init/fini processing needs to compensate for interposer expectations
6516118		Reserved space needed in ELF dynamic section and string table (D)
		PSARC/2007/127 Reserved space for editing ELF dynamic sections
6535688		elfdump could be more robust in the face of Purify (D)
6516665		The link-editors should be more resilient against gcc’s symbol versioning
6541004		hwcap filter processing can leak memory
5108874		elfdump SEGVs on bad object file
6547441		Uninitialized variable causes ld.so.1 to crash on object cleanup
6341667		elfdump should check alignments of ELF header elements
6387860		elfdump cores, when processing linux built ELF file
6198202		mcs -d dumps core
6246083		elfdump should allow section index specification (numeric -N equivalent) (D)
		PSARC/2007/247 Add -I option to elfdump

6556563		elfdump section overlap checking is too slow for large files
5006034		need ?E mapfile feature extension (D)
6565476		rtld symbol version check prevents GNU ld binary from running
6567670		ld(1) symbol size/section size verification uncovers Haskell compiler inconsistency
6530249		elfdump should handle ELF files with no section header table (D)
		PSARC/2007/395 Add -P option to elfdump
6573641		ld.so.1 does not maintain parent relationship to a dlopen() caller.
6577462		Additional improvements needed to handling of gcc’s symbol versioning
6583742		ELF string conversion library needs to lose static writable buffers
6589819		ld generated reference to __tls_get_addr() fails when resolving to a shared object reference
6595139		various applications should export yy* global variables for libl
		PSARC/2007/474 new ldd(1) -w option
6597841		gelf_getdyn() reads one too many dynamic entries
6603313		dlclose() can fail to unload objects after fix for 6573641
6234471		need a way to edit ELF objects (D)
		PSARC/2007/509 elfedit
5035454		mixing -Kpic and -KPIC may cause SIGSEGV with -xarch=v9
6473571		strip and mcs get confused and corrupt files when passed non-ELF arguments
6253589		mcs has problems handling multiple SHT_NOTE sections
6610591		do_reloc() should not require unused arguments
6602451		new symbol visibilities required: EXPORTED, SINGLETON and ELIMINATE (D)
		PSARC/2007/559 new symbol visibilities - EXPORTED, SINGLETON, and ELIMINATE
6570616		elfdump should display incorrectly aligned note section
6614968		elfedit needs string table module (D)
6620533		HWCAP filtering can leave uninitialized data behind - results in "rejected: Invalid argument"
6617855		nodirect tag can be ignored when other syminfo tags are available (link-editor components only)
6621066		Reduce need for new elfdump options with every section type (D)
		PSARC/2007/620 elfdump -T, and simplified matching
6627765		soffice failure after integration of 6603313 - dangling GROUP pointer.
6319025		SUNWbtool packaging issues in Nevada and S10u1.
6626135		elfedit capabilities str->value mapping should come from usr/src/common/elfcap
6642769		ld(1) -z combreloc should become default behavior (D)
		PSARC/2008/006 make ld(1) -z combreloc become default behavior
6634436		XFFLAG should be updated. (link-editor components only)
6492726		Merge SHF_MERGE|SHF_STRINGS input sections (D)
4947191		OSNet should use direct bindings (link-editor components only)
6654381		lazy loading fall-back needs optimizing
6658385		ld core dumps when building Xorg on nv_82
6516808		ld.so.1’s token expansion provides no escape for platforms that don’t report HWCAP
6668534		Direct bindings can compromise function address comparisons from executables
6667661		Direct bindings can compromise executables with insufficient copy relocation information
6357282		ldd should recognize PARENT and EXTERN symbols (D)
		PSARC/2008/148 new ldd(1) -p option
6672394		ldd(1) unused dependency processing is tricked by relocations errors

---

Solaris Nevada (OpenSolaris 2008.11, snv_101)

