```
**********************************************************
   16037 Sun Feb 24 19:19:04 2019
new/usr/src/cmd/file/elf_read.c
file: support DT_SUNW_KMOD usefully
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
  22 /*        All Rights Reserved   */


  25 /*      Copyright (c) 1987, 1988 Microsoft Corporation  */
  26 /*        All Rights Reserved   */

  28 /*
  29  * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */

  33 /*
  34  * ELF files can exceed 2GB in size. A standard 32-bit program
  35  * like 'file' cannot read past 2GB, and will be unable to see
  36  * the ELF section headers that typically are at the end of the
  37  * object. The simplest solution to this problem would be to make
  38  * the 'file' command a 64-bit application. However, as a matter of
  39  * policy, we do not want to require this. A simple command like
  40  * 'file' should not carry such a requirement, especially as we
  41  * support 32-bit only hardware.
  42  *
  43  * An alternative solution is to build this code as 32-bit
  44  * large file aware. The usual way to do this is to define a pair
  45  * of preprocessor definitions:
  46  *
  47  *      _LARGEFILE64_SOURCE
  48  *              Map standard I/O routines to their largefile aware versions.
  49  *
  50  *      _FILE_OFFSET_BITS=64
  51  *              Map off_t to off64_t
  52  *
  53  * The problem with this solution is that libelf is not large file capable,
  54  * and the libelf header file will prevent compilation if
  55  * _FILE_OFFSET_BITS is set to 64.
  56  *
  57  * So, the solution used in this code is to define _LARGEFILE64_SOURCE
  58  * to get access to the 64-bit APIs, not to define _FILE_OFFSET_BITS, and to
  59  * use our own types in place of off_t, and size_t. We read all the file
  60  * data directly using pread64(), and avoid the use of libelf for anything
  61  * other than the xlate functionality.
```



```
  62  */
  63 #define _LARGEFILE64_SOURCE
  64 #define FILE_ELF_OFF_T  off64_t
  65 #define FILE_ELF_SIZE_T uint64_t

  67 #include <ctype.h>
  68 #include <unistd.h>
  69 #include <fcntl.h>
  70 #include <stdio.h>
  71 #include <libelf.h>
  72 #include <stdlib.h>
  73 #include <limits.h>
  74 #include <locale.h>
  75 #include <string.h>
  76 #include <errno.h>
  77 #include <procfs.h>
  78 #include <sys/param.h>
  79 #include <sys/types.h>
  80 #include <sys/stat.h>
  81 #include <sys/elf.h>
  82 #include <sys/link.h>
  83 #endif /* ! codereview */
  84 #include <elfcap.h>
  85 #include "file.h"
  86 #include "elf_read.h"

  88 extern const char *File;

  90 static int get_class(void);
  91 static int get_version(void);
  92 static int get_format(void);
  93 static int process_shdr(Elf_Info *);
  94 static int process_phdr(Elf_Info *);
  95 static int file_xlatetom(Elf_Type, char *);
  96 static int xlatetom_nhdr(Elf_Nhdr *);
  97 static int get_phdr(Elf_Info *, int);
  98 static int get_shdr(Elf_Info *, int);

 100 static Elf_Ehdr EI_Ehdr;                /* Elf_Ehdr to be stored */
 101 static Elf_Word EI_Ehdr_shnum;          /* # section headers */
 102 static Elf_Word EI_Ehdr_phnum;          /* # program headers */
 103 static Elf_Word EI_Ehdr_shstrndx;       /* Index of section hdr string table */
 104 static Elf_Shdr EI_Shdr;                /* recent Elf_Shdr to be stored */
 105 static Elf_Phdr EI_Phdr;                /* recent Elf_Phdr to be stored */


 108 static int
 109 get_class(void)
 110 {
 111         return (EI_Ehdr.e_ident[EI_CLASS]);
 112 }

 114 static int
 115 get_version(void)
 116 {
 117         /* do as what libelf:_elf_config() does */
 118         return (EI_Ehdr.e_ident[EI_VERSION] ?
 119             EI_Ehdr.e_ident[EI_VERSION] : 1);
 120 }

 122 static int
 123 get_format(void)
 124 {
 125         return (EI_Ehdr.e_ident[EI_DATA]);
 126 }
```

```
 128 /*
 129  * file_xlatetom:       translate different headers from file
 130  *                      representation to memory representaion.
 131  */
 132 #define HDRSZ 512
 133 static int
 134 file_xlatetom(Elf_Type type, char *hdr)
 135 {
 136         Elf_Data src, dst;
 137         char *hbuf[HDRSZ];
 138         int version, format;

 140         version = get_version();
 141         format = get_format();

 143         /* will convert only these types */
 144         if (type != ELF_T_EHDR && type != ELF_T_PHDR &&
 145             type != ELF_T_SHDR && type != ELF_T_WORD &&
 146             type != ELF_T_CAP && type != ELF_T_DYN)
  82             type != ELF_T_CAP)
 147                 return (ELF_READ_FAIL);

 149         src.d_buf = (Elf_Void *)hdr;
 150         src.d_type = type;
 151         src.d_version = version;

 153         dst.d_buf = (Elf_Void *)&hbuf;
 154         dst.d_version = EV_CURRENT;

 156         src.d_size = elf_fsize(type, 1, version);
 157         dst.d_size = elf_fsize(type, 1, EV_CURRENT);
 158         if (elf_xlatetom(&dst, &src, format) == NULL)
 159                 return (ELF_READ_FAIL);

 161         (void) memcpy(hdr, &hbuf, dst.d_size);
 162         return (ELF_READ_OKAY);
 163 }
_____unchanged_portion_omitted_

 409 /*
 410  * process_shdr:        Read Section Headers to attempt to get HW/SW
 411  *                      capabilities by looking at the SUNW_cap
 412  *                      section and set string in Elf_Info.
 413  *                      Also look for symbol tables and debug
 414  *                      information sections. Set the "stripped" field
 415  *                      in Elf_Info with corresponding flags.
 416  */
 417 static int
 418 process_shdr(Elf_Info *EI)
 419 {
 420         int             mac;
 356         int             capn, mac;
 421         int             i, j, idx;
 358         FILE_ELF_OFF_T  cap_off;
 359         FILE_ELF_SIZE_T csize;
 422         char            *strtab;
 423         size_t          strtab_sz;
 362         Elf_Cap         Chdr;
 424         Elf_Shdr        *shdr = &EI_Shdr;


 366         csize = sizeof (Elf_Cap);
 426         mac = EI_Ehdr.e_machine;

 428         /* if there are no sections, return success anyway */
 429         if (EI_Ehdr.e_shoff == 0 && EI_Ehdr_shnum == 0)
```



```
 430                 return (ELF_READ_OKAY);

 432         /* read section names from String Section */
 433         if (get_shdr(EI, EI_Ehdr_shstrndx) == ELF_READ_FAIL)
 434                 return (ELF_READ_FAIL);

 436         if ((strtab = malloc(shdr->sh_size)) == NULL)
 437                 return (ELF_READ_FAIL);

 439         if (pread64(EI->elffd, strtab, shdr->sh_size, shdr->sh_offset)
 440             != shdr->sh_size)
 441                 return (ELF_READ_FAIL);

 443         strtab_sz = shdr->sh_size;

 445         /* read all the sections and process them */
 446         for (idx = 1, i = 0; i < EI_Ehdr_shnum; idx++, i++) {
 447                 char *shnam;

 449                 if (get_shdr(EI, i) == ELF_READ_FAIL)
 450                         return (ELF_READ_FAIL);

 452                 if (shdr->sh_type == SHT_NULL) {
 453                         idx--;
 454                         continue;
 455                 }

 398                 cap_off = shdr->sh_offset;
 457                 if (shdr->sh_type == SHT_SUNW_cap) {
 458                         char            capstr[128];
 459                         Elf_Cap         Chdr;
 460                         FILE_ELF_OFF_T  cap_off;
 461                         FILE_ELF_SIZE_T csize;
 462                         int capn;

 464                         cap_off = shdr->sh_offset;
 465                         csize = sizeof (Elf_Cap);
 466 #endif /* ! codereview */

 468                         if (shdr->sh_size == 0 || shdr->sh_entsize == 0) {
 469                                 (void) fprintf(stderr, ELF_ERR_ELFCAP1,
 470                                     File, EI->file);
 471                                 return (ELF_READ_FAIL);
 472                         }
 473                         capn = (shdr->sh_size / shdr->sh_entsize);
 474                         for (j = 0; j < capn; j++) {
 475                                 /*
 476                                  * read cap and xlate the values
 477                                  */
 478                                 if ((pread64(EI->elffd, &Chdr, csize, cap_off)
 479                                     != csize) ||
 401                                 if (pread64(EI->elffd, &Chdr, csize, cap_off)
 402                                     != csize ||
 480                                     file_xlatetom(ELF_T_CAP, (char *)&Chdr)
 481                                     == 0) {
 482                                         (void) fprintf(stderr, ELF_ERR_ELFCAP2,
 483                                             File, EI->file);
 484                                         return (ELF_READ_FAIL);
 485                                 }

 487                                 cap_off += csize;

 489                                 /*
 490                                  * Each capatibility group is terminated with
 491                                  * CA_SUNW_NULL.  Groups other than the first
 492                                  * represent symbol capabilities, and aren't
```

```
493                                  * interesting here.
494                                  */
495                                 if (Chdr.c_tag == CA_SUNW_NULL)
496                                         break;
497
498                                 (void) elfcap_tag_to_str(ELFCAP_STYLE_UC,
499                                     Chdr.c_tag, Chdr.c_un.c_val, capstr,
500                                     sizeof (capstr), ELFCAP_FMT_SNGSPACE,
501                                     mac);
502
503                                 if ((*EI->cap_str != '\0') && (*capstr != '\0'))
504                                         (void) strlcat(EI->cap_str, " ",
505                                             sizeof (EI->cap_str));
506
507                                 (void) strlcat(EI->cap_str, capstr,
508                                     sizeof (EI->cap_str));
509                         }
510                 } else if (shdr->sh_type == SHT_DYNAMIC) {
511                         Elf_Dyn dyn;
512                         FILE_ELF_SIZE_T dsize;
513                         FILE_ELF_OFF_T doff;
514                         int dynn;
515
516                         doff = shdr->sh_offset;
517                         dsize = sizeof (Elf_Dyn);
518
519                         if (shdr->sh_size == 0 || shdr->sh_entsize == 0) {
520                                 (void) fprintf(stderr, ELF_ERR_DYNAMIC1,
521                                     File, EI->file);
522                                 return (ELF_READ_FAIL);
523                         }
524
525                         dynn = (shdr->sh_size / shdr->sh_entsize);
526                         for (j = 0; j < dynn; j++) {
527                                 if (pread64(EI->elffd, &dyn, dsize, doff)
528                                     != dsize ||
529                                     file_xlatetom(ELF_T_DYN, (char *)&dyn)
530                                     == 0) {
531                                         (void) fprintf(stderr, ELF_ERR_DYNAMIC2,
532                                             File, EI->file);
533                                         return (ELF_READ_FAIL);
534                                 }
535
536                                 doff += dsize;
537
538                                 if ((dyn.d_tag == DT_SUNW_KMOD) &&
539                                     (dyn.d_un.d_val == 1)) {
540                                         EI->kmod = B_TRUE;
541                                 }
542 #endif /* ! codereview */
543                         }
544                 }
545
546                 /*
547                  * Definition time:
548                  *      - "not stripped" means that an executable file
549                  *      contains a Symbol Table (.symtab)
550                  *      - "stripped" means that an executable file
551                  *      does not contain a Symbol Table.
552                  * When strip -l or strip -x is run, it strips the
553                  * debugging information (.line section name (strip -l),
554                  * .line, .debug*, .stabs*, .dwarf* section names
555                  * and SHT_SUNW_DEBUGSTR and SHT_SUNW_DEBUG
556                  * section types (strip -x), however the Symbol
557                  * Table will still be present.
558                  * Therefore, if
```



```
559                  *      - No Symbol Table present, then report
560                  *              "stripped"
561                  *      - Symbol Table present with debugging
562                  *      information (line number or debug section names,
563                  *      or SHT_SUNW_DEBUGSTR or SHT_SUNW_DEBUG section
564                  *      types) then report:
565                  *              "not stripped"
566                  *      - Symbol Table present with no debugging
567                  *      information (line number or debug section names,
568                  *      or SHT_SUNW_DEBUGSTR or SHT_SUNW_DEBUG section
569                  *      types) then report:
570                  *              "not stripped, no debugging information
571                  *              available"
572                  */
573                 if ((EI->stripped & E_NOSTRIP) == E_NOSTRIP)
574                         continue;
575
576                 if (!(EI->stripped & E_SYMTAB) &&
577                     (shdr->sh_type == SHT_SYMTAB)) {
578                         EI->stripped |= E_SYMTAB;
579                         continue;
580                 }
581
582                 if (shdr->sh_name >= strtab_sz)
583                         shnam = NULL;
584                 else
585                         shnam = &strtab[shdr->sh_name];
586
587                 if (!(EI->stripped & E_DBGINF) &&
588                     ((shdr->sh_type == SHT_SUNW_DEBUG) ||
589                     (shdr->sh_type == SHT_SUNW_DEBUGSTR) ||
590                     (shnam != NULL && is_in_list(shnam)))) {
591                         EI->stripped |= E_DBGINF;
592                 }
593         }
594         free(strtab);
595
596         return (ELF_READ_OKAY);
597 }
```

```
**********************************************************
    3256 Sun Feb 24 19:19:05 2019
new/usr/src/cmd/file/elf_read.h
file: support DT_SUNW_KMOD usefully
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _ELF_READ_H
#define _ELF_READ_H

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

#define BUFSZ 128
typedef struct Elf_Info {
        boolean_t       dynamic;        /* dymanically linked? */
        unsigned        core_type;      /* core? what type of core? */
        unsigned        stripped;       /* symtab, debug info */
        unsigned        flags;          /* e_flags */
        unsigned        machine;        /* e_machine */
        unsigned        type;           /* e_type */
        int             elffd;          /* fd of file being processed */
        char            fname[PRFNSZ];  /* name of process that dumped core */
        char            cap_str[BUFSZ]; /* hw/sw capabilities */
        char            *file;          /* file being processed */
        boolean_t       kmod;
#endif /* ! codereview */
} Elf_Info;

/* values for Elf_Info.stripped */
#define E_DBGINF        0x01
#define E_SYMTAB        0x02
#define E_NOSTRIP       0x03

/* values for Elf_Info.core_type; */
#define EC_NOTCORE      0x0
#define EC_OLDCORE      0x1
#define EC_NEWCORE      0x2

/* elf file processing errors */
#define ELF_ERR_ELFCAP1         gettext("%s: %s zero size or zero entry ELF " \
        "section - ELF capabilities ignored\n")
#define ELF_ERR_ELFCAP2         gettext("%s: %s: can't read ELF capabilities " \
        "data - ELF capabilities ignored\n")
```



```
#define ELF_ERR_DYNAMIC1        gettext("%s: %s zero size or zero entry ELF " \
        "section - ELF dynamic tags ignored\n")
#define ELF_ERR_DYNAMIC2        gettext("%s: %s: can't read ELF dynamic " \
        "data - ELF dynamic tags ignored\n")
#endif /* ! codereview */

extern int is_in_list(char *str);

/* return status for elf_read and its helper functions */
#define ELF_READ_OKAY 1
#define ELF_READ_FAIL 0

#if defined(_ELF64)

#define Elf_Ehdr        Elf64_Ehdr
#define Elf_Shdr        Elf64_Shdr
#define Elf_Phdr        Elf64_Phdr
#define Elf_Cap         Elf64_Cap
#define Elf_Nhdr        Elf64_Nhdr
#define Elf_Word        Elf64_Word
#define Elf_Dyn         Elf64_Dyn
#endif /* ! codereview */

#define elf_read        elf_read64
#define elf_xlatetom    elf64_xlatetom
#define elf_fsize       elf64_fsize
#define get_class       get_class64
#define get_version     get_version64
#define get_format      get_format64

#else

#define Elf_Ehdr        Elf32_Ehdr
#define Elf_Shdr        Elf32_Shdr
#define Elf_Phdr        Elf32_Phdr
#define Elf_Cap         Elf32_Cap
#define Elf_Nhdr        Elf32_Nhdr
#define Elf_Word        Elf32_Word
#define Elf_Dyn         Elf32_Dyn
#endif /* ! codereview */

#define elf_read        elf_read32
#define elf_xlatetom    elf32_xlatetom
#define elf_fsize       elf32_fsize
#define get_class       get_class32
#define get_version     get_version32
#define get_format      get_format32

#endif

/* so lint can understand elf_read64 is defined */
#ifdef lint
#define elf_read64      elf_read
#endif /* lint */

#endif /* _ELF_READ_H */
```

```
**********************************************************
   44683 Sun Feb 24 19:19:05 2019
new/usr/src/cmd/file/file.c
file: support DT_SUNW_KMOD usefully
**********************************************************
______unchanged_portion_omitted_

1270 static int
1271 elf_check(char *file)
1272 {
1273         Elf_Info EInfo;
1274         int class, version, format;
1275         unsigned char ident[EI_NIDENT];

1277         (void) memset(&EInfo, 0, sizeof (Elf_Info));
1278         EInfo.file = file;

1280         /*
1281          * Verify information in file indentifier.
1282          * Return quietly if not elf; Different type of file.
1283          */
1284         if (check_ident(ident, elffd) == ELF_READ_FAIL)
1285                 return (1);

1287         /*
1288          * Read the elf headers for processing and get the
1289          * get the needed information in Elf_Info struct.
1290          */
1291         class = ident[EI_CLASS];
1292         if (class == ELFCLASS32) {
1293                 if (elf_read32(elffd, &EInfo) == ELF_READ_FAIL) {
1294                         (void) fprintf(stderr, gettext("%s: %s: can't "
1295                             "read ELF header\n"), File, file);
1296                         return (1);
1297                 }
1298         } else if (class == ELFCLASS64) {
1299                 if (elf_read64(elffd, &EInfo) == ELF_READ_FAIL) {
1300                         (void) fprintf(stderr, gettext("%s: %s: can't "
1301                             "read ELF header\n"), File, file);
1302                         return (1);
1303                 }
1304         } else {
1305                 /* something wrong */
1306                 return (1);
1307         }

1309         /* version not in ident then 1 */
1310         version = ident[EI_VERSION] ? ident[EI_VERSION] : 1;

1312         format = ident[EI_DATA];
1313         (void) printf("%s", gettext("ELF"));
1314         print_elf_class(class);
1315         print_elf_datatype(format);
1316         print_elf_type(EInfo);

1318         if (EInfo.core_type != EC_NOTCORE) {
1319                 /* Print what kind of core is this */
1320                 if (EInfo.core_type == EC_OLDCORE)
1321                         (void) printf(" %s", gettext("pre-2.6 core file"));
1322                 else
1323                         (void) printf(" %s", gettext("core file"));
1324         }

1326         /* Print machine info */
1327         print_elf_machine(EInfo.machine);
```



```
1329         /* Print Version */
1330         if (version == 1)
1331                 (void) printf(" %s %d", gettext("Version"), version);

1333         if (EInfo.kmod) {
1334                 (void) printf(", %s", gettext("kernel module"));
1335         }

1337 #endif /* ! codereview */
1338         /* Print Flags */
1339         print_elf_flags(EInfo);

1341         /* Last bit, if it is a core */
1342         if (EInfo.core_type != EC_NOTCORE) {
1343                 /* Print the program name that dumped this core */
1344                 (void) printf(gettext(", from '%s'"), EInfo.fname);
1345                 return (0);
1346         }

1348         /* Print Capabilities */
1349         if (EInfo.cap_str[0] != '\0')
1350                 (void) printf(" [%s]", EInfo.cap_str);

1352         if ((EInfo.type != ET_EXEC) && (EInfo.type != ET_DYN))
1353                 return (0);

1355         /* Print if it is dynamically linked */
1356         if (EInfo.dynamic)
1357                 (void) printf(gettext(", dynamically linked"));
1358         else
1359                 (void) printf(gettext(", statically linked"));

1361         /* Printf it it is stripped */
1362         if (EInfo.stripped & E_SYMTAB) {
1363                 (void) printf(gettext(", not stripped"));
1364                 if (!(EInfo.stripped & E_DBGINF)) {
1365                         (void) printf(gettext(
1366                             ", no debugging information available"));
1367                 }
1368         } else {
1369                 (void) printf(gettext(", stripped"));
1370         }

1372         return (0);
1373 }

1375 /*
1376  * is_rtld_config - If file is a runtime linker config file, prints
1377  * the description and returns True (1). Otherwise, silently returns
1378  * False (0).
1379  */
1380 int
1381 is_rtld_config(void)
1382 {
1383         Rtc_id *id;

1385         if ((fbsz >= sizeof (*id)) && RTC_ID_TEST(fbuf)) {
1386                 (void) printf(gettext("Runtime Linking Configuration"));
1387                 id = (Rtc_id *) fbuf;
1388                 print_elf_class(id->id_class);
1389                 print_elf_datatype(id->id_data);
1390                 print_elf_machine(id->id_machine);
1391                 (void) printf("\n");
1392                 return (1);
1393         }
```

```
1395         return (0);
1396 }

1398 /*
1399  * lookup -
1400  * Attempts to match one of the strings from a list, 'tab',
1401  * with what is in the file, starting at the current index position 'i'.
1402  * Looks past any initial whitespace and expects whitespace or other
1403  * delimiting characters to follow the matched string.
1404  * A match identifies the file as being 'assembler', 'fortran', 'c', etc.
1405  * Returns 1 for a successful match, 0 otherwise.
1406  */
1407 static int
1408 lookup(char **tab)
1409 {
1410         register char   r;
1411         register int    k, j, l;

1413         while (fbuf[i] == ' ' || fbuf[i] == '\t' || fbuf[i] == '\n')
1414                 i++;
1415         for (j = 0; tab[j] != 0; j++) {
1416                 l = 0;
1417                 for (k = i; ((r = tab[j][l++]) == fbuf[k] && r != '\0'); k++)
1418                         ;
1419                 if (r == '\0')
1420                         if (fbuf[k] == ' ' || fbuf[k] == '\n' ||
1421                             fbuf[k] == '\t' || fbuf[k] == '{' ||
1422                             fbuf[k] == '/') {
1423                                 i = k;
1424                                 return (1);
1425                         }
1426         }
1427         return (0);
1428 }

1430 /*
1431  * ccom -
1432  * Increments the current index 'i' into the file buffer 'fbuf' past any
1433  * whitespace lines and C-style comments found, starting at the current
1434  * position of 'i'.  Returns 1 as long as we don't increment i past the
1435  * size of fbuf (fbsz).  Otherwise, returns 0.