Bugid	Risk	Synopsis
6671255		link-editor should support cross linking (D)
		PSARC/2008/179 cross link-editor
6674666		elfedit dyn:posflag1 needs option to locate element via NEEDED item
6675591		elfwrap - wrap data in an ELF file (D,P)
		PSARC/2008/198 elfwrap - wrap data in an ELF file

```
1446 6678244 elfdump dynamic section sanity checking needs refinement
1447 6679212 sgs use of SCCS id for versioning is obstacle to mercurial migration
1448 6681761 lies, darn lies, and linker README files
1449 6509323 Need to disable the Multiple Files loading - same name, different
1450         directories (or its stat() use)
1451 6686889 ld.so.1 regression - bad pointer created with 6509323 integration
1452 6695681 ldd(1) crashes when run from a chrooted environment
1453 6516212 usr/src/cmd/sgs/libelf warlock targets should be fixed or abandoned
1454 6678310 using LD_AUDIT, ld.so.1 calls shared library's .init before library is
1455         fully relocated (link-editor components only)
1456 6699594 The ld command has a problem handling 'protected' mapfile keyword.
1457 6699131 elfdump should display core file notes (D)
1458 6702260 single threading .init/.fini sections breaks staroffice
1459 6703919 boot hangs intermittently on x86 with onnv daily.0430 and on
1460 6701798 ld can enter infinite loop processing bad mapfile
1461 6706401 direct binding copy relocation fallback is insufficient for ild
1462         generated objects
1463 6705846 multithreaded C++ application seems to get deadlocked in the dynamic
1464         linker code
1465 6686343 ldd(1) - unused search path diagnosis should be enabled
1466 6712292 ld.so.1 should fall back to an interposer for failed direct bindings
1467 6716350 usr/src/cmd/sgs should be linted by nightly builds
1468 6720509 usr/src/cmd/sgs/sgsdemangler should be removed
1469 6617475 gas creates erroneous FILE symbols [was: ld.so.1 is reported as
1470         false positive by wsdiff]
1471 6724311 dldump() mishandles R_AMD64_JUMP_SLOT relocations
1472 6724774 elfdump -n doesn't print siginfo structure
1473 6728555 Fix for amd64 aw (6617475) breaks pure gcc builds
1474 6734598 ld(1) archive processing failure due to mismatched file descriptors (D)
1475 6735939 ld(1) discarded symbol relocations errors (Studio and GNU).
1476 6354160 Solaris linker includes more than one copy of code in binary when
1477         linking gnu object code
1478 6744003 ld(1) could provide better argument processing diagnostics (D)
1479         PSARC 2008/583 add gld options to ld(1)
1480 6749055 ld should generate GNU style VERSYM indexes for VERNEED records (D)
1481         PSARC/2008/603 ELF objects to adopt GNU-style Versym indexes
1482 6752728 link-editor can enter UNDEF symbols in symbol sort sections
1483 6756472 AOUT search path pruning (D)
1484 --------------------------------------------------------------------------------

1486 ---------------------------------------------
1487 Solaris Nevada (OpenSolaris 2009.06, snv_111)
1488 ---------------------------------------------
1489 Bugid   Risk Synopsis
1490 ================================================================================

1492 6754965 introduce the SF1_SUNW_ADDR32 bit in software capabilities (D)
1493         (link-editor components only)
1494         PSARC/2008/622 32-bit Address Restriction Software Capabilities Flag
1495 6756953 customer requests that DT_CONFIG strings be honored for secure apps (D)
1496 6765299 ld --version-script option not compatible with GNU ld (D)
1497 6748160 problem with -zrescan (D)
1498         PSARC/2008/651 New ld archive rescan options
1499 6763342 sloppy relocations need to get sloppier
1500 6736890 PT_SUNWBSS should be disabled (D)
1501         PSARC/2008/715 PT_SUNWBSS removal
1502 6772661 ldd/lddstub/ld.so.1 dump core in current nightly while processing
1503         libsoftcrypto_hwcap.so.1
1504 6765931 mcs generates unlink(NULL) system calls
1505 6775062 remove /usr/lib/libldstab.so (D)
1506 6782977 ld segfaults after support lib version error sends bad args to vprintf()
1507 6773695 ld -z nopartial can break non-pic objects
1508 6778453 RTLD_GROUP prevents use of application defined malloc
1509 6789925 64-bit applications with SF1_SUNW_ADDR32 require non-default starting
1510         address
1511 6792906 ld -z nopartial fix breaks TLS
```



```
1512 6686372 ld.so.1 should use mmapobj(2)
1513 6726108 dlopen() performance could be improved.
1514 6792836 ld is slow when processing GNU linkonce sections
1515 6797468 ld.so.1: orphaned handles aren't processed correctly
1516 6798676 ld.so.1: enters infinite loop with realloc/defragmentation logic
1517 6237063 request extension to dl* family to provide segment bounds
1518         information (D)
1519         PSARC/2009/054 dlinfo(3c) - segment mapping retrieval
1520 6800388 shstrtab can be sized incorrectly when -z ignore is used
1521 6805009 ld.so.1: link map control list tear down leaves dangling pointer -
1522         pfinstall does it again.
1523 6807050 GNU linkonce sections can create duplicate and incompatible
1524         eh_frame FDE entries
1525 --------------------------------------------------------------------------------

1527 --------------
1528 Solaris Nevada
1529 --------------
1530 Bugid   Risk Synopsis
1531 ================================================================================
1532 6813909 generalize eh_frame support to non-amd64 platforms
1533 6801536 ld: mapfile processing oddities unveiled through mmapobj(2) observations
1534 6802452 libelf shouldn't use MS_SYNC
1535 6818012 nm tries to modify readonly segment and dumps core
1536 6821646 xVM dom0 doesn't boot on daily.0324 and beyond
1537 6822828 librtld_db can return RD_ERR before RD_NOMAPS, which compromises dbx
1538         expectations.
1539 6821619 Solaris linkers need systematic approach to ELF OSABI (D)
1540         PSARC/2009/196 ELF objects to set OSABI / elfdump -O option
1541 6827468 6801536 breaks 'ld -s' if there are weak/strong symbol pairs
1542 6715578 AOUT (BCP) symbol lookup can be compromised with lazy loading.
1543 6752883 ld.so.1 error message should be buffered (not sent to stderr).
1544 6577982 ld.so.1 calls getpid() before it should when any LD_* are set
1545 6831285 linker LD_DEBUG support needs improvements (D)
1546 6806791 filter builds could be optimized (link-editor components only)
1547 6823371 calloc() uses suboptimal memset() causing 15% regression in SpecCPU2006
1548         gcc code (link-editor components only)
1549 6831308 ld.so.1: symbol rescanning does a little too much work
1550 6837777 ld ordered section code uses too much memory and works too hard
1551 6841199 Undo 10 year old workaround and use 64-bit ld on 32-bit objects
1552 6784790 ld should examine archives to determine output object class/machine (D)
1553         PSARC/2009/305 ld -32 option
1554 6849998 remove undocumented mapfile $SPECVERS and $NEED options
1555 6851224 elf_getshnum() and elf_getshstrndx() incompatible with 2002 ELF gABI
1556         agreement (D)
1557         PSARC/2009/363 replace elf_getphnum, elf_getshnum, and elf_getshstrndx
1558 6853809 ld.so.1: rescan fallback optimization is invalid
1559 6854158 ld.so.1: interposition can be skipped because of incorrect
1560         caller/destination validation
1561 6862967 rd_loadobj_iter() failing for core files
1562 6856173 streams core dumps when compiled in 64bit with a very large static
1563         array size
1564 6834197 ld pukes when given an empty plate
1565 6516644 per-symbol filtering shouldn't be allowed in executables
1566 6878605 ld should accept '%' syntax when matching input SHT_PROGBITS sections
1567 6850768 ld option to autogenerate wrappers/interposers similar to GNU ld
1568         --wrap (D)
1569         PSARC/2009/493 ld -z wrap option
1570 6888489 Null environment variables are not overriding crle(1) replaceable
1571         environment variables.
1572 6885456 Need to implement GNU-ld behavior in construction of .init/.fini
1573         sections
1574 6900241 ld should track SHT_GROUP sections by symbol name, not section name
1575 6901773 Special handling of STT_SECTION group signature symbol for GNU objects
1576 6901895 Failing asserts in ld update_osym() trying to build gcc 4.5 develpment
1577         head
```

```
1578 6909523 core dump when run "LD_DEBUG=help ls" in non-English locale
1579 6903688 mdb(1) can't resolve certain symbols in solaris10-branded processes
1580         from the global zone
1581 6923449 elfdump misinterprets _init/_fini symbols in dynamic section test
1582 6914728 Add dl_iterate_phdr() function to ld.so.1 (D)
1583         PSARC/2010/015 dl_iterate_phdr
1584 6916788 ld version 2 mapfile syntax (D)
1585         PSARC/2009/688 Human readable and extensible ld mapfile syntax
1586 6929607 ld generates incorrect VERDEF entries for ET_REL output objects
1587 6924224 linker should ignore SUNW_dof when calculating the elf checksum
1588 6918143 symbol capabilities (D)
1589         PSARC/2010/022 Linker-editors: Symbol Capabilities
1590 6910387 .tdata and .tbss separation invalidates TLS program header information
1591 6934123 elfdump -d coredumps on PA-RISC elf
1592 6931044 ld should not allow SHT_PROGBITS .eh_frame sections on amd64 (D)
1593 6931056 pvs -r output can include empty versions in output
1594 6938628 ld.so.1 should produce diagnostics for all dl*() entry points
1595 6938111 nm 'No symbol table data' message goes to stdout
1596 6941727 ld relocation cache memory use is excessive
1597 6932220 ld -z allextract skips objects that lack global symbols
1598 6943772 Testing for a symbols existence with RTLD_PROBE is compromised by
1599         RTLD_BIND_NOW
1600         PSARC/2010/XXX Deferred symbol references
1601 6943432 dlsym(RTLD_PROBE) should only bind to symbol definitions
1602 6668759 an external method for determining whether an ELF dependency is optional
1603 6954032 Support library with ld_open and -z allextract in snv_139 do not mix
1604 6949596 wrong section alignment generated in joint compilation with shared
1605         library
1606 6961755 ld.so.1's -e arguments should take precedence over environment
1607         variables. (D)
1608 6748925 moe returns wrong hwcap library in some circumstances
1609 6916796 OSnet mapfiles should use version 2 link-editor syntax
1610 6964517 OSnet mapfiles should use version 2 link-editor syntax (2nd pass)
1611 6948720 SHT_INIT_ARRAY etc. section names don't follow ELF gABI (D)
1612 6962343 sgsmsg should use mkstemp() for temporary file creation
1613 6965723 libsoftcrypto symbol capabilities rely on compiler generated
1614         capabilities - gcc failure (link-editor components only)
1615 6952219 ld support for archives larger than 2 GB (D, P)
1616         PSARC/2010/224 Support for archives larger than 2 GB
1617 6956152 dlclose() from an auditor can be fatal. Preinit/activity events should
1618         be more flexible. (D)
1619 6971440 moe can core dump while processing libc.
1620 6972234 sgs demo's could use some cleanup
1621 6935867 .dynamic could be readonly in sharable objects
1622 6975290 ld mishandles GOT relocation against local ABS symbol
1623 6972860 ld should provide user guidance to improve objects (D)
1624         PSARC/2010/312 Link-editor guidance
1625 --------------------------------------------------------------------------------

1627 --------------
1628 Illumos
1629 --------------
1630 Bugid   Risk Synopsis
1631 ================================================================================

1633 308     ld may misalign sections only preceded by empty sections
1634 1301    ld crashes with '-z ignore' due to a null data descriptor
1635 1626    libld may accidentally return success while failing
1636 2413    %ymm* need to be preserved on way through PLT
1637 3210    ld should tolerate SHT_PROGBITS for .eh_frame sections on amd64
1638 3228    Want -zassert-deflib for ld
1639 3230    ld.so.1 should check default paths for DT_DEPAUDIT
1640 3260    linker is insufficiently careful with strtok
1641 3261    linker should ignore unknown hardware capabilities
1642 3265    link-editor builds bogus .eh_frame_hdr on ia32
1643 3453    GNU comdat redirection does exactly the wrong thing
```



```
1644 3439    discarded sections shouldn't end up on output lists
1645 3436    relocatable objects also need sloppy relocation
1646 3451    archive libraries with no symbols shouldn't require a string table
1647 3616    SHF_GROUP sections should not be discarded via other COMDAT mechanisms
1648 3709    need sloppy relocation for GNU .debug_macro
1649 3722    link-editor is over restrictive of R_AMD64_32 addends
1650 3926    multiple extern map file definitions corrupt symbol table entry
1651 3999    libld extended section handling is broken
1652 4003    dldump() can't deal with extended sections
1653 4227    ld --library-path is translated to -l-path, not -L
1654 4270    ld(1) argument error reporting is still pretty bad
1655 4383    libelf can't write extended sections when ELF_F_LAYOUT
1656 4959    completely discarded merged string sections will corrupt output objects
1657 4996    rtld _init race leads to incorrect symbol values
1658 5688    ELF tools need to be more careful with dwarf data
1659 6098    ld(1) should not require symbols which identify group sections be global
1660 6252    ld should merge function/data-sections in the same manner as GNU ld
1661 7323    ld(1) -zignore can erroneously discard init and fini arrays as unreferen
1662 7594    ld -zaslr should accept Solaris-compatible values
1663 8616    ld has trouble parsing -z options specified with -Wl
1664 10267   ld and GCC disagree about i386 local dynamic TLS
1665 10366   ld(1) should support GNU-style linker sets
1666 #endif /* ! codereview */
```

```
**********************************************************
    2618 Mon Feb 11 00:23:21 2019
new/usr/src/pkg/manifests/system-test-elftest.mf
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

#
# Copyright 2018, Richard Lowe.
#

set name=pkg.fmri value=pkg:/system/test/elftest@$(PKGVERS)
set name=pkg.description value="ELF Unit Tests"
set name=pkg.summary value="ELF Test Suite"
set name=info.classification \
    value=org.opensolaris.category.2008:Development/System
set name=variant.arch value=$(ARCH)
dir path=opt/elf-tests
dir path=opt/elf-tests/bin
dir path=opt/elf-tests/runfiles
dir path=opt/elf-tests/tests
dir path=opt/elf-tests/tests/assert-deflib
dir path=opt/elf-tests/tests/linker-sets
dir path=opt/elf-tests/tests/tls
dir path=opt/elf-tests/tests/tls/x64
dir path=opt/elf-tests/tests/tls/x64/ie
dir path=opt/elf-tests/tests/tls/x86
dir path=opt/elf-tests/tests/tls/x86/ld
file path=opt/elf-tests/bin/elftest mode=0555
file path=opt/elf-tests/runfiles/default.run mode=0444
file path=opt/elf-tests/tests/assert-deflib/link.c mode=0444
file path=opt/elf-tests/tests/assert-deflib/test-deflib mode=0555
file path=opt/elf-tests/tests/linker-sets/in-use-check mode=0555
file path=opt/elf-tests/tests/linker-sets/simple mode=0555
file path=opt/elf-tests/tests/linker-sets/simple-src.c mode=0444
file path=opt/elf-tests/tests/linker-sets/simple.out mode=0444
file path=opt/elf-tests/tests/tls/x64/ie/Makefile.test mode=0444
file path=opt/elf-tests/tests/tls/x64/ie/style1-func-with-r12.s mode=0444
file path=opt/elf-tests/tests/tls/x64/ie/style1-func-with-r13.s mode=0444
file path=opt/elf-tests/tests/tls/x64/ie/style1-func.s mode=0444
file path=opt/elf-tests/tests/tls/x64/ie/style1-main.s mode=0444
file path=opt/elf-tests/tests/tls/x64/ie/style2-with-badness.s mode=0444
file path=opt/elf-tests/tests/tls/x64/ie/style2-with-r12.s mode=0444
file path=opt/elf-tests/tests/tls/x64/ie/style2-with-r13.s mode=0444
file path=opt/elf-tests/tests/tls/x64/ie/style2.s mode=0444
file path=opt/elf-tests/tests/tls/x64/ie/x64-ie-test mode=0555
file path=opt/elf-tests/tests/tls/x86/ld/Makefile.test mode=0444
file path=opt/elf-tests/tests/tls/x86/ld/half-ldm.s mode=0444
file path=opt/elf-tests/tests/tls/x86/ld/x86-ld-test mode=0555
license lic_CDDL license=lic_CDDL
depend fmri=developer/linker type=require
depend fmri=developer/object-file type=require
depend fmri=system/test/testrunner type=require
#endif /* ! codereview */
```

```
**********************************************************
     687 Mon Feb 11 00:23:21 2019
new/usr/src/test/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

  12 #
  13 # Copyright (c) 2012 by Delphix. All rights reserved.
  14 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
  15 #

  17 .PARALLEL: $(SUBDIRS)

  19 SUBDIRS = \
  20         crypto-tests \
  21         elf-tests \
  22         libc-tests \
  23         os-tests \
  24         smbclient-tests \
  25         test-runner \
  26         util-tests \
  27         zfs-tests
  19 SUBDIRS = libc-tests crypto-tests os-tests test-runner util-tests zfs-tests \
  20         smbclient-tests

  29 include Makefile.com
```

```
**********************************************************
     559 Mon Feb 11 00:23:21 2019
new/usr/src/test/elf-tests/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

  12 #
  13 # Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
  14 #

  16 .PARALLEL: $(SUBDIRS)

  18 SUBDIRS = cmd doc runfiles tests

  20 include $(SRC)/test/Makefile.com
  21 #endif /* ! codereview */
```