1436  */

1438 static int
1439 ccom(void)
1440 {
1441         register char   cc;
1442         int             len;

1444         while ((cc = fbuf[i]) == ' ' || cc == '\t' || cc == '\n')
1445                 if (i++ >= fbsz)
1446                         return (0);
1447         if (fbuf[i] == '/' && fbuf[i+1] == '*') {
1448                 i += 2;
1449                 while (fbuf[i] != '*' || fbuf[i+1] != '/') {
1450                         if (fbuf[i] == '\\')
1451                                 i++;
1452                         if ((len = mblen(&fbuf[i], MB_CUR_MAX)) <= 0)
1453                                 len = 1;
1454                         i += len;
1455                         if (i >= fbsz)
1456                                 return (0);
1457                 }
1458                 if ((i += 2) >= fbsz)
1459                         return (0);
1460         }
```



```
1461         if (fbuf[i] == '\n')
1462                 if (ccom() == 0)
1463                         return (0);
1464         return (1);
1465 }

1467 /*
1468  * ascom -
1469  * Increments the current index 'i' into the file buffer 'fbuf' past
1470  * consecutive assembler program comment lines starting with ASCOMCHAR,
1471  * starting at the current position of 'i'.
1472  * Returns 1 as long as we don't increment i past the
1473  * size of fbuf (fbsz).  Otherwise returns 0.
1474  */

1476 static int
1477 ascom(void)
1478 {
1479         while (fbuf[i] == ASCOMCHAR) {
1480                 i++;
1481                 while (fbuf[i++] != '\n')
1482                         if (i >= fbsz)
1483                                 return (0);
1484                 while (fbuf[i] == '\n')
1485                         if (i++ >= fbsz)
1486                                 return (0);
1487         }
1488         return (1);
1489 }

1491 /* look for "1hdddddd" where d is a digit */
1492 #endif /* ! codereview */
1493 static int
1494 sccs(void)
1495 {
1333 {                               /* look for "1hdddddd" where d is a digit */
1496         register int j;

1498         if (fbuf[0] == 1 && fbuf[1] == 'h') {
1499                 for (j = 2; j <= 6; j++) {
1500                         if (isdigit(fbuf[j]))
1501                                 continue;
1502                         else
1503                                 return (0);
1504                 }
1505         } else {
1506                 return (0);
1507         }
1508         return (1);
1509 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   53641 Sun Feb 24 19:19:06 2019
new/usr/src/cmd/sgs/dump/common/dump.c
ld: implement -ztype and rework option parsing
**********************************************************
______unchanged_portion_omitted_


1064 /*
1065  * Print dynamic linking information.  Input is an ELF
1066  * file descriptor, the SCNTAB structure, the number of
1067  * sections, and the filename.
1068  */
1069 static void
1070 dump_dynamic(Elf *elf_file, SCNTAB *p_scns, int num_scns, char *filename)
1071 {
1072 #define pdyn_Fmtptr     "%#llx"

1074         Elf_Data        *dyn_data;
1075         GElf_Dyn        p_dyn;
1076         GElf_Phdr       p_phdr;
1077         GElf_Ehdr       p_ehdr;
1078         int             index = 1;
1079         int             lib_scns = num_scns;
1080         SCNTAB          *l_scns = p_scns;
1081         int             header_num = 0;
1082         const char      *str;

1084         (void) gelf_getehdr(elf_file, &p_ehdr);

1086         if (!p_flag)
1087                 (void) printf("\n  **** DYNAMIC SECTION INFORMATION ****\n");

1089         for (; num_scns > 0; num_scns--, p_scns++) {
1090                 GElf_Word       link;
1091                 int             ii;


1094                 if (p_scns->p_shdr.sh_type != SHT_DYNAMIC)
1095                         continue;

1097                 if (!p_flag) {
1098                         (void) printf("%s:\n", p_scns->scn_name);
1099                         (void) printf("[INDEX]\tTag         Value\n");
1100                 }

1102                 if ((dyn_data = elf_getdata(p_scns->p_sd, NULL)) == 0) {
1103                         (void) fprintf(stderr, "%s: %s: no data in "
1104                             "%s section\n", prog_name, filename,
1105                             p_scns->scn_name);
1106                         return;
1107                 }

1109                 link = p_scns->p_shdr.sh_link;
1110                 ii = 0;

1112                 (void) gelf_getdyn(dyn_data, ii++, &p_dyn);
1113                 while (p_dyn.d_tag != DT_NULL) {
1114                         union {
1115                                 Conv_inv_buf_t          inv;
1116                                 Conv_dyn_flag_buf_t     dyn_flag;
1117                                 Conv_dyn_flag1_buf_t    dyn_flag1;
1118                                 Conv_dyn_feature1_buf_t dyn_feature1;
1119                                 Conv_dyn_posflag1_buf_t dyn_posflag1;
1120                         } conv_buf;
```



```
1122                         (void) printf("[%d]\t%-15.15s ", index++,
1123                             conv_dyn_tag(p_dyn.d_tag,
1124                             p_ehdr.e_ident[EI_OSABI], p_ehdr.e_machine,
1125                             DUMP_CONVFMT, &conv_buf.inv));

1127                         /*
1128                          * It would be nice to use a table driven loop
1129                          * here, but the address space is too sparse
1130                          * and irregular. A switch is simple and robust.
1131                          */
1132                         switch (p_dyn.d_tag) {
1133                         /*
1134                          * Items with an address value
1135                          */
1136                         case DT_PLTGOT:
1137                         case DT_HASH:
1138                         case DT_STRTAB:
1139                         case DT_RELA:
1140                         case DT_SYMTAB:
1141                         case DT_INIT:
1142                         case DT_FINI:
1143                         case DT_REL:
1144                         case DT_DEBUG:
1145                         case DT_TEXTREL:
1146                         case DT_JMPREL:
1147                         case DT_INIT_ARRAY:
1148                         case DT_FINI_ARRAY:
1149                         case DT_INIT_ARRAYSZ:
1150                         case DT_FINI_ARRAYSZ:
1151                         case DT_PREINIT_ARRAY:
1152                         case DT_PREINIT_ARRAYSZ:
1153                         case DT_SUNW_RTLDINF:
1154                         case DT_SUNW_CAP:
1155                         case DT_SUNW_CAPINFO:
1156                         case DT_SUNW_CAPCHAIN:
1157                         case DT_SUNW_SYMTAB:
1158                         case DT_SUNW_SYMSORT:
1159                         case DT_SUNW_TLSSORT:
1160                         case DT_PLTPAD:
1161                         case DT_MOVETAB:
1162                         case DT_SYMINFO:
1163                         case DT_RELACOUNT:
1164                         case DT_RELCOUNT:
1165                         case DT_VERSYM:
1166                         case DT_VERDEF:
1167                         case DT_VERDEFNUM:
1168                         case DT_VERNEED:
1169                                 (void) printf(pdyn_Fmtptr,
1170                                     EC_ADDR(p_dyn.d_un.d_ptr));
1171                                 break;

1173                         /*
1174                          * Items with a string value
1175                          */
1176                         case DT_NEEDED:
1177                         case DT_SONAME:
1178                         case DT_RPATH:
1179                         case DT_RUNPATH:
1180                         case DT_SUNW_AUXILIARY:
1181                         case DT_SUNW_FILTER:
1182                         case DT_CONFIG:
1183                         case DT_DEPAUDIT:
1184                         case DT_AUDIT:
1185                         case DT_AUXILIARY:
1186                         case DT_USED:
1187                         case DT_FILTER:
```

```
1188                                if (v_flag) {   /* Look up the string */
1189                                        str = (char *)elf_strptr(elf_file, link,
1190                                            p_dyn.d_un.d_ptr);
1191                                        if (!(str && *str))
1192                                                str = (char *)UNKNOWN;
1193                                        (void) printf("%s", str);
1194                                } else {        /* Show the address */
1195                                        (void) printf(pdyn_Fmtptr,
1196                                            EC_ADDR(p_dyn.d_un.d_ptr));
1197                                }
1198                                break;

1200                        /*
1201                         * Items with a literal value
1202                         */
1203                        case DT_PLTRELSZ:
1204                        case DT_RELASZ:
1205                        case DT_RELAENT:
1206                        case DT_STRSZ:
1207                        case DT_SYMENT:
1208                        case DT_RELSZ:
1209                        case DT_RELENT:
1210                        case DT_PLTREL:
1211                        case DT_BIND_NOW:
1212                        case DT_CHECKSUM:
1213                        case DT_PLTPADSZ:
1214                        case DT_MOVEENT:
1215                        case DT_MOVESZ:
1216                        case DT_SYMINSZ:
1217                        case DT_SYMINENT:
1218                        case DT_VERNEEDNUM:
1219                        case DT_SPARC_REGISTER:
1220                        case DT_SUNW_SYMSZ:
1221                        case DT_SUNW_SORTENT:
1222                        case DT_SUNW_SYMSORTSZ:
1223                        case DT_SUNW_TLSSORTSZ:
1224                        case DT_SUNW_STRPAD:
1225                        case DT_SUNW_CAPCHAINENT:
1226                        case DT_SUNW_CAPCHAINSZ:
1227                        case DT_SUNW_ASLR:
1228                        case DT_SUNW_KMOD:
1229 #endif /* ! codereview */
1230                                (void) printf(pdyn_Fmtptr,
1231                                    EC_XWORD(p_dyn.d_un.d_val));
1232                                break;

1234                        /*
1235                         * Integer items that are bitmasks, or which
1236                         * can be otherwise formatted in symbolic form.
1237                         */
1238                        case DT_FLAGS:
1239                        case DT_FEATURE_1:
1240                        case DT_POSFLAG_1:
1241                        case DT_FLAGS_1:
1242                        case DT_SUNW_LDMACH:
1243                                str = NULL;
1244                                if (v_flag) {
1245                                        switch (p_dyn.d_tag) {
1246                                        case DT_FLAGS:
1247                                                str = conv_dyn_flag(
1248                                                    p_dyn.d_un.d_val,
1249                                                    DUMP_CONVFMT,
1250                                                    &conv_buf.dyn_flag);
1251                                                break;
1252                                        case DT_FEATURE_1:
1253                                                str = conv_dyn_feature1(
```



```
1254                                                    p_dyn.d_un.d_val,
1255                                                    DUMP_CONVFMT,
1256                                                    &conv_buf.dyn_feature1);
1257                                                break;
1258                                        case DT_POSFLAG_1:
1259                                                str = conv_dyn_posflag1(
1260                                                    p_dyn.d_un.d_val,
1261                                                    DUMP_CONVFMT,
1262                                                    &conv_buf.dyn_posflag1);
1263                                                break;
1264                                        case DT_FLAGS_1:
1265                                                str = conv_dyn_flag1(
1266                                                    p_dyn.d_un.d_val, 0,
1267                                                    &conv_buf.dyn_flag1);
1268                                                break;
1269                                        case DT_SUNW_LDMACH:
1270                                                str = conv_ehdr_mach(
1271                                                    p_dyn.d_un.d_val, 0,
1272                                                    &conv_buf.inv);
1273                                                break;
1274                                        }
1275                                }
1276                                if (str) {      /* Show as string */
1277                                        (void) printf("%s", str);
1278                                } else {        /* Numeric form */
1279                                        (void) printf(pdyn_Fmtptr,
1280                                            EC_ADDR(p_dyn.d_un.d_ptr));
1281                                }
1282                                break;

1284                        /*
1285                         * Depreciated items with a literal value
1286                         */
1287                        case DT_DEPRECATED_SPARC_REGISTER:
1288                                (void) printf(pdyn_Fmtptr
1289                                    "  (deprecated value)",
1290                                    EC_XWORD(p_dyn.d_un.d_val));
1291                                break;

1293                        /* Ignored items */
1294                        case DT_SYMBOLIC:
1295                                (void) printf("(ignored)");
1296                                break;
1297                        }
1298                        (void) printf("\n");
1299                        (void) gelf_getdyn(dyn_data, ii++, &p_dyn);
1300                }
1301        }

1303        /*
1304         * Check for existence of static shared library information.
1305         */
1306        while (header_num < p_ehdr.e_phnum) {
1307                (void) gelf_getphdr(elf_file, header_num, &p_phdr);
1308                if (p_phdr.p_type == PT_SHLIB) {
1309                        while (--lib_scns > 0) {
1310                                if (strcmp(l_scns->scn_name, ".lib") == 0) {
1311                                        print_static(l_scns, filename);
1312                                }
1313                                l_scns++;
1314                        }
1315                }
1316                header_num++;
1317        }
1318 #undef  pdyn_Fmtptr
1319 }
```

```
/*
 * Print the ELF header.  Input is an ELF file descriptor
 * and the filename.  If f_flag is set, the ELF header is
 * printed to stdout, otherwise the function returns after
 * setting the pointer to the ELF header.  Any values which
 * are not known are printed in decimal.  Fields must be updated
 * as new values are added.
 */
static GElf_Ehdr *
dump_elf_header(Elf *elf_file, char *filename, GElf_Ehdr * elf_head_p)
{
        int class;
        int field;

        if (gelf_getehdr(elf_file, elf_head_p) == NULL) {
                (void) fprintf(stderr, "%s: %s: %s\n", prog_name, filename,
                    elf_errmsg(-1));
                return (NULL);
        }

        class = (int)elf_head_p->e_ident[4];

        if (class == ELFCLASS64)
                field = 21;
        else
                field = 13;

        if (!f_flag)
                return (elf_head_p);

        if (!p_flag) {
                (void) printf("\n                    **** ELF HEADER ****\n");
                (void) printf("%-*s%-11s%-*sMachine     Version\n",
                    field, "Class", "Data", field, "Type");
                (void) printf("%-*s%-11s%-*sFlags       Ehsize\n",
                    field, "Entry", "Phoff", field, "Shoff");
                (void) printf("%-*s%-11s%-*sShnum       Shstrndx\n\n",
                    field, "Phentsize", "Phnum", field, "Shentsz");
        }

        if (!v_flag) {
                (void) printf("%-*d%-11d%-*d%-12d%d\n",
                    field, elf_head_p->e_ident[4], elf_head_p->e_ident[5],
                    field, (int)elf_head_p->e_type, (int)elf_head_p->e_machine,
                    elf_head_p->e_version);
        } else {
                Conv_inv_buf_t  inv_buf;

                (void) printf("%-*s", field,
                    conv_ehdr_class(class, DUMP_CONVFMT, &inv_buf));
                (void) printf("%-11s",
                    conv_ehdr_data(elf_head_p->e_ident[5], DUMP_CONVFMT,
                    &inv_buf));
                (void) printf("%-*s", field,
                    conv_ehdr_type(elf_head_p->e_ident[EI_OSABI],
                    elf_head_p->e_type, DUMP_CONVFMT, &inv_buf));
                (void) printf("%-12s",
                    conv_ehdr_mach(elf_head_p->e_machine, DUMP_CONVFMT,
                    &inv_buf));
                (void) printf("%s\n",
                    conv_ehdr_vers(elf_head_p->e_version, DUMP_CONVFMT,
                    &inv_buf));
        }
        (void) printf("%-#*llx%-#11llx%-#*llx%-#12x%#x\n",
            field, EC_ADDR(elf_head_p->e_entry), EC_OFF(elf_head_p->e_phoff),
```



```
            field, EC_OFF(elf_head_p->e_shoff), EC_WORD(elf_head_p->e_flags),
            EC_WORD(elf_head_p->e_ehsize));
        if (!v_flag || (elf_head_p->e_shstrndx != SHN_XINDEX)) {
                (void) printf("%-#*x%-11u%-#*x%-12u%u\n",
                    field, EC_WORD(elf_head_p->e_phentsize),
                    EC_WORD(elf_head_p->e_phnum),
                    field, EC_WORD(elf_head_p->e_shentsize),
                    EC_WORD(elf_head_p->e_shnum),
                    EC_WORD(elf_head_p->e_shstrndx));
        } else {
                (void) printf("%-#*x%-11u%-#*x%-12uXINDEX\n",
                    field, EC_WORD(elf_head_p->e_phentsize),
                    EC_WORD(elf_head_p->e_phnum),
                    field, EC_WORD(elf_head_p->e_shentsize),
                    EC_WORD(elf_head_p->e_shnum));
        }
        if ((elf_head_p->e_shnum == 0) && (elf_head_p->e_shoff > 0)) {
                Elf_Scn         *scn;
                GElf_Shdr       shdr0;
                int             field;

                if (gelf_getclass(elf_file) == ELFCLASS64)
                        field = 21;
                else
                        field = 13;
                if (!p_flag) {
                        (void) printf("\n          **** SECTION HEADER[0] "
                            "{Elf Extensions} ****\n");
                        (void) printf(
                            "[No]\tType\tFlags\t%-*s %-*s%-*s%sName\n",
                            field, "Addr", field, "Offset", field,
                            "Size(shnum)",
                            /* compatibility:  tab for elf32 */
                            (field == 13) ? "\t" : "  ");
                        (void) printf("\tLn(strndx) Info\t%-*s Entsize\n",
                            field, "Adralgn");
                }
                if ((scn = elf_getscn(elf_file, 0)) == NULL) {
                        (void) fprintf(stderr,
                            "%s: %s: elf_getscn failed: %s\n",
                            prog_name, filename, elf_errmsg(-1));
                        return (NULL);
                }
                if (gelf_getshdr(scn, &shdr0) == 0) {
                        (void) fprintf(stderr,
                            "%s: %s: gelf_getshdr: %s\n",
                            prog_name, filename, elf_errmsg(-1));
                        return (NULL);
                }
                (void) printf("[0]\t%u\t%llu\t", EC_WORD(shdr0.sh_type),
                    EC_XWORD(shdr0.sh_flags));

                (void) printf("%-#*llx %-#*llx%-*llu%s%-*u\n",
                    field, EC_ADDR(shdr0.sh_addr),
                    field, EC_OFF(shdr0.sh_offset),
                    field, EC_XWORD(shdr0.sh_size),
                    /* compatibility:  tab for elf32 */
                    ((field == 13) ? "\t" : "  "),
                    field, EC_WORD(shdr0.sh_name));

                (void) printf("\t%u\t%u\t%-#*llx %-#*llx\n",
                    EC_WORD(shdr0.sh_link),
                    EC_WORD(shdr0.sh_info),
                    field, EC_XWORD(shdr0.sh_addralign),
                    field, EC_XWORD(shdr0.sh_entsize));
        }
```

```
1452         (void) printf("\n");

1454         return (elf_head_p);
1455 }

1457 /*
1458  * Print section contents.  Input is an ELF file descriptor,
1459  * the ELF header, the SCNTAB structure,
1460  * the number of symbols, and the filename.
1461  * The number of sections,
1462  * and the offset into the SCNTAB structure will be
1463  * set in dump_section if d_flag or n_flag are set.
1464  * If v_flag is set, sections which can be interpreted will
1465  * be interpreted, otherwise raw data will be output in hexidecimal.
1466  */
1467 static void
1468 print_section(Elf *elf_file,
1469     GElf_Ehdr *p_ehdr, SCNTAB *p, int num_scns, char *filename)
1470 {
1471         unsigned char    *p_sec;
1472         int     i;
1473         size_t  size;

1475         for (i = 0; i < num_scns; i++, p++) {
1476                 GElf_Shdr shdr;

1478                 size = 0;
1479                 if (s_flag && !v_flag)
1480                         p_sec = (unsigned char *)get_rawscn(p->p_sd, &size);
1481                 else
1482                         p_sec = (unsigned char *)get_scndata(p->p_sd, &size);

1484                 if ((gelf_getshdr(p->p_sd, &shdr) != NULL) &&
1485                     (shdr.sh_type == SHT_NOBITS)) {
1486                         continue;
1487                 }
1488                 if (s_flag && !v_flag) {
1489                         (void) printf("\n%s:\n", p->scn_name);
1490                         print_rawdata(p_sec, size);
1491                         continue;
1492                 }
1493                 if (shdr.sh_type == SHT_SYMTAB) {
1494                         dump_symbol_table(elf_file, p, filename);
1495                         continue;
1496                 }
1497                 if (shdr.sh_type == SHT_DYNSYM) {
1498                         dump_symbol_table(elf_file, p, filename);
1499                         continue;
1500                 }
1501                 if (shdr.sh_type == SHT_STRTAB) {
1502                         dump_string_table(p, 1);
1503                         continue;
1504                 }
1505                 if (shdr.sh_type == SHT_RELA) {
1506                         dump_reloc_table(elf_file, p_ehdr, p, 1, filename);
1507                         continue;
1508                 }
1509                 if (shdr.sh_type == SHT_REL) {
1510                         dump_reloc_table(elf_file, p_ehdr, p, 1, filename);
1511                         continue;
1512                 }
1513                 if (shdr.sh_type == SHT_DYNAMIC) {
1514                         dump_dynamic(elf_file, p, 1, filename);
1515                         continue;
1516                 }
```



```
1518                 (void) printf("\n%s:\n", p->scn_name);
1519                 print_rawdata(p_sec, size);
1520         }
1521         (void) printf("\n");
1522 }

1524 /*
1525  * Print section contents. This function does not print the contents
1526  * of the sections but sets up the parameters and then calls
1527  * print_section to print the contents.  Calling another function to print