****
544 Mon Feb 11 00:23:22 2019
new/usr/src/test/elf-tests/cmd/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
****

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

#
# Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
#

.PARALLEL: $(SUBDIRS)

SUBDIRS = scripts

include $(SRC)/test/Makefile.com
#endif /* ! codereview */
```

```
**********************************************************
     852 Mon Feb 11 00:23:22 2019
new/usr/src/test/elf-tests/cmd/scripts/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
#

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

ROOTOPTPKG = $(ROOT)/opt/elf-tests
ROOTBIN = $(ROOTOPTPKG)/bin

PROGS = elftest

CMDS = $(PROGS:%=$(ROOTBIN)/%)
$(CMDS) := FILEMODE = 0555

all lint clean clobber:

install: $(CMDS)

$(CMDS): $(ROOTBIN)

$(ROOTBIN):
        $(INS.dir)

$(ROOTBIN)/%: %.ksh
        $(INS.rename)
#endif /* ! codereview */
```

```
**********************************************************
     990 Mon Feb 11 00:23:22 2019
new/usr/src/test/elf-tests/cmd/scripts/elftest.ksh
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

```
#!/usr/bin/ksh

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

#
# Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
#

export ELF_TESTS="/opt/elf-tests"
runner="/opt/test-runner/bin/run"

function fail
{
        echo $1
        exit ${2:-1}
}

function find_runfile
{
        typeset distro=default

        [[ -n $distro ]] && echo $ELF_TESTS/runfiles/$distro.run
}

while getopts c: c; do
        case $c in
        'c')
                runfile=$OPTARG
                [[ -f $runfile ]] || fail "Cannot read file: $runfile"
                ;;
        esac
done
shift $((OPTIND - 1))

[[ -z $runfile ]] && runfile=$(find_runfile)
[[ -z $runfile ]] && fail "Couldn't determine distro"

$runner -c $runfile

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

```
**********************************************************
    2003 Mon Feb 11 00:23:22 2019
new/usr/src/test/elf-tests/doc/README
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
#

ELF Software Generation Utilities Unit Test Suite README

1. Building and installing the ELF/SGS Unit Test Suite
2. Running the ELF/SGS Unit Test Suite
3. Test results

--------------------------------------------------------------------------------

1. Building and installing the ELF/SGS Unit Test Suite

The ELF/SGS Unit Test Suite runs under the testrunner framework (which can be
installed as pkg:/system/test/testrunner). To build both the ELF/SGS Unit Test S
and the testrunner without running a full nightly:

        build_machine$ bldenv [-d] <your_env_file>
        build_machine$ cd $SRC/test
        build_machine$ dmake install
        build_machine$ cd $SRC/pkg
        build_machine$ dmake install

Then set the publisher on the test machine to point to your repository and
install the ELF/SGS Unit Test Suite.

        test_machine# pkg install pkg:/system/test/elftest

Note, the framework will be installed automatically, as the ELF/SGS Unit Test Su
depends on it.

2. Running the ELF/SGS Unit Test Suite

The pre-requisites for running the ELF/SGS Unit Test Suite are:
    None

Once the pre-requisites are satisfied, simply run the elftest script:

        test_machine$ /opt/elf-tests/bin/elftest

3. Test results

While the ELF/SGS Unit Test Suite is running, one informational line is printed
the end of each test, and a results summary is printed at the end of the run.
The results summary includes the location of the complete logs, which is of the
form /var/tmp/test_results/<ISO 8601 date>.
```



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

****
908 Mon Feb 11 00:23:23 2019
new/usr/src/test/elf-tests/runfiles/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
****

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

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
#

include $(SRC)/Makefile.master

SRCS = default.run

ROOTOPTPKG = $(ROOT)/opt/elf-tests
RUNFILES = $(ROOTOPTPKG)/runfiles

CMDS = $(SRCS:%=$(RUNFILES)/%)
$(CMDS) := FILEMODE = 0444

all: $(SRCS)

install: $(CMDS)

clean lint clobber:

$(CMDS): $(RUNFILES) $(SRCS)

$(RUNFILES):
        $(INS.dir)

$(RUNFILES)/%: %
        $(INS.file)
#endif /* ! codereview */
```

```
**********************************************************
     815 Mon Feb 11 00:23:23 2019
new/usr/src/test/elf-tests/runfiles/default.run
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

# Copyright 2018, Richard Lowe.

[DEFAULT]
pre =
verbose = False
quiet = False
timeout = 60
post =
outputdir = /var/tmp/test_results

[/opt/elf-tests/tests/linker-sets]
tests = ['simple', 'in-use-check']

[/opt/elf-tests/tests/assert-deflib]
tests = ['test-deflib']


[/opt/elf-tests/tests/tls/x64/ie]
arch = i86pc
tests = ['x64-ie-test']

[/opt/elf-tests/tests/tls/x86/ld]
arch = i86pc
tests = ['x86-ld-test']
#endif /* ! codereview */
```

```
**********************************************************
     582 Mon Feb 11 00:23:23 2019
new/usr/src/test/elf-tests/tests/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

#
# Copyright (c) 2012, 2016 by Delphix. All rights reserved.
# Copyright 2018 Joyent, Inc.
#

SUBDIRS =               \
        assert-deflib   \
        linker-sets     \
        tls

include $(SRC)/test/Makefile.com
#endif /* ! codereview */
```

****
940 Mon Feb 11 00:23:23 2019
new/usr/src/test/elf-tests/tests/assert-deflib/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
****

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

# Copyright 2018, Richard Lowe.

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/test/Makefile.com

PROG =  test-deflib

DATAFILES =     link.c

ROOTOPTPKG = $(ROOT)/opt/elf-tests
TESTDIR = $(ROOTOPTPKG)/tests/assert-deflib

CMDS = $(PROG:%=$(TESTDIR)/%)
$(CMDS) := FILEMODE = 0555


DATA = $(DATAFILES:%=$(TESTDIR)/%)
$(DATA) := FILEMODE = 0444

all: $(PROG)

install: all $(CMDS) $(DATA)

lint:

clobber: clean
        -$(RM) $(PROG)

clean:
        -$(RM) $(CLEANFILES)

$(CMDS): $(TESTDIR) $(PROG)

$(TESTDIR):
        $(INS.dir)

$(TESTDIR)/%: %
        $(INS.file)
#endif /* ! codereview */
```

```
**********************************************************
    3870 Mon Feb 11 00:23:24 2019
new/usr/src/test/elf-tests/tests/assert-deflib/test-deflib.sh
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

#
# Copyright (c) 2012, Joyent, Inc.
#

#
# This test validates that the -zassert-deflib option of ld(1) works correctly.
# It requires that some cc is in your path and that you have passed in the path
# to the proto area with the new version of libld.so.4. One thing that we have
# to do is be careful with using LD_LIBRARY_PATH. Setting LD_LIBRARY_PATH does
# not change the default search path so we want to make sure that we use a
# different ISA (e.g. 32-bit vs 64-bit) from the binary we're generating.
#
unalias -a

if [[ -z $ELF_TESTS ]]; then
    print -u2 "Don't know where the test data is rooted";
    exit 1;
fi

#endif /* ! codereview */
sh_path=
sh_lib="lib"
sh_lib64="$sh_lib/64"
sh_soname="libld.so.4"
sh_cc="gcc"
sh_cc="cc"
sh_cflags="-m32"
sh_file="${ELF_TESTS}/tests/assert-deflib/link.c"
sh_file="link.c"
sh_arg0=$(basename $0)

function fatal
{
        local msg="$*"
        [[ -z "$msg" ]] && msg="failed"
        echo "$sh_arg0: $msg" >&2
        exit 1
}
_____unchanged_portion_omitted_

sh_path=${1:-/}
sh_path=$1
[[ -z "$1" ]] && fatal "<proto root>"
validate

run "-Wl,-zassert-deflib" 0 \
    "Testing basic compilation succeeds with warnings..." \
```



```
    "failed to compile with warnings"

run "-Wl,-zassert-deflib -Wl,-zfatal-warnings" 1 \
    "Testing basic compilation fails if warning are fatal..." \
    "linking succeeeded, expected failure"

run "-Wl,-zassert-deflib=libc.so -Wl,-zfatal-warnings" 0 \
    "Testing basic exception with fatal warnings..." \
    "linking failed despite exception"

run "-Wl,-zassert-deflib=libc.so -Wl,-zfatal-warnings" 0 \
    "Testing basic exception with fatal warnings..." \
    "linking failed despite exception"


run "-Wl,-zassert-deflib=lib.so -Wl,-zfatal-warnings" 1 \
    "Testing invalid library name..." \
    "ld should not allow invalid library name"

run "-Wl,-zassert-deflib=libf -Wl,-zfatal-warnings" 1 \
    "Testing invalid library name..." \
    "ld should not allow invalid library name"

run "-Wl,-zassert-deflib=libf.s -Wl,-zfatal-warnings" 1 \
    "Testing invalid library name..." \
    "ld should not allow invalid library name"

run "-Wl,-zassert-deflib=libc.so -Wl,-zfatal-warnings -lelf" 1 \
    "Errors even if one library is under exception path..." \
    "one exception shouldn't stop another"

args="-Wl,-zassert-deflib=libc.so -Wl,-zassert-deflib=libelf.so"
args="$args -Wl,-zfatal-warnings -lelf"

run "$args" 0 \
    "Multiple exceptions work..." \
    "multiple exceptions don't work"

args="-Wl,-zassert-deflib=libc.so -Wl,-zassert-deflib=libelfe.so"
args="$args -Wl,-zfatal-warnings -lelf"

run "$args" 1 \
    "Exceptions only catch the specific library" \
    "exceptions caught the wrong library"

args="-Wl,-zassert-deflib=libc.so -Wl,-zassert-deflib=libel.so"
args="$args -Wl,-zfatal-warnings -lelf"

run "$args" 1 \
    "Exceptions only catch the specific library" \
    "exceptions caught the wrong library"

echo "Tests passed."
exit 0
```

****
967 Mon Feb 11 00:23:24 2019
new/usr/src/test/elf-tests/tests/linker-sets/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
****

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

# Copyright 2018, Richard Lowe.

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/test/Makefile.com

PROG =  simple in-use-check

DATAFILES =     simple-src.c \
                simple.out

ROOTOPTPKG = $(ROOT)/opt/elf-tests
TESTDIR = $(ROOTOPTPKG)/tests/linker-sets

CMDS = $(PROG:%=$(TESTDIR)/%)
$(CMDS) := FILEMODE = 0555


DATA = $(DATAFILES:%=$(TESTDIR)/%)
$(DATA) := FILEMODE = 0444

all: $(PROG)

install: all $(CMDS) $(DATA)

lint:

clobber: clean
        -$(RM) $(PROG)

clean:
        -$(RM) $(CLEANFILES)

$(CMDS): $(TESTDIR) $(PROG)

$(TESTDIR):
        $(INS.dir)

$(TESTDIR)/%: %
        $(INS.file)
#endif /* ! codereview */
```

****
1217 Mon Feb 11 00:23:24 2019
new/usr/src/test/elf-tests/tests/linker-sets/in-use-check.sh
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
****

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

#
# Copyright 2018, Richard Lowe.
#

# Test that a simple use of linker-sets, tat is, automatically generated start
# and end symbols for sections can be generated and used.

tmpdir=/tmp/test.$$
mkdir $tmpdir
cd $tmpdir

cleanup() {
    cd /
    rm -fr $tmpdir
}

trap 'cleanup' EXIT

cat > broken.c <<EOF
void *__start_text;

int
main()
{
        return (0);
}
EOF

# We expect any alternate linker to be in LD_ALTEXEC for us already
gcc -o broken broken.c -Wall -Wextra -Wl,-zfatal-warnings > in-use.$$.out 2>&1
if (( $? == 0 )); then
    print -u2 "use of a reserved symbol didn't fail"
    exit 1;
fi

grep -q "^ld: warning: reserved symbol '__start_text' already defined in file" i
if (( $? != 0 )); then
    print -u2 "use of a reserved symbol failed for the wrong reason"
    exit 1;
fi
#endif /* ! codereview */
```