1528  * the contents allows both -d and -n to work correctly
1529  * simultaneously. Input is an ELF file descriptor, the ELF header,
1530  * the SCNTAB structure, the number of sections, and the filename.
1531  * Set the range of sections if d_flag, and set section name if
1532  * n_flag.
1533  */
1534 static void
1535 dump_section(Elf *elf_file,
1536     GElf_Ehdr *p_ehdr, SCNTAB *s, int num_scns, char *filename)
1537 {
1538         SCNTAB *n_range, *d_range; /* for use with -n and -d modifiers */
1539         int i;
1540         int found_it = 0;  /* for use with -n section_name */

1542         if (n_flag) {
1543                 n_range = s;

1545                 for (i = 0; i < num_scns; i++, n_range++) {
1546                         if ((strcmp(name, n_range->scn_name)) != 0)
1547                                 continue;
1548                         else {
1549                                 found_it = 1;
1550                                 print_section(elf_file, p_ehdr,
1551                                     n_range, 1, filename);
1552                         }
1553                 }

1555                 if (!found_it) {
1556                         (void) fprintf(stderr, "%s: %s: %s not found\n",
1557                             prog_name, filename, name);
1558                 }
1559         } /* end n_flag */

1561         if (d_flag) {
1562                 d_range = s;
1563                 d_num = check_range(d_low, d_hi, num_scns, filename);
1564                 if (d_num < 0)
1565                         return;
1566                 d_range += d_low - 1;

1568                 print_section(elf_file, p_ehdr, d_range, d_num, filename);
1569         }       /* end d_flag */

1571         if (!n_flag && !d_flag)
1572                 print_section(elf_file, p_ehdr, s, num_scns, filename);
1573 }

1575 /*
1576  * Print the section header table. This function does not print the contents
1577  * of the section headers but sets up the parameters and then calls
1578  * print_shdr to print the contents.  Calling another function to print
1579  * the contents allows both -d and -n to work correctly
1580  * simultaneously.  Input is the SCNTAB structure,
1581  * the number of sections from the ELF header, and the filename.
1582  * Set the range of section headers to print if d_flag, and set
1583  * name of section header to print if n_flag.
```

```
 */
static void
dump_shdr(Elf *elf_file, SCNTAB *s, int num_scns, char *filename)
{

        SCNTAB *n_range, *d_range;      /* for use with -n and -d modifiers */
        int field;
        int i;
        int found_it = 0;  /* for use with -n section_name */

        if (gelf_getclass(elf_file) == ELFCLASS64)
                field = 21;
        else
                field = 13;

        if (!p_flag) {
                (void) printf("\n          **** SECTION HEADER TABLE ****\n");
                (void) printf("[No]\tType\tFlags\t%-*s %-*s %-*s%sName\n",
                    field, "Addr", field, "Offset", field, "Size",
                    /* compatibility:  tab for elf32 */
                    (field == 13) ? "\t" : "  ");
                (void) printf("\tLink\tInfo\t%-*s Entsize\n\n",
                    field, "Adralgn");
        }

        if (n_flag) {
                n_range = s;

                for (i = 1; i <= num_scns; i++, n_range++) {
                        if ((strcmp(name, n_range->scn_name)) != 0)
                                continue;
                        else {
                                found_it = 1;
                                print_shdr(elf_file, n_range, 1, i);
                        }
                }

                if (!found_it) {
                        (void) fprintf(stderr, "%s: %s: %s not found\n",
                            prog_name, filename, name);
                }
        } /* end n_flag */

        if (d_flag) {
                d_range = s;
                d_num = check_range(d_low, d_hi, num_scns, filename);
                if (d_num < 0)
                        return;
                d_range += d_low - 1;

                print_shdr(elf_file, d_range, d_num, d_low);
        }       /* end d_flag */

        if (!n_flag && !d_flag)
                print_shdr(elf_file, s, num_scns, 1);
}

/*
 * Process all of the command line options (except
 * for -a, -g, -f, and -o).  All of the options processed
 * by this function require the presence of the section
 * header table and will not be processed if it is not present.
 * Set up a buffer containing section name, section header,
 * and section descriptor for each section in the file.  This
 * structure is used to avoid duplicate calls to libelf functions.
 * Structure members for the symbol table, the debugging information,
```



```
 * and the line number information are global.  All of the
 * rest are local.
 */
static void
dump_section_table(Elf *elf_file, GElf_Ehdr *elf_head_p, char *filename)
{

        static SCNTAB   *buffer, *p_scns;
        Elf_Scn         *scn = 0;
        char            *s_name = NULL;
        int             found = 0;
        unsigned int    num_scns;
        size_t          shstrndx;
        size_t          shnum;


        if (elf_getshdrnum(elf_file, &shnum) == -1) {
                (void) fprintf(stderr,
                    "%s: %s: elf_getshdrnum failed: %s\n",
                    prog_name, filename, elf_errmsg(-1));
                return;
        }
        if (elf_getshdrstrndx(elf_file, &shstrndx) == -1) {
                (void) fprintf(stderr,
                    "%s: %s: elf_getshdrstrndx failed: %s\n",
                    prog_name, filename, elf_errmsg(-1));
                return;
        }

        if ((buffer = calloc(shnum, sizeof (SCNTAB))) == NULL) {
                (void) fprintf(stderr, "%s: %s: cannot calloc space\n",
                    prog_name, filename);
                return;
        }
        /* LINTED */
        num_scns = (int)shnum - 1;

        p_symtab = (SCNTAB *)0;
        p_dynsym = (SCNTAB *)0;
        p_scns = buffer;
        p_head_scns = buffer;

        while ((scn = elf_nextscn(elf_file, scn)) != 0) {
                if ((gelf_getshdr(scn, &buffer->p_shdr)) == 0) {
                        (void) fprintf(stderr,
                            "%s: %s: %s\n", prog_name, filename,
                            elf_errmsg(-1));
                        return;
                }
                s_name = (char *)
                    elf_strptr(elf_file, shstrndx, buffer->p_shdr.sh_name);
                buffer->scn_name = s_name ? s_name : (char *)UNKNOWN;
                buffer->p_sd   =  scn;

                if (buffer->p_shdr.sh_type == SHT_SYMTAB) {
                        found += 1;
                        p_symtab = buffer;
                }
                if (buffer->p_shdr.sh_type == SHT_DYNSYM)
                        p_dynsym = buffer;
                buffer++;
        }

        /*
         * These functions depend upon the presence of the section header table
         * and will not be invoked in its absence
```

```
1716          */
1717         if (h_flag) {
1718                 dump_shdr(elf_file, p_scns, num_scns, filename);
1719         }
1720         if (p_symtab && (t_flag || T_flag)) {
1721                 dump_symbol_table(elf_file, p_symtab, filename);
1722         }
1723         if (c_flag) {
1724                 dump_string_table(p_scns, num_scns);
1725         }
1726         if (r_flag) {
1727                 dump_reloc_table(elf_file, elf_head_p,
1728                     p_scns, num_scns, filename);
1729         }
1730         if (L_flag) {
1731                 dump_dynamic(elf_file, p_scns, num_scns, filename);
1732         }
1733         if (s_flag) {
1734                 dump_section(elf_file, elf_head_p, p_scns,
1735                     num_scns, filename);
1736         }
1737 }

1739 /*
1740  * Load the archive string table(s) (for extended-length strings)
1741  * into an in-core table/list
1742  */
1743 static struct stab_list_s *
1744 load_arstring_table(struct stab_list_s *STabList,
1745     int fd, Elf *elf_file, Elf_Arhdr *p_ar, char *filename)
1746 {
1747         off_t here;
1748         struct stab_list_s *STL_entry, *STL_next;

1750         if (p_ar) {
1751                 STL_entry = malloc(sizeof (struct stab_list_s));
1752                 STL_entry->next    = 0;
1753                 STL_entry->strings = 0;
1754                 STL_entry->size    = 0;

1756                 if (!STabList)
1757                         STabList = STL_entry;
1758                 else {
1759                         STL_next = STabList;
1760                         while (STL_next->next != (void *)0)
1761                                 STL_next = STL_next->next;
1762                         STL_next->next = STL_entry;
1763                 }

1765                 STL_entry->size    = p_ar->ar_size;
1766                 STL_entry->strings = malloc(p_ar->ar_size);
1767                 here = elf_getbase(elf_file);
1768                 if ((lseek(fd, here, 0)) != here) {
1769                         (void) fprintf(stderr,
1770                             "%s: %s: could not lseek\n", prog_name, filename);
1771                 }

1773                 if ((read(fd, STL_entry->strings, p_ar->ar_size)) == -1) {
1774                         (void) fprintf(stderr,
1775                             "%s: %s: could not read\n", prog_name, filename);
1776                 }
1777         }
1778         return (STabList);
1779 }

1781 /*
```



```
1782  * Print the archive header for each member of an archive.
1783  * Also call ar_sym_read to print the symbols in the
1784  * archive symbol table if g_flag.  Input is a file descriptor,
1785  * an ELF file descriptor, and the filename.  Putting the call
1786  * to dump the archive symbol table in this function is more
1787  * efficient since it is necessary to examine the archive member
1788  * name in the archive header to determine which member is the
1789  * symbol table.
1790  */
1791 static void
1792 dump_ar_hdr(int fd, Elf *elf_file, char *filename)
1793 {
1794         extern int v_flag, g_flag, a_flag, p_flag;
1795         Elf_Arhdr  *p_ar;
1796         Elf *arf;
1797         Elf_Cmd cmd;
1798         int title = 0;
1799         int err = 0;

1801         char buf[DATESIZE];

1803         cmd = ELF_C_READ;
1804         while ((arf = elf_begin(fd, cmd, elf_file)) != 0) {
1805                 p_ar = elf_getarhdr(arf);
1806                 if (p_ar == NULL) {
1807                         (void) fprintf(stderr,
1808                             "%s: %s: %s\n", prog_name, filename,
1809                             elf_errmsg(-1));
1810                         continue;
1811                 }
1812                 if ((strcmp(p_ar->ar_name, "/") == 0) ||
1813                     (strcmp(p_ar->ar_name, "/SYM64/") == 0)) {
1814                         if (g_flag)
1815                                 ar_sym_read(elf_file, filename);
1816                 } else if (strcmp(p_ar->ar_name, "//") == 0) {
1817                         StringTableList = load_arstring_table(
1818                             StringTableList, fd, arf, p_ar, filename);
1819                         cmd = elf_next(arf);
1820                         (void) elf_end(arf);
1821                         continue;
1822                 } else {
1823                         if (a_flag) {
1824                                 (void) printf("%s[%s]:\n", filename,
1825                                     p_ar->ar_name);
1826                                 if (!p_flag && title == 0) {
1827                                         if (!v_flag)
1828                                                 (void) printf(
1829 "\n\n\t\t\t***ARCHIVE HEADER***"
1830 "\n     Date          Uid     Gid    Mode      Size      Member Name\n\n");
1831                                         else
1832                                                 (void) printf(
1833 "\n\n\t\t\t***ARCHIVE HEADER***"
1834 "\n     Date                    Uid    Gid   Mode     Size     Member Name\n\n");
1835                                         title = 1;
1836                                 }
1837                                 if (!v_flag) {
1838                                         (void) printf(
1839 "\t0x%.8lx  %6d  %6d  0%.6ho  0x%.8lx  %-s\n\n",
1840                                             p_ar->ar_date, (int)p_ar->ar_uid,
1841                                             (int)p_ar->ar_gid,
1842                                             (int)p_ar->ar_mode,
1843                                             p_ar->ar_size, p_ar->ar_name);
1844                                 } else {
1845                                         if ((strftime(buf, DATESIZE,
1846                                             "%b %d %H:%M:%S %Y",
1847                                             localtime(
```

```
1848                                            &(p_ar->ar_date)))) == 0) {
1849                                                (void) fprintf(stderr,
1850 "%s: %s: don't have enough space to store the date\n", prog_name, filename);
1851                                                exit(1);
1852                                        }
1853                                        (void) printf(
1854 "\t%s %6d %6d 0%.6ho 0x%.8lx %-s\n\n",
1855                                            buf, (int)p_ar->ar_uid,
1856                                            (int)p_ar->ar_gid,
1857                                            (int)p_ar->ar_mode,
1858                                            p_ar->ar_size, p_ar->ar_name);
1859                                }
1860                        }
1861                }
1862                cmd = elf_next(arf);
1863                (void) elf_end(arf);
1864        } /* end while */

1866        err = elf_errno();
1867        if (err != 0) {
1868                (void) fprintf(stderr,
1869                    "%s: %s: %s\n", prog_name, filename, elf_errmsg(err));
1870        }
1871 }

1873 /*
1874  * Process member files of an archive.  This function provides
1875  * a loop through an archive equivalent the processing of
1876  * each_file for individual object files.
1877  */
1878 static void
1879 dump_ar_files(int fd, Elf *elf_file, char *filename)
1880 {
1881        Elf_Arhdr  *p_ar;
1882        Elf *arf;
1883        Elf_Cmd cmd;
1884        Elf_Kind file_type;
1885        GElf_Ehdr elf_head;
1886        char *fullname;

1888        cmd = ELF_C_READ;
1889        while ((arf = elf_begin(fd, cmd, elf_file)) != 0) {
1890                size_t  len;

1892                p_ar = elf_getarhdr(arf);
1893                if (p_ar == NULL) {
1894                        (void) fprintf(stderr, "%s: %s: %s\n",
1895                            prog_name, filename, elf_errmsg(-1));
1896                        return;
1897                }
1898                if (p_ar->ar_name[0] == '/') {
1899                        cmd = elf_next(arf);
1900                        (void) elf_end(arf);
1901                        continue;
1902                }

1904                len = strlen(filename) + strlen(p_ar->ar_name) + 3;
1905                if ((fullname = malloc(len)) == NULL)
1906                        return;
1907                (void) snprintf(fullname, len, "%s[%s]", filename,
1908                    p_ar->ar_name);
1909                (void) printf("\n%s:\n", fullname);
1910                file_type = elf_kind(arf);
1911                if (file_type == ELF_K_ELF) {
1912                        if (dump_elf_header(arf, fullname, &elf_head) == NULL)
1913                                return;
```



```
1914                        if (o_flag)
1915                                dump_exec_header(arf,
1916                                    (unsigned)elf_head.e_phnum, fullname);
1917                        if (x_flag)
1918                                dump_section_table(arf, &elf_head, fullname);
1919                } else {
1920                        (void) fprintf(stderr, "%s: %s: invalid file type\n",
1921                            prog_name, fullname);
1922                        cmd = elf_next(arf);
1923                        (void) elf_end(arf);
1924                        continue;
1925                }

1927                cmd = elf_next(arf);
1928                (void) elf_end(arf);
1929        } /* end while */
1930 }

1932 /*
1933  * Takes a filename as input.  Test first for a valid version
1934  * of libelf.a and exit on error.  Process each valid file
1935  * or archive given as input on the command line.  Check
1936  * for file type.  If it is an archive, process the archive-
1937  * specific options first, then files within the archive.
1938  * If it is an ELF object file, process it; otherwise
1939  * warn that it is an invalid file type.
1940  * All options except the archive-specific and program
1941  * execution header are processed in the function, dump_section_table.
1942  */
1943 static void
1944 each_file(char *filename)
1945 {
1946        Elf *elf_file;
1947        GElf_Ehdr elf_head;
1948        int fd;
1949        Elf_Kind   file_type;

1951        struct stat buf;

1953        Elf_Cmd cmd;
1954        errno = 0;

1956        if (stat(filename, &buf) == -1) {
1957                int     err = errno;
1958                (void) fprintf(stderr, "%s: %s: %s", prog_name, filename,
1959                    strerror(err));
1960                return;
1961        }

1963        if ((fd = open((filename), O_RDONLY)) == -1) {
1964                (void) fprintf(stderr, "%s: %s: cannot read\n", prog_name,
1965                    filename);
1966                return;
1967        }
1968        cmd = ELF_C_READ;
1969        if ((elf_file = elf_begin(fd, cmd, (Elf *)0)) == NULL) {
1970                (void) fprintf(stderr, "%s: %s: %s\n", prog_name, filename,
1971                    elf_errmsg(-1));
1972                return;
1973        }

1975        file_type = elf_kind(elf_file);
1976        if (file_type == ELF_K_AR) {
1977                if (a_flag || g_flag) {
1978                        dump_ar_hdr(fd, elf_file, filename);
1979                        elf_file = elf_begin(fd, cmd, (Elf *)0);
```

```
1980                }
1981                if (z_flag)
1982                        dump_ar_files(fd, elf_file, filename);
1983        } else {
1984                if (file_type == ELF_K_ELF) {
1985                        (void) printf("\n%s:\n", filename);
1986                        if (dump_elf_header(elf_file, filename, &elf_head)) {
1987                                if (o_flag)
1988                                        dump_exec_header(elf_file,
1989                                            (unsigned)elf_head.e_phnum,
1990                                            filename);
1991                                if (x_flag)
1992                                        dump_section_table(elf_file,
1993                                            &elf_head, filename);
1994                        }
1995                } else {
1996                        (void) fprintf(stderr, "%s: %s: invalid file type\n",
1997                            prog_name, filename);
1998                }
1999        }
2000        (void) elf_end(elf_file);
2001        (void) close(fd);
2002 }

2004 /*
2005  * Sets up flags for command line options given and then
2006  * calls each_file() to process each file.
2007  */
2008 int
2009 main(int argc, char *argv[], char *envp[])
2010 {
2011        char *optstr = OPTSTR; /* option string used by getopt() */
2012        int optchar;

2014        /*
2015         * Check for a binary that better fits this architecture.
2016         */
2017        (void) conv_check_native(argv, envp);

2019        prog_name = argv[0];

2021        (void) setlocale(LC_ALL, "");
2022        while ((optchar = getopt(argc, argv, optstr)) != -1) {
2023                switch (optchar) {
2024                case 'a':
2025                        a_flag = 1;
2026                        x_flag = 1;
2027                        break;
2028                case 'g':
2029                        g_flag = 1;
2030                        x_flag = 1;
2031                        break;
2032                case 'v':
2033                        v_flag = 1;
2034                        break;
2035                case 'p':
2036                        p_flag = 1;
2037                        break;
2038                case 'f':
2039                        f_flag = 1;
2040                        z_flag = 1;
2041                        break;
2042                case 'o':
2043                        o_flag = 1;
2044                        z_flag = 1;
2045                        break;
```



```
2046                case 'h':
2047                        h_flag = 1;
2048                        x_flag = 1;
2049                        z_flag = 1;
2050                        break;
2051                case 's':
2052                        s_flag = 1;
2053                        x_flag = 1;
2054                        z_flag = 1;
2055                        break;
2056                case 'd':
2057                        d_flag = 1;
2058                        x_flag = 1;
2059                        z_flag = 1;
2060                        set_range(optarg, &d_low, &d_hi);
2061                        break;
2062                case 'n':
2063                        n_flag++;
2064                        x_flag = 1;
2065                        z_flag = 1;
2066                        name = optarg;
2067                        break;
2068                case 'r':
2069                        r_flag = 1;
2070                        x_flag = 1;
2071                        z_flag = 1;
2072                        break;
2073                case 't':
2074                        t_flag = 1;
2075                        x_flag = 1;
2076                        z_flag = 1;
2077                        break;
2078                case 'C':
2079                        C_flag = 1;
2080                        t_flag = 1;
2081                        x_flag = 1;
2082                        z_flag = 1;
2083                        break;
2084                case 'T':
2085                        T_flag = 1;
2086                        x_flag = 1;
2087                        z_flag = 1;
2088                        set_range(optarg, &T_low, &T_hi);
2089                        break;
2090                case 'c':
2091                        c_flag = 1;
2092                        x_flag = 1;
2093                        z_flag = 1;
2094                        break;
2095                case 'L':
2096                        L_flag = 1;
2097                        x_flag = 1;
2098                        z_flag = 1;
2099                        break;
2100                case 'V':
2101                        V_flag = 1;
2102                        (void) fprintf(stderr, "dump: %s %s\n",
2103                            (const char *)SGU_PKG,
2104                            (const char *)SGU_REL);
2105                        break;
2106                case '?':
2107                        errflag += 1;
2108                        break;
2109                default:
2110                        break;
2111                }
```

```
2112         }

2114         if (errflag || (optind >= argc) || (!z_flag && !x_flag)) {
2115                 if (!(V_flag && (argc == 2))) {
2116                         usage();
2117                         exit(269);
2118                 }
2119         }

2121         if (elf_version(EV_CURRENT) == EV_NONE) {
2122                 (void) fprintf(stderr, "%s: libelf is out of date\n",
2123                     prog_name);
2124                 exit(101);
2125         }

2127         while (optind < argc) {
2128                 each_file(argv[optind]);
2129                 optind++;
2130         }
2131         return (0);
2132 }
```