```
**********************************************************
    3415 Mon Feb 11 00:23:25 2019
new/usr/src/test/elf-tests/tests/linker-sets/simple-src.c
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

```
  1 /* The meat of this file is a copy of the FreeBSD sys/link_set.h */
  2 /*
  3  * SPDX-License-Identifier: BSD-2-Clause-FreeBSD
  4  *
  5  * Copyright (c) 1999 John D. Polstra
  6  * Copyright (c) 1999,2001 Peter Wemm <peter@FreeBSD.org>
  7  * All rights reserved.
  8  *
  9  * Redistribution and use in source and binary forms, with or without
 10  * modification, are permitted provided that the following conditions
 11  * are met:
 12  * 1. Redistributions of source code must retain the above copyright
 13  *    notice, this list of conditions and the following disclaimer.
 14  * 2. Redistributions in binary form must reproduce the above copyright
 15  *    notice, this list of conditions and the following disclaimer in the
 16  *    documentation and/or other materials provided with the distribution.
 17  *
 18  * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS'' AND
 19  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 20  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 21  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 22  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 23  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 24  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 25  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 26  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 27  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 28  * SUCH DAMAGE.
 29  *
 30  * $FreeBSD$
 31  */

 33 #include <stdio.h>

 35 #define MAKE_SET(set, sym)                                      \
 36         __asm__(".globl __start_set_" #set);                    \
 37         __asm__(".globl __stop_set_" #set);                     \
 38         static __attribute__((section(".set_" #set), used))     \
 39         void const *__set_##set##_sym_##sym = &(sym)

 41 /*
 42  * Initialize before referring to a given linker set.
 43  */
 44 #define SET_DECLARE(set, ptype)                                 \
 45         extern  __attribute__((weak)) ptype *__start_set_ ## set;       \
 46         extern __attribute__((weak)) ptype *__stop_set_ ## set

 48 #define SET_BEGIN(set)  (&__start_set_ ## set)
 49 #define SET_LIMIT(set)  (&__stop_set_ ## set)

 51 /*
 52  * Iterate over all the elements of a set.
 53  *
 54  * Sets always contain addresses of things, and "pvar" points to words
 55  * containing those addresses.  Thus is must be declared as "type **pvar",
 56  * and the address of each set item is obtained inside the loop by "*pvar".
 57  */
 58 #define SET_FOREACH(pvar, set)                                  \
 59         for (pvar = SET_BEGIN(set); pvar < SET_LIMIT(set); pvar++)
```



```
 61 #define SET_ITEM(set, i)                                        \
 62         ((SET_BEGIN(set))[i])

 64 /*
 65  * Provide a count of the items in a set.
 66  */
 67 #define SET_COUNT(set)                                          \
 68         (SET_LIMIT(set) - SET_BEGIN(set))

 70 struct foo {
 71         char buf[128];
 72 };

 74 SET_DECLARE(foo, struct foo);

 76 struct foo a = { "foo" };
 77 struct foo b = { "bar" };
 78 struct foo c = { "baz" };

 80 MAKE_SET(foo, a);
 81 MAKE_SET(foo, b);
 82 MAKE_SET(foo, c);

 84 int
 85 main(int __attribute__((unused)) argc, char __attribute__((unused)) **argv)
 86 {
 87         struct foo **c;
 88         int i = 0;

 90         printf("Set count: %d\n", SET_COUNT(foo));


 93         printf("a: %s\n", ((struct foo *)__set_foo_sym_a)->buf);
 94         printf("b: %s\n", ((struct foo *)__set_foo_sym_b)->buf);
 95         printf("c: %s\n", ((struct foo *)__set_foo_sym_c)->buf);

 97         printf("item(foo, 0): %s\n", SET_ITEM(foo, 0)->buf);
 98         printf("item(foo, 1): %s\n", SET_ITEM(foo, 1)->buf);
 99         printf("item(foo, 2): %s\n", SET_ITEM(foo, 2)->buf);

101         SET_FOREACH(c, foo) {
102                 printf("foo[%d]: %s\n", i, (*c)->buf);
103                 i++;
104         }
105 }
106 #endif /* ! codereview */
```

****
*** 124 Mon Feb 11 00:23:25 2019***
new/usr/src/test/elf-tests/tests/linker-sets/simple.out
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
****

```
Set count: 3
a: foo
b: bar
c: baz
item(foo, 0): foo
item(foo, 1): bar
item(foo, 2): baz
foo[0]: foo
foo[1]: bar
foo[2]: baz
#endif /* ! codereview */
```

```
**********************************************************
    1397 Mon Feb 11 00:23:25 2019
new/usr/src/test/elf-tests/tests/linker-sets/simple.sh
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

 13 #
 14 # Copyright 2018, Richard Lowe.
 15 #

 17 # Test that a simple use of linker-sets, tat is, automatically generated start
 18 # and end symbols for sections can be generated and used.

 20 if [[ -z $ELF_TESTS ]]; then
 21     print -u2 "Don't know where the test data is rooted";
 22     exit 1;
 23 fi

 25 tmpdir=/tmp/test.$$
 26 mkdir $tmpdir
 27 cd $tmpdir

 29 cleanup() {
 30     cd /
 31     rm -fr $tmpdir
 32 }

 34 trap 'cleanup' EXIT

 36 # We expect any alternate linker to be in LD_ALTEXEC for us already
 37 gcc -o simple ${ELF_TESTS}/tests/linker-sets/simple-src.c -Wall -Wextra
 38 if (( $? != 0 )); then
 39     print -u2 "compilation of ${ELF_TESTS}/tests/linker-sets/simple-src.c failed
 40     exit 1;
 41 fi

 43 ./simple > simple.$$.out 2>&1

 45 if (( $? != 0 )); then
 46     print -u2 "execution of ${ELF_TESTS}/tests/linker-sets/simple-src.c failed";
 47     exit 1;
 48 fi

 50 diff -u ${ELF_TESTS}/tests/linker-sets/simple.out simple.$$.out
 51 if (( $? != 0 )); then
 52     print -u2 "${ELF_TESTS}/tests/linker-sets/simple-src.c output mismatch"
 53     exit 1;
 54 fi
 55 #endif /* ! codereview */
```

****
550 Mon Feb 11 00:23:25 2019
new/usr/src/test/elf-tests/tests/tls/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
****

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

#
# Copyright (c) 2012, 2016 by Delphix. All rights reserved.
# Copyright 2018 Joyent, Inc.
#

SUBDIRS = x64 x86

include $(SRC)/test/Makefile.com
#endif /* ! codereview */
```

****
545 Mon Feb 11 00:23:26 2019
new/usr/src/test/elf-tests/tests/tls/x64/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
****

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

#
# Copyright (c) 2012, 2016 by Delphix. All rights reserved.
# Copyright 2018 Joyent, Inc.
#

SUBDIRS = ie

include $(SRC)/test/Makefile.com
#endif /* ! codereview */
```

```
**********************************************************
    1117 Mon Feb 11 00:23:26 2019
new/usr/src/test/elf-tests/tests/tls/x64/ie/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

# Copyright 2018, Richard Lowe.

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/test/Makefile.com

PROG =  x64-ie-test

DATAFILES =                     \
        Makefile.test           \
        style1-func-with-r12.s  \
        style1-func-with-r13.s  \
        style1-func.s           \
        style1-main.s           \
        style2-with-badness.s   \
        style2-with-r12.s       \
        style2-with-r13.s       \
        style2.s

ROOTOPTPKG = $(ROOT)/opt/elf-tests
TESTDIR = $(ROOTOPTPKG)/tests/tls/x64/ie

CMDS = $(PROG:%=$(TESTDIR)/%)
$(CMDS) := FILEMODE = 0555


DATA = $(DATAFILES:%=$(TESTDIR)/%)
$(DATA) := FILEMODE = 0444

all: $(PROG)

install: all $(CMDS) $(DATA)

lint:

clobber: clean
        -$(RM) $(PROG)

clean:
        -$(RM) $(CLEANFILES)

$(CMDS): $(TESTDIR) $(PROG)

$(TESTDIR):
        $(INS.dir)

$(TESTDIR)/%: %
        $(INS.file)
#endif /* ! codereview */
```

```
**********************************************************
    2363 Mon Feb 11 00:23:26 2019
new/usr/src/test/elf-tests/tests/tls/x64/ie/Makefile.test
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD_tls transition (10267)
**********************************************************
```

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

# Copyright 2012, Richard Lowe.

CC =            gcc
include $(SRC)/Makefile.master

# We have to use GCC, and only GCC.  The best way is to ask cw(1) which GCC to u
CC_CMD =        $(ONBLD_TOOLS)/bin/$(MACH)/cw -_gcc -_compiler
CC =            $(CC_CMD:sh)
CFLAGS =        -O1 -m64

LINK.c =  $(CC) $(CFLAGS) -o $@ $^
LINK.c = env LD_ALTEXEC=$(PROTO)/usr/bin/amd64/ld $(CC) $(CFLAGS) -o $@ $^
COMPILE.c = $(CC) $(CFLAGS) -c -o $@ $^
COMPILE.s = $(CC) $(CFLAGS) -c -o $@ $^

.KEEP_STATE:

install default: all

%.o: $(ELF_TESTS)/tests/tls/x64/ie/%.c
.c.o:
        $(COMPILE.c)
%.o: $(ELF_TESTS)/tests/tls/x64/ie/%.s

.s.o:
        $(COMPILE.s)

# A basic use of TLS that uses the movq m/r --> movq i/r variant
PROGS +=        style2
STYLE2OBJS =    style2.o
style2: $(STYLE2OBJS)
        $(LINK.c)

# A copy of style2 that uses %r13 in the TLS sequence, and thus excercises the
# REX transitions of the movq mem,reg -> movq imm,reg variant.
PROGS +=        style2-with-r13
STYLE2R13OBJS = style2-with-r13.o
style2-with-r13: $(STYLE2R13OBJS)
        $(LINK.c)

# A copy of style2 that uses %r12 in the TLS sequence, so we can verify that
# it is _not_ special to this variant
PROGS +=        style2-with-r12
STYLE2R12OBJS = style2-with-r12.o
style2-with-r12: $(STYLE2R12OBJS)
        $(LINK.c)

# A copy of style2 that has a R_AMD64_GOTTPOFF relocation with a bad insn sequen
```



```
STYLE2BADNESSOBJS =     style2-with-badness.o
style2-with-badness: $(STYLE2BADNESSOBJS)
        -$(LINK.c)

# A basic use of TLS that uses the addq mem/reg --> leaq mem,reg variant
PROGS +=        style1
STYLE1OBJS =    style1-main.o style1-func.o
style1: $(STYLE1OBJS)
        $(LINK.c)

# A copy of style1-func that uses %r13 in the TLS sequence and thus excercises
# the REX transitions. of the addq mem,reg --> leaq mem,reg variant
PROGS +=        style1-with-r13
STYLE1R13OBJS = style1-main.o style1-func-with-r13.o
style1-with-r13: $(STYLE1R13OBJS)
        $(LINK.c)

# A copy of style1-func that uses %r12 to test the addq mem,reg --> addq imm,reg
PROGS +=        style1-with-r12
STYLE1R12OBJS = style1-main.o style1-func-with-r12.o
style1-with-r12: $(STYLE1R12OBJS)
        $(LINK.c)

all:    $(PROGS)

clobber clean:
        rm -f $(PROGS) $(STYLE1OBJS) $(STYLE1R13OBJS) $(STYLE1R12OBJS) \
            $(STYLE2OBJS) $(STYLE2R13OBJS) $(STYLE2R12OBJS) $(STYLE2BADNESSOBJS)

fail: style2-with-badness FRC

FRC:
```

```
*********************************************************
     842 Mon Feb 11 00:23:27 2019
new/usr/src/test/elf-tests/tests/tls/x64/ie/style1-func-with-r12.s
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
*********************************************************
______unchanged_portion_omitted_
```

```
**********************************************************
     925 Mon Feb 11 00:23:29 2019
new/usr/src/test/elf-tests/tests/tls/x64/ie/style2-with-badness.s
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
______unchanged_portion_omitted_
```

```
**********************************************************
     953 Mon Feb 11 00:23:29 2019
new/usr/src/test/elf-tests/tests/tls/x64/ie/style2-with-r12.s
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

  12 /*
  13  * Copyright 2012, Richard Lowe.
  14  */

  16         .section        .rodata.str1.1,"aMS",@progbits,1
  17 .LC0:
  18         .string "foo: %p\n"
  19         .text
  20 .globl main
  21         .type   main, @function
  22 main:
  23 .LFB0:
  24         pushq   %rbp
  25 .LCFI0:
  26         movq    %rsp, %rbp
  27 .LCFI1:
  28         movq    foo@GOTTPOFF(%rip), %r12
  29         addq    %fs:0, %r12
  30         movq    %r12, %rsi
  31         movl    $.LC0, %edi
  32         movl    $0, %eax
  33         call    printf
  34         movl    $0, %eax
  35         leave
  36         ret
  37 .LFE0:
  38         .size   main, .-main
  39 .globl foo
  40         .section        .rodata.str1.1
  41 .LC1:
  42         .string "foo"

  44 #endif /* ! codereview */
  45         .section        .tdata,"awT",@progbits
  46         .align 8
  47         .type   foo, @object
  48         .size   foo, 8
  49 foo:
  50         .quad   .LC1
```

```
**********************************************************
     952 Mon Feb 11 00:23:30 2019
new/usr/src/test/elf-tests/tests/tls/x64/ie/style2-with-r13.s
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
______unchanged_portion_omitted_
```

```
**********************************************************
     925 Mon Feb 11 00:23:30 2019
new/usr/src/test/elf-tests/tests/tls/x64/ie/style2.s
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
______unchanged_portion_omitted_
```

```
**********************************************************
    2251 Mon Feb 11 00:23:31 2019
new/usr/src/test/elf-tests/tests/tls/x64/ie/x64-ie-test.sh
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

# Copyright 2012, Richard Lowe.

function grep_test {
    name=$1
    pattern=$2

    if /usr/bin/fgrep -q "${pattern}"; then
        print -u2 "pass: $name"
    else
        print -u2 "FAIL: $name"
        exit 1
#endif /* ! codereview */
    fi
}

function dis_test {
    name=${1}
    func=${2}
    file=${3}
    pattern=${4}

    dis -F${func} ${file} | grep_test "${name}" "${pattern}"
}

if [[ -z $ELF_TESTS ]]; then
    print -u2 "Don't know where the test data is rooted";
    exit 1;
fi

make -f ${ELF_TESTS}/tests/tls/x64/ie/Makefile.test
make PROTO="${1}"

dis_test "addq-->leaq 1" func style1 \
    'func+0x10: 48 8d 92 f8 ff ff  leaq   -0x8(%rdx),%rdx'
dis_test "addq-->leaq 2" func style1 \
    'func+0x17: 48 8d b6 f0 ff ff  leaq   -0x10(%rsi),%rsi'

dis_test "addq-->leaq w/REX 1" func style1-with-r13 \
    'func+0x10: 48 8d 92 f8 ff ff  leaq   -0x8(%rdx),%rdx'
dis_test "addq-->leaq w/REX 2" func style1-with-r13 \
    'func+0x17: 4d 8d ad f0 ff ff  leaq   -0x10(%r13),%r13'

dis_test "addq-->addq for SIB 1" func style1-with-r12 \
    'func+0x10: 48 8d 92 f8 ff ff  leaq   -0x8(%rdx),%rdx'
dis_test "addq-->addq for SIB 2" func style1-with-r12 \
    'func+0x17: 49 81 c4 f0 ff ff  addq   $-0x10,%r12   <0xfffffffffffffff0>'
```