****
1735 Sun Feb 24 19:19:06 2019
new/usr/src/cmd/sgs/include/_libelf.h
ld should reject kernel modules as input
****

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

/*
 * Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
 */

#ifndef __LIBELF_H
#define __LIBELF_H

/*
 * Version of libelf.h that supplies definitions for APIs that
 * are private to the linker package. Includes the standard libelf.h
 * and then supplements it with the private additions.
 */

#include <libelf.h>
#include <gelf.h>

#ifdef  __cplusplus
extern "C" {
#endif

typedef void _elf_execfill_func_t(void *, off_t, size_t);

extern void             _elf_execfill(_elf_execfill_func_t *);
extern size_t           _elf_getnextoff(Elf *);
extern off_t            _elf_getarhdrbase(Elf *);
extern size_t           _elf_getarsymwordsize(Elf *);
extern Elf64_Off        _elf_getxoff(Elf_Data *);
extern GElf_Xword       _gelf_getdyndtflags_1(Elf *);
extern GElf_Xword       _gelf_getdynval(Elf *, GElf_Sxword);
#endif /* ! codereview */
extern int              _elf_swap_wrimage(Elf *);
extern uint_t           _elf_sys_encoding(void);

#ifdef  __cplusplus
}
#endif

#endif  /* __LIBELF_H */
```

```
**********************************************************
   66899 Sun Feb 24 19:19:07 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)
ld: implement -ztype and rework option parsing
**********************************************************
______unchanged_portion_omitted_

 412 #define FLG_OF_DYNAMIC  0x00000001      /* generate dynamic output module */
 413 #define FLG_OF_STATIC   0x00000002      /* generate static output module */
 414 #define FLG_OF_EXEC     0x00000004      /* generate an executable */
 415 #define FLG_OF_RELOBJ   0x00000008      /* generate a relocatable object */
 416 #define FLG_OF_SHAROBJ  0x00000010      /* generate a shared object */
 417 #define FLG_OF_BFLAG    0x00000020      /* do no special plt building: -b */
 418 #define FLG_OF_IGNENV   0x00000040      /* ignore LD_LIBRARY_PATH: -i */
 419 #define FLG_OF_STRIP    0x00000080      /* strip output: -s */
 420 #define FLG_OF_NOWARN   0x00000100      /* disable symbol warnings: -t */
 421 #define FLG_OF_NOUNDEF  0x00000200      /* allow no undefined symbols: -zdefs */
 422 #define FLG_OF_PURETXT  0x00000400      /* allow no text relocations: -ztext */
 423 #define FLG_OF_GENMAP   0x00000800      /* generate a memory map: -m */
 424 #define FLG_OF_DYNLIBS  0x00001000      /* dynamic input allowed: -Bdynamic */
 425 #define FLG_OF_SYMBOLIC 0x00002000      /* bind global symbols: -Bsymbolic */
 426 #define FLG_OF_ADDVERS  0x00004000      /* add version stamp: -Qy */
 427 #define FLG_OF_NOLDYNSYM 0x00008000     /* -znoldynsym set */
 428 #define FLG_OF_IS_ORDER 0x00010000      /* input section ordering within a */
 429                                         /*      segment is required */
 430 #define FLG_OF_EC_FILES 0x00020000      /* Ent_desc exist w/non-NULL ec_files */
 431 #define FLG_OF_TEXTREL  0x00040000      /* text relocations have been found */
 432 #define FLG_OF_MULDEFS  0x00080000      /* multiple symbols are allowed */
 433 #define FLG_OF_TLSPHDR  0x00100000      /* a TLS program header is required */
 434 #define FLG_OF_BLDGOT   0x00200000      /* build GOT table */
 435 #define FLG_OF_VERDEF   0x00400000      /* record version definitions */
 436 #define FLG_OF_VERNEED  0x00800000      /* record version dependencies */
 437 #define FLG_OF_NOVERSEC 0x01000000      /* don't record version sections */
 438 #define FLG_OF_KEY      0x02000000      /* file requires sort keys */
 439 #define FLG_OF_PROCRED  0x04000000      /* process any symbol reductions by */
 440                                         /*      effecting the symbol table */
 441                                         /*      output and relocations */
 442 #define FLG_OF_SYMINFO  0x08000000      /* create a syminfo section */
 443 #define FLG_OF_AUX      0x10000000      /* ofl_filter is an auxiliary filter */
 444 #define FLG_OF_FATAL    0x20000000      /* fatal error during input */
 445 #define FLG_OF_WARN     0x40000000      /* warning during input processing. */
 446 #define FLG_OF_VERBOSE  0x80000000      /* -z verbose flag set */

 448 #define FLG_OF_MAPSYMB  0x0000100000000  /* symbolic scope definition seen */
 449 #define FLG_OF_MAPGLOB  0x0000200000000  /* global scope definition seen */
 450 #define FLG_OF_COMREL   0x0000400000000  /* -z combreloc set, which enables */
 451                                          /*      DT_RELACNT tracking, */
 452 #define FLG_OF_NOCOMREL 0x0000800000000  /* -z nocombreloc set */
 453 #define FLG_OF_AUTOLCL  0x0001000000000  /* automatically reduce unspecified */
 454                                          /*      global symbols to locals */
 455 #define FLG_OF_AUTOELM  0x0002000000000  /* automatically eliminate */
 456                                          /*      unspecified global symbols */
 457 #define FLG_OF_REDLSYM  0x0004000000000  /* reduce local symbols */
 458 #define FLG_OF_OS_ORDER 0x0008000000000  /* output section ordering required */
 459 #define FLG_OF_OSABI    0x0010000000000  /* tag object as ELFOSABI_SOLARIS */
 460 #define FLG_OF_ADJOSCNT 0x0020000000000  /* adjust ofl_shdrcnt to accommodate */
 461                                          /*      discarded sections */
 462 #define FLG_OF_OTOSCAP  0x0040000000000  /* convert object capabilities to */
 463                                          /*      symbol capabilities */
 464 #define FLG_OF_PTCAP    0x0080000000000  /* PT_SUNWCAP required */
 465 #define FLG_OF_CAPSTRS  0x0100000000000  /* capability strings are required */
 466 #define FLG_OF_EHFRAME  0x0200000000000  /* output contains .eh_frame section */
 467 #define FLG_OF_FATWARN  0x0400000000000  /* make warnings fatal */
```



```
 468 #define FLG_OF_ADEFLIB  0x0800000000000  /* no libraries in default path */

 470 #define FLG_OF_KMOD     0x1000000000000 /* output is a kernel module */

 472 #endif /* ! codereview */
 473 /*
 474  * In the flags1 arena, establish any options that are applicable to archive
 475  * extraction first, and associate a mask.  These values are recorded with any
 476  * archive descriptor so that they may be reset should the archive require a
 477  * rescan to try and resolve undefined symbols.
 478  */
 479 #define FLG_OF1_ALLEXRT 0x0000000001    /* extract all members from an */
 480                                         /*      archive file */
 481 #define FLG_OF1_WEAKEXT 0x0000000002    /* allow archive extraction to */
 482                                         /*      resolve weak references */
 483 #define MSK_OF1_ARCHIVE 0x0000000003    /* archive flags mask */

 485 #define FLG_OF1_NOINTRP 0x0000000008    /* -z nointerp flag set */
 486 #define FLG_OF1_ZDIRECT 0x0000000010    /* -z direct flag set */
 487 #define FLG_OF1_NDIRECT 0x0000000020    /* no-direct bindings specified */
 488 #define FLG_OF1_DEFERRED 0x0000000040   /* deferred dependency recording */

 490 #define FLG_OF1_RELDYN  0x0000000100    /* process .dynamic in rel obj */
 491 #define FLG_OF1_NRLXREL 0x0000000200    /* -z norelaxreloc flag set */
 492 #define FLG_OF1_RLXREL  0x0000000400    /* -z relaxreloc flag set */
 493 #define FLG_OF1_IGNORE  0x0000000800    /* ignore unused dependencies */
 494 #define FLG_OF1_NOSGHND 0x0000001000    /* -z nosighandler flag set */
 495 #define FLG_OF1_TEXTOFF 0x0000002000    /* text relocations are ok */
 496 #define FLG_OF1_ABSEXEC 0x0000004000    /* -zabsexec set */
 497 #define FLG_OF1_LAZYLD  0x0000008000    /* lazy loading of objects enabled */
 498 #define FLG_OF1_GRPPRM  0x0000010000    /* dependencies are to have */
 499                                         /*      GROUPPERM enabled */

 501 #define FLG_OF1_NOPARTI 0x0000040000    /* -znopartial set */
 502 #define FLG_OF1_BSSOREL 0x0000080000    /* output relocation against bss */
 503                                         /*      section */
 504 #define FLG_OF1_TLSOREL 0x0000100000    /* output relocation against .tlsbss */
 505                                         /*      section */
 506 #define FLG_OF1_MEMORY  0x0000200000    /* produce a memory model */
 507 #define FLG_OF1_NGLBDIR 0x0000400000    /* no DT_1_DIRECT flag allowed */
 508 #define FLG_OF1_ENCDIFF 0x0000800000    /* host running linker has different */
 509                                         /*      byte order than output object */
 510 #define FLG_OF1_VADDR   0x0001000000    /* a segment defines explicit vaddr */
 511 #define FLG_OF1_EXTRACT 0x0002000000    /* archive member has been extracted */
 512 #define FLG_OF1_RESCAN  0x0004000000    /* any archives should be rescanned */
 513 #define FLG_OF1_IGNPRC  0x0008000000    /* ignore processing required */
 514 #define FLG_OF1_NCSTTAB 0x0010000000    /* -znocompstrtab set */
 515 #define FLG_OF1_DONE    0x0020000000    /* link-editor processing complete */
 516 #define FLG_OF1_NONREG  0x0040000000    /* non-regular file specified as */
 517                                         /*      the output file */
 518 #define FLG_OF1_ALNODIR 0x0080000000    /* establish NODIRECT for all */
 519                                         /*      exported interfaces. */
 520 #define FLG_OF1_OVHWCAP1 0x0100000000   /* override CA_SUNW_HW_1 capabilities */
 521 #define FLG_OF1_OVSFCAP1 0x0200000000   /* override CA_SUNW_SF_1 capabilities */
 522 #define FLG_OF1_OVHWCAP2 0x0400000000   /* override CA_SUNW_HW_2 capabilities */
 523 #define FLG_OF1_OVMACHCAP 0x0800000000  /* override CA_SUNW_MACH capability */
 524 #define FLG_OF1_OVPLATCAP 0x1000000000  /* override CA_SUNW_PLAT capability */
 525 #define FLG_OF1_OVIDCAP 0x2000000000    /* override CA_SUNW_ID capability */

 527 /*
 528  * Guidance flags. The flags with the FLG_OFG_NO_ prefix are used to suppress
 529  * messages for a given category, and use the lower 28 bits of the word,
 530  * The upper nibble is reserved for other guidance status.
 531  */
 532 #define FLG_OFG_ENABLE          0x10000000      /* -z guidance option active */
 533 #define FLG_OFG_ISSUED          0x20000000      /* -z guidance message issued */
```

```
#define FLG_OFG_NO_ALL          0x0fffffff      /* disable all guidance */
#define FLG_OFG_NO_DEFS         0x00000001      /* specify all dependencies */
#define FLG_OFG_NO_DB           0x00000002      /* use direct bindings */
#define FLG_OFG_NO_LAZY         0x00000004      /* be explicit about lazyload */
#define FLG_OFG_NO_MF           0x00000008      /* use v2 mapfile syntax */
#define FLG_OFG_NO_TEXT         0x00000010      /* verify pure text segment */
#define FLG_OFG_NO_UNUSED       0x00000020      /* remove unused dependency */
#define FLG_OFG_NO_KMOD         0x00000040      /* use -z type=kmod */
#endif /* ! codereview */

/*
 * Test to see if a guidance should be given for a given category
 * or not. _no_flag is one of the FLG_OFG_NO_xxx flags. Returns TRUE
 * if the guidance should be issued, and FALSE to remain silent.
 */
#define OFL_GUIDANCE(_ofl, _no_flag) (((_ofl)->ofl_guideflags & \
        (FLG_OFG_ENABLE | (_no_flag))) == FLG_OFG_ENABLE)

/*
 * Test to see if the output file would allow the presence of
 * a .dynsym section.
 */
#define OFL_ALLOW_DYNSYM(_ofl) (((_ofl)->ofl_flags & \
        (FLG_OF_DYNAMIC | FLG_OF_RELOBJ)) == FLG_OF_DYNAMIC)

/*
 * Test to see if the output file would allow the presence of
 * a .SUNW_ldynsym section. The requirements are that a .dynsym
 * is allowed, and -znoldynsym has not been specified. Note that
 * even if the answer is True (1), we will only generate one if there
 * are local symbols that require it.
 */
#define OFL_ALLOW_LDYNSYM(_ofl) (((_ofl)->ofl_flags & \
        (FLG_OF_DYNAMIC | FLG_OF_RELOBJ | FLG_OF_NOLDYNSYM)) == FLG_OF_DYNAMIC)

/*
 * Test to see if relocation processing should be done. This is normally
 * true, but can be disabled via the '-z noreloc' option. Note that
 * relocatable objects are still relocated even if '-z noreloc' is present.
 */
#define OFL_DO_RELOC(_ofl) (((_ofl)->ofl_flags & FLG_OF_RELOBJ) || \
        !((_ofl)->ofl_dtflags_1 & DF_1_NORELOC))

/*
 * Determine whether a static executable is being built.
 */
#define OFL_IS_STATIC_EXEC(_ofl) (((_ofl)->ofl_flags & \
        (FLG_OF_STATIC | FLG_OF_EXEC)) == (FLG_OF_STATIC | FLG_OF_EXEC))

/*
 * Determine whether a static object is being built.  This macro is used
 * to select the appropriate string table, and symbol table that other
 * sections need to reference.
 */
#define OFL_IS_STATIC_OBJ(_ofl) ((_ofl)->ofl_flags & \
        (FLG_OF_RELOBJ | FLG_OF_STATIC))

/*
 * Macros for counting symbol table entries.  These are used to size symbol
 * tables and associated sections (.syminfo, SUNW_capinfo, .hash, etc.) and
 * set required sh_info entries (the offset to the first global symbol).
 */
#define SYMTAB_LOC_CNT(_ofl)            /* local .symtab entries */     \
        (2 +                            /*    NULL and STT_FILE */      \
        (_ofl)->ofl_shdrcnt +           /*    section symbol */         \
```



```
        (_ofl)->ofl_caploclcnt +        /*    local capabilities */     \
        (_ofl)->ofl_scopecnt +          /*    scoped symbols */         \
        (_ofl)->ofl_locscnt)            /*    standard locals */
#define SYMTAB_ALL_CNT(_ofl)            /* all .symtab entries */       \
        (SYMTAB_LOC_CNT(_ofl) +         /*    .symtab locals */         \
        (_ofl)->ofl_globcnt)            /*    standard globals */

#define DYNSYM_LOC_CNT(_ofl)            /* local .dynsym entries */     \
        (1 +                            /*    NULL */                   \
        (_ofl)->ofl_dynshdrcnt +        /*    section symbols */        \
        (_ofl)->ofl_caploclcnt +        /*    local capabilities */     \
        (_ofl)->ofl_lregsymcnt)         /*    local register symbols */
#define DYNSYM_ALL_CNT(_ofl)            /* all .dynsym entries */       \
        (DYNSYM_LOC_CNT(_ofl) +         /*    .dynsym locals */         \
        (_ofl)->ofl_globcnt)            /*    standard globals */

/*
 * Define a move descriptor used within relocation structures.
 */
typedef struct {
        Move            *mr_move;
        Sym_desc        *mr_sym;
} Mv_reloc;

/*
 * Relocation (active & output) processing structure - transparent to common
 * code. There can be millions of these structures in a large link, so it
 * is important to keep it small. You should only add new items to Rel_desc
 * if they are critical, apply to most relocations, and cannot be easily
 * computed from the other information.
 *
 * Items that can be derived should be implemented as a function that accepts
 * a Rel_desc argument, and returns the desired data. ld_reloc_sym_name() is
 * an example of this.
 *
 * Lesser used relocation data is kept in an auxiliary block, Rel_aux,
 * that is only allocated as necessary. In exchange for adding one pointer
 * of overhead to Rel_desc (rel_aux), most relocations are reduced in size
 * by the size of Rel_aux. This strategy relies on the data in Rel_aux
 * being rarely needed --- otherwise it will backfire badly.
 *
 * Note that rel_raddend is primarily only of interest to RELA relocations,
 * and is set to 0 for REL. However, there is an exception: If FLG_REL_NADDEND
 * is set, then rel_raddend contains a replacement value for the implicit
 * addend found in the relocation target.
 *
 * Fields should be ordered from largest to smallest, to minimize packing
 * holes in the struct layout.
 */
struct rel_desc {
        Is_desc         *rel_isdesc;    /* input section reloc is against */
        Sym_desc        *rel_sym;       /* sym relocation is against */
        Rel_aux         *rel_aux;       /* NULL, or auxiliary data */
        Xword           rel_roffset;    /* relocation offset */
        Sxword          rel_raddend;    /* addend from input relocation */
        Word            rel_flags;      /* misc. flags for relocations */
        Word            rel_rtype;      /* relocation type */
};

/*
 * Data that would be kept in Rel_desc if the size of that structure was
 * not an issue. This auxiliary block is only allocated as needed,
 * and must only contain rarely needed items. The goal is for the vast
 * majority of Rel_desc structs to not have an auxiliary block.
 *
 * When a Rel_desc does not have an auxiliary block, a default value
```

```
 * is assumed for each auxiliary item:
 *
 * -    ra_osdesc:
 *      Output section to which relocation applies. The default
 *      value for this is the output section associated with the
 *      input section (rel_isdesc->is_osdesc), or NULL if there
 *      is no associated input section.
 *
 * -    ra_usym:
 *      If the symbol associated with a relocation is part of a weak/strong
 *      pair, then ra_usym contains the strong symbol and rel_sym the weak.
 *      Otherwise, the default value is the same value as rel_sym.
 *
 * -    ra_move:
 *      Move table data. The default value is NULL.
 *
 * -    ra_typedata:
 *      ELF_R_TYPE_DATA(info). This value applies only to a small
 *      subset of 64-bit sparc relocations, and is otherwise 0. The
 *      default value is 0.
 *
 * If any value in Rel_aux is non-default, then an auxiliary block is
 * necessary, and each field contains its actual value. If all the auxiliary
 * values are default, no Rel_aux is needed, and the RELAUX_GET_xxx()
 * macros below are able to supply the proper default.
 *
 * To set a Rel_aux value, use the ld_reloc_set_aux_XXX() functions.
 * These functions are written to avoid unnecessary auxiliary allocations,
 * and know the rules for each item.
 */
struct rel_aux {
        Os_desc         *ra_osdesc;     /* output section reloc is against */
        Sym_desc        *ra_usym;       /* strong sym if this is a weak pair */
        Mv_reloc        *ra_move;       /* move table information */
        Word            ra_typedata;    /* ELF_R_TYPE_DATA(info) */
};

/*
 * Test a given auxiliary value to determine if it has the default value
 * for that item, as described above. If all the auxiliary items have
 * their default values, no auxiliary place is necessary to represent them.
 * If any one of them is non-default, the auxiliary block is needed.
 */
#define RELAUX_ISDEFAULT_MOVE(_rdesc, _mv) (_mv == NULL)
#define RELAUX_ISDEFAULT_USYM(_rdesc, _usym) ((_rdesc)->rel_sym == _usym)
#define RELAUX_ISDEFAULT_OSDESC(_rdesc, _osdesc) \
        ((((_rdesc)->rel_isdesc == NULL) && (_osdesc == NULL)) || \
        ((_rdesc)->rel_isdesc && ((_rdesc)->rel_isdesc->is_osdesc == _osdesc)))
#define RELAUX_ISDEFAULT_TYPEDATA(_rdesc, _typedata) (_typedata == 0)

/*
 * Retrieve the value of an auxiliary relocation item, preserving the illusion
 * that every relocation descriptor has an auxiliary block attached. The
 * real implementation is that an auxiliary block is only present if one or
 * more auxiliary items have non-default values. These macros return the true
 * value if an auxiliary block is present, and the default value for the
 * item otherwise.
 */
#define RELAUX_GET_MOVE(_rdesc) \
        ((_rdesc)->rel_aux ? (_rdesc)->rel_aux->ra_move : NULL)
#define RELAUX_GET_USYM(_rdesc) \
        ((_rdesc)->rel_aux ? (_rdesc)->rel_aux->ra_usym : (_rdesc)->rel_sym)
#define RELAUX_GET_OSDESC(_rdesc) \
        ((_rdesc)->rel_aux ? (_rdesc)->rel_aux->ra_osdesc : \
        ((_rdesc)->rel_isdesc ? (_rdesc)->rel_isdesc->is_osdesc : NULL))
#define RELAUX_GET_TYPEDATA(_rdesc) \
```



```
        ((_rdesc)->rel_aux ? (_rdesc)->rel_aux->ra_typedata : 0)

/*
 * common flags used on the Rel_desc structure (defined in machrel.h).
 */
#define FLG_REL_GOT     0x00000001      /* relocation against GOT */
#define FLG_REL_PLT     0x00000002      /* relocation against PLT */
#define FLG_REL_BSS     0x00000004      /* relocation against BSS */
#define FLG_REL_LOAD    0x00000008      /* section loadable */
#define FLG_REL_SCNNDX  0x00000010      /* use section index for symbol ndx */
#define FLG_REL_CLVAL   0x00000020      /* clear VALUE for active relocation */
#define FLG_REL_ADVAL   0x00000040      /* add VALUE for output relocation, */
                                        /*      only relevant to SPARC and */
                                        /*      R_SPARC_RELATIVE */
#define FLG_REL_GOTCL   0x00000080      /* clear the GOT entry.  This is */
                                        /* relevant to RELA relocations, */
                                        /* not REL (i386) relocations */
#define FLG_REL_MOVETAB 0x00000100      /* Relocation against .SUNW_move */
                                        /*      adjustments required before */
                                        /*      actual relocation */
#define FLG_REL_NOINFO  0x00000200      /* Relocation comes from a section */
                                        /*      with a null sh_info field */
#define FLG_REL_REG     0x00000400      /* Relocation target is reg sym */
#define FLG_REL_FPTR    0x00000800      /* relocation against func. desc. */
#define FLG_REL_RFPTR1  0x00001000      /* Relative relocation against */
                                        /*   1st part of FD */
#define FLG_REL_RFPTR2  0x00002000      /* Relative relocation against */
                                        /*   2nd part of FD */
#define FLG_REL_DISP    0x00004000      /* *disp* relocation */
#define FLG_REL_STLS    0x00008000      /* IE TLS reference to */
                                        /*      static TLS GOT index */
#define FLG_REL_DTLS    0x00010000      /* GD TLS reference relative to */
                                        /*      dynamic TLS GOT index */
#define FLG_REL_MTLS    0x00020000      /* LD TLS reference against GOT */
#define FLG_REL_STTLS   0x00040000      /* LE TLS reference directly */
                                        /*      to static tls index */
#define FLG_REL_TLSFIX  0x00080000      /* relocation points to TLS instr. */
                                        /*      which needs updating */
#define FLG_REL_RELA    0x00100000      /* descriptor captures a Rela */
#define FLG_REL_GOTFIX  0x00200000      /* relocation points to GOTOP instr. */
                                        /*      which needs updating */
#define FLG_REL_NADDEND 0x00400000      /* Replace implicit addend in dest */
                                        /*      with value in rel_raddend */
                                        /*      Relevant to REL (i386) */
                                        /*      relocations, not to RELA. */

/*
 * We often need the name of the symbol contained in a relocation descriptor
 * for diagnostic or error output. This is usually the symbol name, but
 * we substitute a constructed name in some cases. Hence, the name is
 * generated on the fly by a private function within libld. This is the
 * prototype for that function.
 */
typedef const char *(* rel_desc_sname_func_t)(Rel_desc *);

/*
 * Header for a relocation descriptor cache buffer.
 */
struct rel_cachebuf {
        Rel_desc        *rc_end;
        Rel_desc        *rc_free;
        Rel_desc        rc_arr[1];
};

/*
 * Header for a relocation auxiliary descriptor cache buffer.
```

```
 */
struct rel_aux_cachebuf {
        Rel_aux         *rac_end;
        Rel_aux         *rac_free;
        Rel_aux         rac_arr[1];
};

/*
 * Convenience macro for traversing every relocation descriptor found within
 * a given relocation cache, transparently handling the cache buffers and
 * skipping any unallocated descriptors within the buffers.
 *
 * entry:
 *      _rel_cache - Relocate descriptor cache (Rel_cache) to traverse
 *      _idx - Aliste index variable for use by the macro
 *      _rcbp - Cache buffer pointer, for use by the macro
 *      _orsp - Rel_desc pointer, which will take on the value of a different
 *              relocation descriptor in the cache in each iteration.
 *
 * The caller must not assign new values to _idx, _rcbp, or _orsp within
 * the scope of REL_CACHE_TRAVERSE.
 */
#define REL_CACHE_TRAVERSE(_rel_cache, _idx, _rcbp, _orsp) \
        for (APLIST_TRAVERSE((_rel_cache)->rc_list, _idx, _rcbp)) \
                for (_orsp = _rcbp->rc_arr; _orsp < _rcbp->rc_free; _orsp++)

/*
 * Symbol value descriptor.  For relocatable objects, each symbols value is
 * its offset within its associated section.  Therefore, to uniquely define
 * each symbol within a relocatable object, record and sort the sh_offset and
 * symbol value.  This information is used to search for displacement
 * relocations as part of copy relocation validation.
 */
typedef struct {
        Addr            ssv_value;
        Sym_desc        *ssv_sdp;
} Ssv_desc;

/*
 * Input file processing structures.
 */
struct ifl_desc {                       /* input file descriptor */
        const char      *ifl_name;      /* full file name */
        const char      *ifl_soname;    /* shared object name */
        dev_t           ifl_stdev;      /* device id and inode number for .so */
        ino_t           ifl_stino;      /*      multiple inclusion checks */
        Ehdr            *ifl_ehdr;      /* elf header describing this file */
        Elf             *ifl_elf;       /* elf descriptor for this file */
        Sym_desc        **ifl_oldndx;   /* original symbol table indices */
        Sym_desc        *ifl_locs;      /* symbol desc version of locals */
        Ssv_desc        *ifl_sortsyms;  /* sorted list of symbols by value */
        Word            ifl_locscnt;    /* no. of local symbols to process */
        Word            ifl_symscnt;    /* total no. of symbols to process */
        Word            ifl_sortcnt;    /* no. of sorted symbols to process */
        Word            ifl_shnum;      /* number of sections in file */
        Word            ifl_shstrndx;   /* index to .shstrtab */
        Word            ifl_vercnt;     /* number of versions in file */
        Half            ifl_neededndx;  /* index to NEEDED in .dyn section */
        Word            ifl_flags;      /* explicit/implicit reference */
        Is_desc         **ifl_isdesc;   /* isdesc[scn ndx] = Is_desc ptr */
        Sdf_desc        *ifl_sdfdesc;   /* control definition */
        Versym          *ifl_versym;    /* version symbol table array */
        Ver_index       *ifl_verndx;    /* verndx[ver ndx] = Ver_index */
        APlist          *ifl_verdesc;   /* version descriptor list */
        APlist          *ifl_relsect;   /* relocation section list */
        Alist           *ifl_groups;    /* SHT_GROUP section list */
```



```
        Cap_desc        *ifl_caps;      /* capabilities descriptor */
};