```
dis_test "movq-->movq" main style2 \
    'main+0x4:  48 c7 c6 f0 ff ff  movq   $-0x10,%rsi   <0xfffffffffffffff0>'

dis_test "movq-->movq w/REX" main style2-with-r13 \
    'main+0x4:  49 c7 c5 f0 ff ff  movq   $-0x10,%r13   <0xfffffffffffffff0>'

dis_test "movq-->movq incase of SIB" main style2-with-r12 \
    'main+0x4:  49 c7 c4 f0 ff ff  movq   $-0x10,%r12   <0xfffffffffffffff0>'

make -f ${ELF_TESTS}/tests/tls/x64/ie/Makefile.test fail 2>&1 | grep_test "bad i
make PROTO="${1}" fail 2>&1 | grep_test "bad insn sequence" \
    'ld: fatal: relocation error: R_AMD64_TPOFF32: file style2-with-badness.o: sy
```

****
545 Mon Feb 11 00:23:31 2019
new/usr/src/test/elf-tests/tests/tls/x86/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
****

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

#
# Copyright (c) 2012, 2016 by Delphix. All rights reserved.
# Copyright 2018 Joyent, Inc.
#

SUBDIRS = ld

include $(SRC)/test/Makefile.com
#endif /* ! codereview */
```

```
**********************************************************
     964 Mon Feb 11 00:23:31 2019
new/usr/src/test/elf-tests/tests/tls/x86/ld/Makefile
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

# Copyright 2018, Richard Lowe.

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/test/Makefile.com

PROG =  x86-ld-test

DATAFILES =             \
        Makefile.test   \
        half-ldm.s      \

ROOTOPTPKG = $(ROOT)/opt/elf-tests
TESTDIR = $(ROOTOPTPKG)/tests/tls/x86/ld

CMDS = $(PROG:%=$(TESTDIR)/%)
$(CMDS) := FILEMODE = 0555


DATA = $(DATAFILES:%=$(TESTDIR)/%)
$(DATA) := FILEMODE = 0444

all: $(PROG)

install: all $(CMDS) $(DATA)

lint:

clobber: clean
        -$(RM) $(PROG)

clean:
        -$(RM) $(CLEANFILES)

$(CMDS): $(TESTDIR) $(PROG)

$(TESTDIR):
        $(INS.dir)

$(TESTDIR)/%: %
        $(INS.file)
#endif /* ! codereview */
```

```
**********************************************************
    1053 Mon Feb 11 00:23:31 2019
new/usr/src/test/elf-tests/tests/tls/x86/ld/Makefile.test
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

# Copyright 2012, Richard Lowe.

CC =            gcc
CFLAGS =        -O1 -m32

LINK.c =  $(CC) $(CFLAGS) -o $@ $^
COMPILE.c = $(CC) $(CFLAGS) -c -o $@ $^
COMPILE.s = $(CC) $(CFLAGS) -c -o $@ $^

.KEEP_STATE:

install default: all

%.o: $(ELF_TESTS)/tests/tls/x86/ld/%.c
        $(COMPILE.c)
%.o: $(ELF_TESTS)/tests/tls/x86/ld/%.s
        $(COMPILE.s)

# an R_386_TLS_LDM with a regular R_386_PLT32 not a R_386_TLS_LDM_PLT
PROGS +=        half-ldm

half-ldm: half-ldm.o
        $(LINK.c)

all:    $(PROGS)

clobber clean:
        rm -f $(PROGS) $(STYLE1OBJS) $(STYLE1R13OBJS) $(STYLE1R12OBJS) \
            $(STYLE2OBJS) $(STYLE2R13OBJS) $(STYLE2R12OBJS) $(STYLE2BADNESSOBJS)

fail: style2-with-badness FRC

FRC:
#endif /* ! codereview */
```

****
1355 Mon Feb 11 00:23:32 2019
new/usr/src/test/elf-tests/tests/tls/x86/ld/half-ldm.s
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
****

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

/*
 * Copyright 2019, Richard Lowe.
 */

        .section .rodata.str1.1,"aMS",@progbits,1
.LC0:
        .string "foo: %s (%p)\n"
        .section .tdata,"awT",@progbits
        .align 4
        .type foo, @object
        .size foo,4
.local foo
foo:
        .string "foo"
        .text
.globl main
        .type main, @function
main:
        pushl %ebp
        movl %esp, %ebp
        /*
         * an R_386_TLS_LDM relocation without a following
         * followed by an R_386_PLT32 relocation, rather than an
         * R_386_TLS_LDM_PLT the call should be removed, and _not_
         * left alone unrelocated as it was prior to:
         * 10267 ld and GCC disagree about i386 local dynamic TLS
         */
        leal foo@TLSLDM(%ebx), %eax
        call ___tls_get_addr@PLT
        leal  foo@DTPOFF(%eax), %edx
        pushl %edx
        pushl %edx
        pushl $.LC0
        call printf@PLT
        movl $0x0,%eax
        leave
        ret
        .size main, .-main
#endif /* ! codereview */
```

```
**********************************************************
    1107 Mon Feb 11 00:23:32 2019
new/usr/src/test/elf-tests/tests/tls/x86/ld/x86-ld-test.sh
10366 ld(1) should support GNU-style linker sets
10367 ld(1) tests should be a real test suite
10368 want an ld(1) regression test for i386 LD tls transition (10267)
**********************************************************
```

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

# Copyright 2012, Richard Lowe.

function grep_test {
    name=$1
    pattern=$2

    if /usr/bin/grep -q "${pattern}"; then
        print -u2 "pass: $name"
        else
        print -u2 "FAIL: $name"
        exit 1
    fi
}

function dis_test {
    name=${1}
    func=${2}
    file=${3}
    pattern=${4}

    dis -F${func} ${file} | grep_test "${name}" "${pattern}"
}

if [[ -z $ELF_TESTS ]]; then
    print -u2 "Don't know where the test data is rooted";
    exit 1;
fi

make -f ${ELF_TESTS}/tests/tls/x86/ld/Makefile.test

dis_test "call-->nop" main half-ldm \
    'main\+0x9:  0f 1f 44 00 00     nopl   0x0(%eax,%eax)'

./half-ldm | grep_test 'half-ldm execution' \
    '^foo: foo ([a-f0-9]*)$'
#endif /* ! codereview */
```