#define FLG_IF_CMDLINE  0x00000001      /* full filename specified from the */
                                        /*      command line (no -l) */
#define FLG_IF_NEEDED   0x00000002      /* shared object should be recorded */
#define FLG_IF_DIRECT   0x00000004      /* establish direct bindings to this */
                                        /*      object */
#define FLG_IF_EXTRACT  0x00000008      /* file extracted from an archive */
#define FLG_IF_VERNEED  0x00000010      /* version dependency information is */
                                        /*      required */
#define FLG_IF_DEPREQD  0x00000020      /* dependency is required to satisfy */
                                        /*      symbol references */
#define FLG_IF_NEEDSTR  0x00000040      /* dependency specified by -Nn */
                                        /*      flag */
#define FLG_IF_IGNORE   0x00000080      /* ignore unused dependencies */
#define FLG_IF_NODIRECT 0x00000100      /* object contains symbols that */
                                        /*      cannot be directly bound to */
#define FLG_IF_LAZYLD   0x00000200      /* dependency should be lazy loaded */
#define FLG_IF_GRPPRM   0x00000400      /* dependency establishes a group */
#define FLG_IF_DISPPEND 0x00000800      /* displacement relocation done */
                                        /*      in the ld time. */
#define FLG_IF_DISPDONE 0x00001000      /* displacement relocation done */
                                        /*      at the run time */
#define FLG_IF_MAPFILE  0x00002000      /* file is a mapfile */
#define FLG_IF_HSTRTAB  0x00004000      /* file has a string section */
#define FLG_IF_FILEREF  0x00008000      /* file contains a section which */
                                        /*      is included in the output */
                                        /*      allocatable image */
#define FLG_IF_GNUVER   0x00010000      /* file used GNU-style versioning */
#define FLG_IF_ORDERED  0x00020000      /* ordered section processing */
                                        /*      required */
#define FLG_IF_OTOSCAP  0x00040000      /* convert object capabilities to */
                                        /*      symbol capabilities */
#define FLG_IF_DEFERRED 0x00080000      /* dependency is deferred */
#define FLG_IF_RTLDINF  0x00100000      /* dependency has DT_SUNW_RTLTINF set */
#define FLG_IF_GROUPS   0x00200000      /* input file has groups to process */

/*
 * Symbol states that require the generation of a DT_POSFLAG_1 .dynamic entry.
 */
#define MSK_IF_POSFLAG1 (FLG_IF_LAZYLD | FLG_IF_GRPPRM | FLG_IF_DEFERRED)

/*
 * Symbol states that require an associated Syminfo entry.
 */
#define MSK_IF_SYMINFO  (FLG_IF_LAZYLD | FLG_IF_DIRECT | FLG_IF_DEFERRED)


struct is_desc {                        /* input section descriptor */
        const char      *is_name;       /* original section name */
        const char      *is_sym_name;   /* NULL, or name string to use for */
                                        /*      related STT_SECTION symbols */
        Shdr            *is_shdr;       /* the elf section header */
        Ifl_desc        *is_file;       /* infile desc for this section */
        Os_desc         *is_osdesc;     /* new output section for this */
                                        /*      input section */
        Elf_Data        *is_indata;     /* input sections raw data */
        Is_desc         *is_symshndx;   /* related SHT_SYM_SHNDX section */
        Is_desc         *is_comdatkeep; /* If COMDAT section is discarded, */
                                        /*      this is section that was kept */
        Word            is_scnndx;      /* original section index in file */
        Word            is_ordndx;      /* index for section.  Used to decide */
                                        /*      where to insert section when */
                                        /*      reordering sections */
        Word            is_keyident;    /* key for SHF_{ORDERED|LINK_ORDER} */
```

```
930                                     /*      processing and ident used for */
931                                     /*       placing/ordering sections */
932         Word            is_flags;   /* Various flags */
933 };

935 #define FLG_IS_ORDERED  0x0001      /* this is a SHF_ORDERED section */
936 #define FLG_IS_KEY      0x0002      /* section requires sort keys */
937 #define FLG_IS_DISCARD  0x0004      /* section is to be discarded */
938 #define FLG_IS_RELUPD   0x0008      /* symbol defined here may have moved */
939 #define FLG_IS_SECTREF  0x0010      /* section has been referenced */
940 #define FLG_IS_GDATADEF 0x0020      /* section contains global data sym */
941 #define FLG_IS_EXTERNAL 0x0040      /* isp from a user file */
942 #define FLG_IS_INSTRMRG 0x0080      /* Usable SHF_MERGE|SHF_STRINGS sec */
943 #define FLG_IS_GNSTRMRG 0x0100      /* Generated mergeable string section */

945 #define FLG_IS_PLACE    0x0400      /* section requires to be placed */
946 #define FLG_IS_COMDAT   0x0800      /* section is COMDAT */
947 #define FLG_IS_EHFRAME  0x1000      /* section is .eh_frame */

949 /*
950  * Output sections contain lists of input sections that are assigned to them.
951  * These items fall into 4 categories:
952  *      BEFORE - Ordered sections that specify SHN_BEFORE, in input order.
953  *      ORDERED - Ordered sections that are sorted using unsorted sections
954  *              as the sort key.
955  *      DEFAULT - Sections that are placed into the output section
956  *              in input order.
957  *      AFTER - Ordered sections that specify SHN_AFTER, in input order.
958  */
959 #define OS_ISD_BEFORE   0
960 #define OS_ISD_ORDERED  1
961 #define OS_ISD_DEFAULT  2
962 #define OS_ISD_AFTER    3
963 #define OS_ISD_NUM      4
964 typedef APlist *os_isdecs_arr[OS_ISD_NUM];

966 /*
967  * Convenience macro for traversing every input section associated
968  * with a given output section. The primary benefit of this macro
969  * is that it preserves a precious level of code indentation in the
970  * code that uses it.
971  */
972 #define OS_ISDESCS_TRAVERSE(_list_idx, _osp, _idx, _isp) \
973         for (_list_idx = 0; _list_idx < OS_ISD_NUM; _list_idx++) \
974                 for (APLIST_TRAVERSE(_osp->os_isdescs[_list_idx], _idx, _isp))


977 /*
978  * Map file and output file processing structures
979  */
980 struct os_desc {                        /* Output section descriptor */
981         const char      *os_name;       /* the section name */
982         Elf_Scn         *os_scn;        /* the elf section descriptor */
983         Shdr            *os_shdr;       /* the elf section header */
984         Os_desc         *os_relosdesc;  /* the output relocation section */
985         APlist          *os_relisdescs; /* reloc input section descriptors */
986                                         /*      for this output section */
987         os_isdecs_arr   os_isdescs;     /* lists of input sections in output */
988         APlist          *os_mstrisdescs; /* FLG_IS_INSTRMRG input sections */
989         Sg_desc         *os_sgdesc;     /* segment os_desc is placed on */
990         Elf_Data        *os_outdata;    /* output sections raw data */
991         avl_tree_t      *os_comdats;    /* AVL tree of COMDAT input sections */
992                                         /*      associated to output section */
993         Word            os_identndx;    /* section identifier for input */
994                                         /*      section processing, followed */
995                                         /*      by section symbol index */
```



```
996         Word            os_ordndx;      /* index for section.  Used to decide */
997                                         /*      where to insert section when */
998                                         /*      reordering sections */
999         Xword           os_szoutrels;   /* size of output relocation section */
1000        uint_t          os_namehash;    /* hash on section name */
1001        uchar_t         os_flags;       /* various flags */
1002 };

1004 #define FLG_OS_KEY              0x01    /* section requires sort keys */
1005 #define FLG_OS_OUTREL           0x02    /* output rel against this section */
1006 #define FLG_OS_SECTREF          0x04    /* isps are not affected by -zignore */
1007 #define FLG_OS_EHFRAME          0x08    /* section is .eh_frame */

1009 /*
1010  * The sg_id field of the segment descriptor is used to establish the default
1011  * order for program headers and segments in the output object. Segments are
1012  * ordered according to the following SGID values that classify them based on
1013  * their attributes. The initial set of built in segments are in this order,
1014  * and new mapfile defined segments are inserted into these groups. Within a
1015  * given SGID group, the position of new segments depends on the syntax
1016  * version of the mapfile that creates them. Version 1 (original sysv)
1017  * mapfiles place the new segment at the head of their group (reverse creation
1018  * order). The newer syntax places them at the end, following the others
1019  * (creation order).
1020  *
1021  * Note that any new segments must always be added after PT_PHDR and
1022  * PT_INTERP (refer Generic ABI, Page 5-4).
1023  */
1024 #define SGID_PHDR       0       /* PT_PHDR */
1025 #define SGID_INTERP     1       /* PT_INTERP */
1026 #define SGID_SUNWCAP    2       /* PT_SUNWCAP */
1027 #define SGID_TEXT       3       /* PT_LOAD */
1028 #define SGID_DATA       4       /* PT_LOAD */
1029 #define SGID_BSS        5       /* PT_LOAD */
1030 #if     defined(_ELF64)
1031 #define SGID_LRODATA    6       /* PT_LOAD (amd64-only) */
1032 #define SGID_LDATA      7       /* PT_LOAD (amd64-only) */
1033 #endif
1034 #define SGID_TEXT_EMPTY 8       /* PT_LOAD, reserved (?E in version 1 syntax) */
1035 #define SGID_NULL_EMPTY 9       /* PT_NULL, reserved (?E in version 1 syntax) */
1036 #define SGID_DYN        10      /* PT_DYNAMIC */
1037 #define SGID_DTRACE     11      /* PT_SUNWDTRACE */
1038 #define SGID_TLS        12      /* PT_TLS */
1039 #define SGID_UNWIND     13      /* PT_SUNW_UNWIND */
1040 #define SGID_SUNWSTACK  14      /* PT_SUNWSTACK */
1041 #define SGID_NOTE       15      /* PT_NOTE */
1042 #define SGID_NULL       16      /* PT_NULL,  mapfile defined empty phdr slots */
1043                                 /*      for use by post processors */
1044 #define SGID_EXTRA      17      /* PT_NULL (final catchall) */

1046 typedef Half sg_flags_t;
1047 struct sg_desc {                        /* output segment descriptor */
1048         Word            sg_id;         /* segment identifier (for sorting) */
1049         Phdr            sg_phdr;       /* segment header for output file */
1050         const char      *sg_name;      /* segment name for PT_LOAD, PT_NOTE, */
1051                                        /*      and PT_NULL, otherwise NULL */
1052         Xword           sg_round;      /* data rounding required (mapfile) */
1053         Xword           sg_length;     /* maximum segment length; if 0 */
1054                                        /*      segment is not specified */
1055         APlist          *sg_osdescs;   /* list of output section descriptors */
1056         APlist          *sg_is_order;  /* list of entry criteria */
1057                                        /*      giving input section order */
1058         Alist           *sg_os_order;  /* list specifying output section */
1059                                        /*      ordering for the segment */
1060         sg_flags_t      sg_flags;
1061         APlist          *sg_sizesym;   /* size symbols for this segment */
```

```
1062         Xword           sg_align;       /* LCM of sh_addralign */
1063         Elf_Scn         *sg_fscn;       /* the SCN of the first section. */
1064         avl_node_t      sg_avlnode;     /* AVL book-keeping */
1065 };

1067 #define FLG_SG_P_VADDR          0x0001  /* p_vaddr segment attribute set */
1068 #define FLG_SG_P_PADDR          0x0002  /* p_paddr segment attribute set */
1069 #define FLG_SG_LENGTH           0x0004  /* length segment attribute set */
1070 #define FLG_SG_P_ALIGN          0x0008  /* p_align segment attribute set */
1071 #define FLG_SG_ROUND            0x0010  /* round segment attribute set */
1072 #define FLG_SG_P_FLAGS          0x0020  /* p_flags segment attribute set */
1073 #define FLG_SG_P_TYPE           0x0040  /* p_type segment attribute set */
1074 #define FLG_SG_IS_ORDER         0x0080  /* input section ordering is required */
1075                                         /*      for this segment. */
1076 #define FLG_SG_NOHDR            0x0100  /* don't map ELF or phdrs into */
1077                                         /*      this segment */
1078 #define FLG_SG_EMPTY            0x0200  /* an empty segment specification */
1079                                         /*      no input sections will be */
1080                                         /*      associated to this section */
1081 #define FLG_SG_KEY              0x0400  /* segment requires sort keys */
1082 #define FLG_SG_NODISABLE        0x0800  /* FLG_SG_DISABLED is not allowed on */
1083                                         /*      this segment */
1084 #define FLG_SG_DISABLED         0x1000  /* this segment is disabled */
1085 #define FLG_SG_PHREQ            0x2000  /* this segment requires a program */
1086                                         /* header */
1087 #define FLG_SG_ORDERED          0x4000  /* SEGMENT_ORDER segment */

1089 struct sec_order {
1090         const char      *sco_secname;   /* section name to be ordered */
1091         Half            sco_flags;
1092 };

1094 #define FLG_SGO_USED    0x0001          /* was ordering used? */

1096 typedef Half ec_flags_t;
1097 struct ent_desc {                       /* input section entrance criteria */
1098         const char      *ec_name;       /* entrace criteria name, or NULL */
1099         Alist           *ec_files;      /* files from which to accept */
1100                                         /*      sections */
1101         const char      *ec_is_name;    /* input section name to match */
1102                                         /*      (NULL if none) */
1103         Word            ec_type;        /* section type */
1104         Word            ec_attrmask;    /* section attribute mask (AWX) */
1105         Word            ec_attrbits;    /* sections attribute bits */
1106         Sg_desc         *ec_segment;    /* output segment to enter if matched */
1107         Word            ec_ordndx;      /* index to determine where section */
1108                                         /*      meeting this criteria should */
1109                                         /*      inserted. Used for reordering */
1110                                         /*      of sections. */
1111         ec_flags_t      ec_flags;
1112         avl_node_t      ec_avlnode;     /* AVL book-keeping */
1113 };

1115 #define FLG_EC_BUILTIN  0x0001          /* built in descriptor */
1116 #define FLG_EC_USED     0x0002          /* entrance criteria met? */
1117 #define FLG_EC_CATCHALL 0x0004          /* Catches any section */

1119 /*
1120  * Ent_desc_file is the type of element maintained in the ec_files Alist
1121  * of an entrance criteria descriptor. Each item maintains one file
1122  * path, and a set of flags that specify the type of comparison it implies,
1123  * and other information about it. The comparison type is maintained in
1124  * the bottom byte of the flags.
1125  */
1126 #define TYP_ECF_MASK            0x00ff  /* Comparison type mask */
1127 #define TYP_ECF_PATH            0       /* Compare to file path */
```



```
1128 #define TYP_ECF_BASENAME        1       /* Compare to file basename */
1129 #define TYP_ECF_OBJNAME         2       /* Compare to regular file basename, */
1130                                         /*      or to archive member name */
1131 #define TYP_ECF_NUM             3

1133 #define FLG_ECF_ARMEMBER        0x0100  /* name includes archive member */

1135 typedef struct {
1136         Word            edf_flags;      /* Type of comparison */
1137         const char      *edf_name;      /* String to compare to */
1138         size_t          edf_name_len;   /* strlen(edf_name) */
1139 } Ent_desc_file;

1141 /*
1142  * One structure is allocated for a move entry, and associated to the symbol
1143  * against which a move is targeted.
1144  */
1145 typedef struct {
1146         Move            *md_move;       /* original Move entry */
1147         Xword           md_start;       /* start position */
1148         Xword           md_len;         /* length of initialization */
1149         Word            md_oidx;        /* output Move entry index */
1150 } Mv_desc;

1152 /*
1153  * Symbol descriptor.
1154  */
1155 typedef Lword           sd_flag_t;
1156 struct sym_desc {
1157         Alist           *sd_GOTndxs;    /* list of associated GOT entries */
1158         Sym             *sd_sym;        /* pointer to symbol table entry */
1159         Sym             *sd_osym;       /* copy of the original symbol entry */
1160                                         /*      used only for local partial */
1161         Alist           *sd_move;       /* move information associated with a */
1162                                         /*      partially initialized symbol */
1163         const char      *sd_name;       /* symbols name */
1164         Ifl_desc        *sd_file;       /* file where symbol is taken */
1165         Is_desc         *sd_isc;        /* input section of symbol definition */
1166         Sym_aux         *sd_aux;        /* auxiliary global symbol info. */
1167         Word            sd_symndx;      /* index in output symbol table */
1168         Word            sd_shndx;       /* sect. index sym is associated w/ */
1169         sd_flag_t       sd_flags;       /* state flags */
1170         Half            sd_ref;         /* reference definition of symbol */
1171 };

1173 /*
1174  * The auxiliary symbol descriptor contains the additional information (beyond
1175  * the symbol descriptor) required to process global symbols.  These symbols are
1176  * accessed via an internal symbol hash table where locality of reference is
1177  * important for performance.
1178  */
1179 struct sym_aux {
1180         APlist          *sa_dfiles;     /* files where symbol is defined */
1181         Sym             sa_sym;         /* copy of symtab entry */
1182         const char      *sa_vfile;      /* first unavailable definition */
1183         const char      *sa_rfile;      /* file with first symbol referenced */
1184         Word            sa_hash;        /* the pure hash value of symbol */
1185         Word            sa_PLTndx;      /* index into PLT for symbol */
1186         Word            sa_PLTGOTndx;   /* GOT entry indx for PLT indirection */
1187         Word            sa_linkndx;     /* index of associated symbol from */
1188                                         /*      ET_DYN file */
1189         Half            sa_symspec;     /* special symbol ids */
1190         Half            sa_overndx;     /* output file versioning index */
1191         Half            sa_dverndx;     /* dependency versioning index */
1192         Os_desc         *sa_boundsec;   /* output section of SECBOUND_ syms */
1193 #endif /* ! codereview */
```

```
1194 };

1196 /*
1197  * Nodes used to track symbols in the global AVL symbol dictionary.
1198  */
1199 struct sym_avlnode {
1200         avl_node_t      sav_node;       /* AVL node */
1201         Word            sav_hash;       /* symbol hash value */
1202         const char      *sav_name;      /* symbol name */
1203         Sym_desc        *sav_sdp;       /* symbol descriptor */
1204 };

1206 /*
1207  * These are the ids for processing of 'Special symbols'.  They are used
1208  * to set the sym->sd_aux->sa_symspec field.
1209  */
1210 #define SDAUX_ID_ETEXT          1       /* etext && _etext symbol */
1211 #define SDAUX_ID_EDATA          2       /* edata && _edata symbol */
1212 #define SDAUX_ID_END            3       /* end, _end, && _END_ symbol */
1213 #define SDAUX_ID_DYN            4       /* DYNAMIC && _DYNAMIC symbol */
1214 #define SDAUX_ID_PLT            5       /* _PROCEDURE_LINKAGE_TABLE_ symbol */
1215 #define SDAUX_ID_GOT            6       /* _GLOBAL_OFFSET_TABLE_ symbol */
1216 #define SDAUX_ID_START          7       /* START_ && _START_ symbol */
1217 #define SDAUX_ID_SECBOUND_START 8       /* __start_<section> symbols */
1218 #define SDAUX_ID_SECBOUND_STOP  9       /* __stop_<section> symbols */
1219 #endif /* ! codereview */

1221 /*
1222  * Flags for sym_desc.sd_flags
1223  */
1224 #define FLG_SY_MVTOCOMM 0x00000001      /* assign symbol to common (.bss) */
1225                                         /*      this is a result of a */
1226                                         /*      copy reloc against sym */
1227 #define FLG_SY_GLOBREF  0x00000002      /* a global reference has been seen */
1228 #define FLG_SY_WEAKDEF  0x00000004      /* a weak definition has been used */
1229 #define FLG_SY_CLEAN    0x00000008      /* 'Sym' entry points to original */
1230                                         /*      input file (read-only). */
1231 #define FLG_SY_UPREQD   0x00000010      /* symbol value update is required, */
1232                                         /*      either it's used as an entry */
1233                                         /*      point or for relocation, but */
1234                                         /*      it must be updated even if */
1235                                         /*      the -s flag is in effect */
1236 #define FLG_SY_NOTAVAIL 0x00000020      /* symbol is not available to the */
1237                                         /*      application either because it */
1238                                         /*      originates from an implicitly */
1239                                         /*      referenced shared object, or */
1240                                         /*      because it is not part of a */
1241                                         /*      specified version. */
1242 #define FLG_SY_REDUCED  0x00000040      /* a global is reduced to local */
1243 #define FLG_SY_VERSPROM 0x00000080      /* version definition has been */
1244                                         /*      promoted to output file */
1245 #define FLG_SY_PROT     0x00000100      /* stv_protected visibility seen */
1246 #define FLG_SY_MAPREF   0x00000200      /* symbol reference generated by user */
1247                                         /*      from mapfile */
1248 #define FLG_SY_REFRSD   0x00000400      /* symbols sd_ref has been raised */
1249                                         /*      due to a copy-relocs */
1250                                         /*      weak-strong pairing */
1251 #define FLG_SY_INTPOSE  0x00000800      /* symbol defines an interposer */
1252 #define FLG_SY_INVALID  0x00001000      /* unwanted/erroneous symbol */
1253 #define FLG_SY_SMGOT    0x00002000      /* small got index assigned to symbol */
1254                                         /*      sparc only */
1255 #define FLG_SY_PARENT   0x00004000      /* symbol to be found in parent */
1256                                         /*    only used with direct bindings */
1257 #define FLG_SY_LAZYLD   0x00008000      /* symbol to cause lazyloading of */
1258                                         /*      parent object */
1259 #define FLG_SY_ISDISC   0x00010000      /* symbol is a member of a DISCARDED */
```



```
1260                                         /*      section (COMDAT) */
1261 #define FLG_SY_PAREXPN  0x00020000      /* partially init. symbol to be */
1262                                         /*      expanded */
1263 #define FLG_SY_PLTPAD   0x00040000      /* pltpadding has been allocated for */
1264                                         /*      this symbol */
1265 #define FLG_SY_REGSYM   0x00080000      /* REGISTER symbol (sparc only) */
1266 #define FLG_SY_SOFOUND  0x00100000      /* compared against an SO definition */
1267 #define FLG_SY_EXTERN   0x00200000      /* symbol is external, allows -zdefs */
1268                                         /*    error suppression */
1269 #define FLG_SY_MAPUSED  0x00400000      /* mapfile symbol used (occurred */
1270                                         /*    within a relocatable object) */
1271 #define FLG_SY_COMMEXP  0x00800000      /* COMMON symbol which has been */
1272                                         /*      allocated */
1273 #define FLG_SY_CMDREF   0x01000000      /* symbol was referenced from the */
1274                                         /*      command line.  (ld -u <>, */
1275                                         /*      ld -zrtldinfo=<>, ...) */
1276 #define FLG_SY_SPECSEC  0x02000000      /* section index is reserved value */
1277                                         /*      ABS, COMMON, ... */
1278 #define FLG_SY_TENTSYM  0x04000000      /* tentative symbol */
1279 #define FLG_SY_VISIBLE  0x08000000      /* symbols visibility determined */
1280 #define FLG_SY_STDFLTR  0x10000000      /* symbol is a standard filter */
1281 #define FLG_SY_AUXFLTR  0x20000000      /* symbol is an auxiliary filter */
1282 #define FLG_SY_DYNSORT  0x40000000      /* req. in dyn[sym|tls]sort section */
1283 #define FLG_SY_NODYNSORT 0x80000000     /* excluded from dyn[sym_tls]sort sec */

1285 #define FLG_SY_DEFAULT  0x0000100000000 /* global symbol, default */
1286 #define FLG_SY_SINGLE   0x0000200000000 /* global symbol, singleton defined */
1287 #define FLG_SY_PROTECT  0x0000400000000 /* global symbol, protected defined */
1288 #define FLG_SY_EXPORT   0x0000800000000 /* global symbol, exported defined */

1290 #define MSK_SY_GLOBAL \
1291         (FLG_SY_DEFAULT | FLG_SY_SINGLE | FLG_SY_PROTECT | FLG_SY_EXPORT)
1292                                         /* this mask indicates that the */
1293                                         /*    symbol has been explicitly */
1294                                         /*    defined within a mapfile */
1295                                         /*    definition, and is a candidate */
1296                                         /*    for versioning */

1298 #define FLG_SY_HIDDEN   0x0001000000000 /* global symbol, reduce to local */
1299 #define FLG_SY_ELIM     0x0002000000000 /* global symbol, eliminate */
1300 #define FLG_SY_IGNORE   0x0004000000000 /* global symbol, ignored */

1302 #define MSK_SY_LOCAL    (FLG_SY_HIDDEN | FLG_SY_ELIM | FLG_SY_IGNORE)
1303                                         /* this mask allows all local state */
1304                                         /*    flags to be removed when the */
1305                                         /*    symbol is copy relocated */

1307 #define FLG_SY_EXPDEF   0x0008000000000 /* symbol visibility defined */
1308                                         /*    explicitly */

1310 #define MSK_SY_NOAUTO   (FLG_SY_SINGLE | FLG_SY_EXPORT | FLG_SY_EXPDEF)
1311                                         /* this mask indicates that the */
1312                                         /*    symbol is not a candidate for */
1313                                         /*    auto-reduction/elimination */

1315 #define FLG_SY_MAPFILE  0x0010000000000 /* symbol attribute defined in a */
1316                                         /*    mapfile */
1317 #define FLG_SY_DIR      0x0020000000000 /* global symbol, direct bindings */
1318 #define FLG_SY_NDIR     0x0040000000000 /* global symbol, nondirect bindings */
1319 #define FLG_SY_OVERLAP  0x0080000000000 /* move entry overlap detected */
1320 #define FLG_SY_CAP      0x0100000000000 /* symbol is associated with */
1321                                         /*    capabilities */
1322 #define FLG_SY_DEFERRED 0x0200000000000 /* symbol should not be bound to */
1323                                         /*      during BIND_NOW relocations */

1325 /*
```

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

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

/*
 * Structure to manage the shared object definition lists.  There are two lists
 * that use this structure:
 *
 *  -   ofl_soneed; maintain the list of implicitly required dependencies
 *      (ie. shared objects needed by other shared objects).  These definitions
 *      may include RPATH's required to locate the dependencies, and any
 *      version requirements.
 *
 *  -   ofl_socntl; maintains the shared object control definitions.  These are
 *      provided by the user (via a mapfile) and are used to indicate any
 *      version control requirements.
 */
struct  sdf_desc {
        const char      *sdf_name;      /* the shared objects file name */
        char            *sdf_rpath;     /* library search path DT_RPATH */
        const char      *sdf_rfile;     /* referencing file for diagnostics */
        Ifl_desc        *sdf_file;      /* the final input file descriptor */
        Alist           *sdf_vers;      /* list of versions that are required */
                                        /*      from this object */
        Alist           *sdf_verneed;   /* list of VERNEEDS to create for */
                                        /*      object via mapfile ADDVERS */
        Word            sdf_flags;
};

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

/*
 * Structure to manage shared object version usage requirements.
 */
struct  sdv_desc {
        const char      *sdv_name;      /* version name */
        const char      *sdv_ref;       /* versions reference */
        Word            sdv_flags;      /* flags */
};

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

/*
 * Structures to manage versioning information.  Two versioning structures are
 * defined:
 *
 *   -  a version descriptor maintains a linked list of versions and their
 *      associated dependencies.  This is used to build the version definitions
 *      for an image being created (see map_symbol), and to determine the
 *      version dependency graph for any input files that are versioned.
 *
 *   -  a version index array contains each version of an input file that is
 *      being processed.  It informs us which versions are available for
 *      binding, and is used to generate any version dependency information.
 */
struct  ver_desc {
```



```
        const char      *vd_name;       /* version name */
        Ifl_desc        *vd_file;       /* file that defined version */
        Word            vd_hash;        /* hash value of name */
        Half            vd_ndx;         /* coordinates with symbol index */
        Half            vd_flags;       /* version information */
        APlist          *vd_deps;       /* version dependencies */
        Ver_desc        *vd_ref;        /* dependency's first reference */
};

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

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

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

/*
 * isalist(1) descriptor - used to break an isalist string into its component
 * options.
 */
struct  isa_opt {
        char            *isa_name;      /* individual isa option name */
        size_t          isa_namesz;     /*      and associated size */
};

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

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

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

```
/*
 * Indexes into the ld_support_funcs[] table.
 */
typedef enum {
        LDS_VERSION = 0,        /* Must be first and have value 0 */
        LDS_INPUT_DONE,
        LDS_START,
        LDS_ATEXIT,
        LDS_OPEN,
        LDS_FILE,
        LDS_INSEC,
        LDS_SEC,
        LDS_NUM
} Support_ndx;

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

typedef struct ar_aux {
        Sym_desc        *au_syms;       /* internal symbol descriptor */
        Ar_mem          *au_mem;        /* associated member */
} Ar_aux;

#define FLG_ARMEM_PROC  (Ar_mem *)-1

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

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

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


/*
```



```
 * Function Declarations.
 */
#if     defined(_ELF64)

#define ld_create_outfile       ld64_create_outfile
#define ld_ent_setup            ld64_ent_setup
#define ld_init_strings         ld64_init_strings
#define ld_init_target          ld64_init_target
#define ld_make_sections        ld64_make_sections
#define ld_main                 ld64_main
#define ld_ofl_cleanup          ld64_ofl_cleanup
#define ld_process_mem          ld64_process_mem
#define ld_reloc_init           ld64_reloc_init
#define ld_reloc_process        ld64_reloc_process
#define ld_sym_validate         ld64_sym_validate
#define ld_update_outfile       ld64_update_outfile

#else

#define ld_create_outfile       ld32_create_outfile
#define ld_ent_setup            ld32_ent_setup
#define ld_init_strings         ld32_init_strings
#define ld_init_target          ld32_init_target
#define ld_make_sections        ld32_make_sections
#define ld_main                 ld32_main
#define ld_ofl_cleanup          ld32_ofl_cleanup
#define ld_process_mem          ld32_process_mem
#define ld_reloc_init           ld32_reloc_init
#define ld_reloc_process        ld32_reloc_process
#define ld_sym_validate         ld32_sym_validate
#define ld_update_outfile       ld32_update_outfile

#endif

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

extern int              ld32_main(int, char **, Half);
extern int              ld64_main(int, char **, Half);

extern uintptr_t        ld_create_outfile(Ofl_desc *);
extern uintptr_t        ld_ent_setup(Ofl_desc *, Xword);
extern uintptr_t        ld_init_strings(Ofl_desc *);
extern int              ld_init_target(Lm_list *, Half mach);
extern uintptr_t        ld_make_sections(Ofl_desc *);
extern void             ld_ofl_cleanup(Ofl_desc *);
extern Ifl_desc         *ld_process_mem(const char *, const char *, char *,
                            size_t, Ofl_desc *, Rej_desc *);
extern uintptr_t        ld_reloc_init(Ofl_desc *);
extern uintptr_t        ld_reloc_process(Ofl_desc *);
extern uintptr_t        ld_sym_validate(Ofl_desc *);
extern uintptr_t        ld_update_outfile(Ofl_desc *);

#ifdef  __cplusplus
}
#endif

#endif  /* _LIBLD_H */
```

****
8404 Sun Feb 24 19:19:07 2019
new/usr/src/cmd/sgs/include/sgs.h
ld should reject kernel modules as input
****
_____unchanged_portion_omitted_

```
177 #define SGS_REJ_NONE            0
178 #define SGS_REJ_MACH            1       /* wrong ELF machine type */
179 #define SGS_REJ_CLASS           2       /* wrong ELF class (32-bit/64-bit) */
180 #define SGS_REJ_DATA            3       /* wrong ELF data format (MSG/LSB) */
181 #define SGS_REJ_TYPE            4       /* bad ELF type */
182 #define SGS_REJ_BADFLAG         5       /* bad ELF flags value */
183 #define SGS_REJ_MISFLAG         6       /* mismatched ELF flags value */
184 #define SGS_REJ_VERSION         7       /* mismatched ELF/lib version */
185 #define SGS_REJ_HAL             8       /* HAL R1 extensions required */
186 #define SGS_REJ_US3             9       /* Sun UltraSPARC III extensions */
187                                         /*      required */
188 #define SGS_REJ_STR             10      /* generic error - info is a string */
189 #define SGS_REJ_UNKFILE         11      /* unknown file type */
190 #define SGS_REJ_UNKCAP          12      /* unknown capabilities */
191 #define SGS_REJ_HWCAP_1         13      /* hardware capabilities mismatch */
192 #define SGS_REJ_SFCAP_1         14      /* software capabilities mismatch */
193 #define SGS_REJ_MACHCAP         15      /* machine capability mismatch */
194 #define SGS_REJ_PLATCAP         16      /* platform capability mismatch */
195 #define SGS_REJ_HWCAP_2         17      /* hardware capabilities mismatch */
196 #define SGS_REJ_ARCHIVE         18      /* archive used in invalid context */
197 #define SGS_REJ_KMOD            19      /* object is a kernel module */
198 #define SGS_REJ_NUM             20
197 #define SGS_REJ_NUM             19

200 #define FLG_REJ_ALTER           0x01    /* object name is an alternative */

202 /*
203  * For those source files used both inside and outside of the
204  * libld source base (tools/common/string_table.c) we can
205  * automatically switch between the allocation models
206  * based off of the 'cc -DUSE_LIBLD_MALLOC' flag.
207  */
208 #ifdef  USE_LIBLD_MALLOC
209 #define calloc(x, a)            libld_malloc(((size_t)x) * ((size_t)a))
210 #define free                    libld_free
211 #define malloc                  libld_malloc
212 #define realloc                 libld_realloc

214 #define libld_calloc(x, a)      libld_malloc(((size_t)x) * ((size_t)a))
215 extern void                     libld_free(void *);
216 extern void                     *libld_malloc(size_t);
217 extern void                     *libld_realloc(void *, size_t);
218 #endif

220 /*
221  * Data structures (defined in libld.h).
222  */
223 typedef struct audit_desc       Audit_desc;
224 typedef struct audit_info       Audit_info;
225 typedef struct audit_list       Audit_list;
226 typedef struct cap_desc         Cap_desc;
227 typedef struct ent_desc         Ent_desc;
228 typedef struct group_desc       Group_desc;
229 typedef struct ifl_desc         Ifl_desc;
230 typedef struct is_desc          Is_desc;
231 typedef struct isa_desc         Isa_desc;
232 typedef struct isa_opt          Isa_opt;
233 typedef struct os_desc          Os_desc;
234 typedef struct ofl_desc         Ofl_desc;
```



```
235 typedef struct rel_cache        Rel_cache;
236 typedef struct rel_cachebuf     Rel_cachebuf;
237 typedef struct rel_aux_cachebuf Rel_aux_cachebuf;
238 typedef struct rel_aux          Rel_aux;
239 typedef struct rel_desc         Rel_desc;
240 typedef struct sdf_desc         Sdf_desc;
241 typedef struct sdv_desc         Sdv_desc;
242 typedef struct sec_order        Sec_order;
243 typedef struct sg_desc          Sg_desc;
244 typedef struct sort_desc        Sort_desc;
245 typedef struct sym_avlnode      Sym_avlnode;
246 typedef struct sym_aux          Sym_aux;
247 typedef struct sym_desc         Sym_desc;
248 typedef struct uts_desc         Uts_desc;
249 typedef struct ver_desc         Ver_desc;
250 typedef struct ver_index        Ver_index;

252 /*
253  * Data structures defined in rtld.h.
254  */
255 typedef struct lm_list          Lm_list;
256 #ifdef _SYSCALL32
257 typedef struct lm_list32        Lm_list32;
258 #endif  /* _SYSCALL32 */

260 /*
261  * For the various utilities that include sgs.h
262  */
263 extern int      assfail(const char *, const char *, int);
264 extern void     eprintf(Lm_list *, Error, const char *, ...);
265 extern void     veprintf(Lm_list *, Error, const char *, va_list);
266 extern uint_t   sgs_str_hash(const char *);
267 extern uint_t   findprime(uint_t);

269 #endif /* _ASM */

271 #ifdef  __cplusplus
272 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   34970 Sun Feb 24 19:19:08 2019
new/usr/src/cmd/sgs/libconv/common/dynamic.c
ld: implement -ztype and rework option parsing
**********************************************************
______unchanged_portion_omitted_

 419 const conv_ds_t **
 420 conv_dyn_tag_strings(conv_iter_osabi_t osabi, Half mach,
 421     Conv_fmt_flags_t fmt_flags)
 422 {
 423         /*
 424          * Maximum # of items that can be in the returned array. Size this
 425          * by counting the maximum depth in the switch statement that fills
 426          * retarr at the end of this function.
 427          */
 428 #define MAX_RET 12

 430         /*
 431          * Generic dynamic tags:
 432          * -    Note hole between DT_FLAGS and DT_PREINIT_ARRAY (tag 32).
 433          *      We use a 0, which is the signal for "not defined".
 434          * -    This range has alternative names for dump, requiring an
 435          *      additional array.
 436          */
 437         static const Msg        tags_null_cf[] = {
 438                 MSG_DT_NULL_CF,                 MSG_DT_NEEDED_CF,
 439                 MSG_DT_PLTRELSZ_CF,             MSG_DT_PLTGOT_CF,
 440                 MSG_DT_HASH_CF,                 MSG_DT_STRTAB_CF,
 441                 MSG_DT_SYMTAB_CF,               MSG_DT_RELA_CF,
 442                 MSG_DT_RELASZ_CF,               MSG_DT_RELAENT_CF,
 443                 MSG_DT_STRSZ_CF,                MSG_DT_SYMENT_CF,
 444                 MSG_DT_INIT_CF,                 MSG_DT_FINI_CF,
 445                 MSG_DT_SONAME_CF,               MSG_DT_RPATH_CF,
 446                 MSG_DT_SYMBOLIC_CF,             MSG_DT_REL_CF,
 447                 MSG_DT_RELSZ_CF,                MSG_DT_RELENT_CF,
 448                 MSG_DT_PLTREL_CF,               MSG_DT_DEBUG_CF,
 449                 MSG_DT_TEXTREL_CF,              MSG_DT_JMPREL_CF,
 450                 MSG_DT_BIND_NOW_CF,             MSG_DT_INIT_ARRAY_CF,
 451                 MSG_DT_FINI_ARRAY_CF,           MSG_DT_INIT_ARRAYSZ_CF,
 452                 MSG_DT_FINI_ARRAYSZ_CF,         MSG_DT_RUNPATH_CF,
 453                 MSG_DT_FLAGS_CF,                0,
 454                 MSG_DT_PREINIT_ARRAY_CF,        MSG_DT_PREINIT_ARRAYSZ_CF
 455         };
 456         static const Msg        tags_null_cfnp[] = {
 457                 MSG_DT_NULL_CFNP,               MSG_DT_NEEDED_CFNP,
 458                 MSG_DT_PLTRELSZ_CFNP,           MSG_DT_PLTGOT_CFNP,
 459                 MSG_DT_HASH_CFNP,               MSG_DT_STRTAB_CFNP,
 460                 MSG_DT_SYMTAB_CFNP,             MSG_DT_RELA_CFNP,
 461                 MSG_DT_RELASZ_CFNP,             MSG_DT_RELAENT_CFNP,
 462                 MSG_DT_STRSZ_CFNP,              MSG_DT_SYMENT_CFNP,
 463                 MSG_DT_INIT_CFNP,               MSG_DT_FINI_CFNP,
 464                 MSG_DT_SONAME_CFNP,             MSG_DT_RPATH_CFNP,
 465                 MSG_DT_SYMBOLIC_CFNP,           MSG_DT_REL_CFNP,
 466                 MSG_DT_RELSZ_CFNP,              MSG_DT_RELENT_CFNP,
 467                 MSG_DT_PLTREL_CFNP,             MSG_DT_DEBUG_CFNP,
 468                 MSG_DT_TEXTREL_CFNP,            MSG_DT_JMPREL_CFNP,
 469                 MSG_DT_BIND_NOW_CFNP,           MSG_DT_INIT_ARRAY_CFNP,
 470                 MSG_DT_FINI_ARRAY_CFNP,         MSG_DT_INIT_ARRAYSZ_CFNP,
 471                 MSG_DT_FINI_ARRAYSZ_CFNP,       MSG_DT_RUNPATH_CFNP,
 472                 MSG_DT_FLAGS_CFNP,              0,
 473                 MSG_DT_PREINIT_ARRAY_CFNP,      MSG_DT_PREINIT_ARRAYSZ_CFNP
 474         };
 475         static const Msg        tags_null_nf[] = {
 476                 MSG_DT_NULL_NF,                 MSG_DT_NEEDED_NF,
 477                 MSG_DT_PLTRELSZ_NF,             MSG_DT_PLTGOT_NF,
```



```
 478                 MSG_DT_HASH_NF,                 MSG_DT_STRTAB_NF,
 479                 MSG_DT_SYMTAB_NF,               MSG_DT_RELA_NF,
 480                 MSG_DT_RELASZ_NF,               MSG_DT_RELAENT_NF,
 481                 MSG_DT_STRSZ_NF,                MSG_DT_SYMENT_NF,
 482                 MSG_DT_INIT_NF,                 MSG_DT_FINI_NF,
 483                 MSG_DT_SONAME_NF,               MSG_DT_RPATH_NF,
 484                 MSG_DT_SYMBOLIC_NF,             MSG_DT_REL_NF,
 485                 MSG_DT_RELSZ_NF,                MSG_DT_RELENT_NF,
 486                 MSG_DT_PLTREL_NF,               MSG_DT_DEBUG_NF,
 487                 MSG_DT_TEXTREL_NF,              MSG_DT_JMPREL_NF,
 488                 MSG_DT_BIND_NOW_NF,             MSG_DT_INIT_ARRAY_NF,
 489                 MSG_DT_FINI_ARRAY_NF,           MSG_DT_INIT_ARRAYSZ_NF,
 490                 MSG_DT_FINI_ARRAYSZ_NF,         MSG_DT_RUNPATH_NF,
 491                 MSG_DT_FLAGS_NF,                0,
 492                 MSG_DT_PREINIT_ARRAY_NF,        MSG_DT_PREINIT_ARRAYSZ_NF
 493         };
 494         static const Msg        tags_null_dmp[] = {
 495                 MSG_DT_NULL_CFNP,               MSG_DT_NEEDED_CFNP,
 496                 MSG_DT_PLTRELSZ_DMP,            MSG_DT_PLTGOT_CFNP,
 497                 MSG_DT_HASH_CFNP,               MSG_DT_STRTAB_CFNP,
 498                 MSG_DT_SYMTAB_CFNP,             MSG_DT_RELA_CFNP,
 499                 MSG_DT_RELASZ_CFNP,             MSG_DT_RELAENT_CFNP,
 500                 MSG_DT_STRSZ_CFNP,              MSG_DT_SYMENT_CFNP,
 501                 MSG_DT_INIT_CFNP,               MSG_DT_FINI_CFNP,
 502                 MSG_DT_SONAME_CFNP,             MSG_DT_RPATH_CFNP,
 503                 MSG_DT_SYMBOLIC_DMP,            MSG_DT_REL_CFNP,
 504                 MSG_DT_RELSZ_CFNP,              MSG_DT_RELENT_CFNP,
 505                 MSG_DT_PLTREL_CFNP,             MSG_DT_DEBUG_CFNP,
 506                 MSG_DT_TEXTREL_CFNP,            MSG_DT_JMPREL_CFNP,
 507                 MSG_DT_BIND_NOW_CFNP,           MSG_DT_INIT_ARRAY_CFNP,
 508                 MSG_DT_FINI_ARRAY_CFNP,         MSG_DT_INIT_ARRAYSZ_CFNP,
 509                 MSG_DT_FINI_ARRAYSZ_CFNP,       MSG_DT_RUNPATH_CFNP,
 510                 MSG_DT_FLAGS_CFNP,              0,
 511                 MSG_DT_PREINIT_ARRAY_CFNP,      MSG_DT_PREINIT_ARRAYSZ_CFNP
 512         };
 513         static const conv_ds_msg_t ds_null_cf = {
 514             CONV_DS_MSG_INIT(DT_NULL, tags_null_cf) };
 515         static const conv_ds_msg_t ds_null_cfnp = {
 516             CONV_DS_MSG_INIT(DT_NULL, tags_null_cfnp) };
 517         static const conv_ds_msg_t ds_null_nf = {
 518             CONV_DS_MSG_INIT(DT_NULL, tags_null_nf) };
 519         static const conv_ds_msg_t ds_null_dmp = {
 520             CONV_DS_MSG_INIT(DT_NULL, tags_null_dmp) };

 522         /*
 523          * DT_SPARC_REGISTER was originally assigned 0x7000001. It is processor
 524          * specific, and should have been in the range DT_LOPROC-DT_HIPROC
 525          * instead of here. When the error was fixed,
 526          * DT_DEPRECATED_SPARC_REGISTER was created to maintain backward
 527          * compatability.
 528          */
 529         static const Msg        tags_sdreg_cf[] = {
 530             MSG_DT_DEP_SPARC_REG_CF };
 531         static const Msg        tags_sdreg_cfnp[] = {
 532             MSG_DT_DEP_SPARC_REG_CFNP };
 533         static const Msg        tags_sdreg_nf[] = {
 534             MSG_DT_DEP_SPARC_REG_NF };

 536         static const conv_ds_msg_t ds_sdreg_cf = {
 537             CONV_DS_MSG_INIT(DT_DEPRECATED_SPARC_REGISTER, tags_sdreg_cf) };
 538         static const conv_ds_msg_t ds_sdreg_cfnp = {
 539             CONV_DS_MSG_INIT(DT_DEPRECATED_SPARC_REGISTER, tags_sdreg_cfnp) };
 540         static const conv_ds_msg_t ds_sdreg_nf = {
 541             CONV_DS_MSG_INIT(DT_DEPRECATED_SPARC_REGISTER, tags_sdreg_nf) };
```

```
544         /*
545          * SUNW: DT_LOOS -> DT_HIOS range. Note holes between DT_SUNW_TLSSORTSZ,
546          * DT_SUNW_STRPAD, and DT_SUNW_LDMACH. We handle the outliers
547          * separately below as single values.
548          */
549         static const Msg        tags_sunw_auxiliary_cf[] = {
550                 MSG_DT_SUNW_AUXILIARY_CF,       MSG_DT_SUNW_RTLDINF_CF,
551                 MSG_DT_SUNW_FILTER_CF,          MSG_DT_SUNW_CAP_CF,
552                 MSG_DT_SUNW_SYMTAB_CF,          MSG_DT_SUNW_SYMSZ_CF,
553                 MSG_DT_SUNW_SORTENT_CF,         MSG_DT_SUNW_SYMSORT_CF,
554                 MSG_DT_SUNW_SYMSORTSZ_CF,       MSG_DT_SUNW_TLSSORT_CF,
555                 MSG_DT_SUNW_TLSSORTSZ_CF,       MSG_DT_SUNW_CAPINFO_CF,
556                 MSG_DT_SUNW_STRPAD_CF,          MSG_DT_SUNW_CAPCHAIN_CF,
557                 MSG_DT_SUNW_LDMACH_CF,          0,
558                 MSG_DT_SUNW_CAPCHAINENT_CF,     0,
559                 MSG_DT_SUNW_CAPCHAINSZ_CF,      0,
560                 0,                              0,
561                 MSG_DT_SUNW_ASLR_CF,            0,
562                 0,                              0,
563                 MSG_DT_SUNW_KMOD_CF
561                 MSG_DT_SUNW_ASLR_CF
564         };
565         static const Msg        tags_sunw_auxiliary_cfnp[] = {
566                 MSG_DT_SUNW_AUXILIARY_CFNP,     MSG_DT_SUNW_RTLDINF_CFNP,
567                 MSG_DT_SUNW_FILTER_CFNP,        MSG_DT_SUNW_CAP_CFNP,
568                 MSG_DT_SUNW_SYMTAB_CFNP,        MSG_DT_SUNW_SYMSZ_CFNP,
569                 MSG_DT_SUNW_SORTENT_CFNP,       MSG_DT_SUNW_SYMSORT_CFNP,
570                 MSG_DT_SUNW_SYMSORTSZ_CFNP,     MSG_DT_SUNW_TLSSORT_CFNP,
571                 MSG_DT_SUNW_TLSSORTSZ_CFNP,     MSG_DT_SUNW_CAPINFO_CFNP,
572                 MSG_DT_SUNW_STRPAD_CFNP,        MSG_DT_SUNW_CAPCHAIN_CFNP,
573                 MSG_DT_SUNW_LDMACH_CFNP,        0,
574                 MSG_DT_SUNW_CAPCHAINENT_CFNP,   0,
575                 MSG_DT_SUNW_CAPCHAINSZ_CFNP,    0,
576                 0,                              0,
577                 MSG_DT_SUNW_ASLR_CFNP,          0,
578                 0,                              0,
579                 MSG_DT_SUNW_KMOD_CFNP
575                 MSG_DT_SUNW_ASLR_CFNP
580         };
581         static const Msg        tags_sunw_auxiliary_nf[] = {
582                 MSG_DT_SUNW_AUXILIARY_NF,       MSG_DT_SUNW_RTLDINF_NF,
583                 MSG_DT_SUNW_FILTER_NF,          MSG_DT_SUNW_CAP_NF,
584                 MSG_DT_SUNW_SYMTAB_NF,          MSG_DT_SUNW_SYMSZ_NF,
585                 MSG_DT_SUNW_SORTENT_NF,         MSG_DT_SUNW_SYMSORT_NF,
586                 MSG_DT_SUNW_SYMSORTSZ_NF,       MSG_DT_SUNW_TLSSORT_NF,
587                 MSG_DT_SUNW_TLSSORTSZ_NF,       MSG_DT_SUNW_CAPINFO_NF,
588                 MSG_DT_SUNW_STRPAD_NF,          MSG_DT_SUNW_CAPCHAIN_NF,
589                 MSG_DT_SUNW_LDMACH_NF,          0,
590                 MSG_DT_SUNW_CAPCHAINENT_NF,     0,
591                 MSG_DT_SUNW_CAPCHAINSZ_NF,      0,
592                 0,                              0,
593                 MSG_DT_SUNW_ASLR_NF,            0,
594                 0,                              0,
595                 MSG_DT_SUNW_KMOD_NF
589                 MSG_DT_SUNW_ASLR_NF
596         };
597         static const conv_ds_msg_t ds_sunw_auxiliary_cf = {
598             CONV_DS_MSG_INIT(DT_SUNW_AUXILIARY, tags_sunw_auxiliary_cf) };
599         static const conv_ds_msg_t ds_sunw_auxiliary_cfnp = {
600             CONV_DS_MSG_INIT(DT_SUNW_AUXILIARY, tags_sunw_auxiliary_cfnp) };
601         static const conv_ds_msg_t ds_sunw_auxiliary_nf = {
602             CONV_DS_MSG_INIT(DT_SUNW_AUXILIARY, tags_sunw_auxiliary_nf) };

604         /*
605          * GNU: (In DT_VALRNGLO section) DT_GNU_PRELINKED - DT_GNU_LIBLISTSZ
606          */
```



```
607         static const Msg        tags_gnu_prelinked_cf[] = {
608                 MSG_DT_GNU_PRELINKED_CF,        MSG_DT_GNU_CONFLICTSZ_CF,
609                 MSG_DT_GNU_LIBLISTSZ_CF
610         };
611         static const Msg        tags_gnu_prelinked_cfnp[] = {
612                 MSG_DT_GNU_PRELINKED_CFNP,      MSG_DT_GNU_CONFLICTSZ_CFNP,
613                 MSG_DT_GNU_LIBLISTSZ_CFNP
614         };
615         static const Msg        tags_gnu_prelinked_nf[] = {
616                 MSG_DT_GNU_PRELINKED_NF,        MSG_DT_GNU_CONFLICTSZ_NF,
617                 MSG_DT_GNU_LIBLISTSZ_NF
618         };
619         static const conv_ds_msg_t ds_gnu_prelinked_cf = {
620             CONV_DS_MSG_INIT(DT_GNU_PRELINKED, tags_gnu_prelinked_cf) };
621         static const conv_ds_msg_t ds_gnu_prelinked_cfnp = {
622             CONV_DS_MSG_INIT(DT_GNU_PRELINKED, tags_gnu_prelinked_cfnp) };
623         static const conv_ds_msg_t ds_gnu_prelinked_nf = {
624             CONV_DS_MSG_INIT(DT_GNU_PRELINKED, tags_gnu_prelinked_nf) };

626         /*
627          * SUNW: DT_VALRNGLO - DT_VALRNGHI range.
628          */
629         static const Msg        tags_checksum_cf[] = {
630                 MSG_DT_CHECKSUM_CF,             MSG_DT_PLTPADSZ_CF,
631                 MSG_DT_MOVEENT_CF,              MSG_DT_MOVESZ_CF,
632                 MSG_DT_FEATURE_1_CF,            MSG_DT_POSFLAG_1_CF,
633                 MSG_DT_SYMINSZ_CF,              MSG_DT_SYMINENT_CF
634         };
635         static const Msg        tags_checksum_cfnp[] = {
636                 MSG_DT_CHECKSUM_CFNP,           MSG_DT_PLTPADSZ_CFNP,
637                 MSG_DT_MOVEENT_CFNP,            MSG_DT_MOVESZ_CFNP,
638                 MSG_DT_FEATURE_1_CFNP,          MSG_DT_POSFLAG_1_CFNP,
639                 MSG_DT_SYMINSZ_CFNP,            MSG_DT_SYMINENT_CFNP
640         };
641         static const Msg        tags_checksum_nf[] = {
642                 MSG_DT_CHECKSUM_NF,             MSG_DT_PLTPADSZ_NF,
643                 MSG_DT_MOVEENT_NF,              MSG_DT_MOVESZ_NF,
644                 MSG_DT_FEATURE_1_NF,            MSG_DT_POSFLAG_1_NF,
645                 MSG_DT_SYMINSZ_NF,              MSG_DT_SYMINENT_NF
646         };
647         static const conv_ds_msg_t ds_checksum_cf = {
648             CONV_DS_MSG_INIT(DT_CHECKSUM, tags_checksum_cf) };
649         static const conv_ds_msg_t ds_checksum_cfnp = {
650             CONV_DS_MSG_INIT(DT_CHECKSUM, tags_checksum_cfnp) };
651         static const conv_ds_msg_t ds_checksum_nf = {
652             CONV_DS_MSG_INIT(DT_CHECKSUM, tags_checksum_nf) };

654         /*
655          * GNU: (In DT_ADDRRNGLO section) DT_GNU_HASH - DT_GNU_LIBLIST
656          */
657         static const Msg        tags_gnu_hash_cf[] = {
658                 MSG_DT_GNU_HASH_CF,             MSG_DT_TLSDESC_PLT_CF,
659                 MSG_DT_TLSDESC_GOT_CF,          MSG_DT_GNU_CONFLICT_CF,
660                 MSG_DT_GNU_LIBLIST_CF
661         };
662         static const Msg        tags_gnu_hash_cfnp[] = {
663                 MSG_DT_GNU_HASH_CFNP,           MSG_DT_TLSDESC_PLT_CFNP,
664                 MSG_DT_TLSDESC_GOT_CFNP,        MSG_DT_GNU_CONFLICT_CFNP,
665                 MSG_DT_GNU_LIBLIST_CFNP
666         };
667         static const Msg        tags_gnu_hash_nf[] = {
668                 MSG_DT_GNU_HASH_NF,             MSG_DT_TLSDESC_PLT_NF,
669                 MSG_DT_TLSDESC_GOT_NF,          MSG_DT_GNU_CONFLICT_NF,
670                 MSG_DT_GNU_LIBLIST_NF
671         };
672         static const conv_ds_msg_t ds_gnu_hash_cf = {
```

```
673             CONV_DS_MSG_INIT(DT_GNU_HASH, tags_gnu_hash_cf) };
674         static const conv_ds_msg_t ds_gnu_hash_cfnp = {
675             CONV_DS_MSG_INIT(DT_GNU_HASH, tags_gnu_hash_cfnp) };
676         static const conv_ds_msg_t ds_gnu_hash_nf = {
677             CONV_DS_MSG_INIT(DT_GNU_HASH, tags_gnu_hash_nf) };

679         /*
680          * SUNW: DT_ADDRRNGLO - DT_ADDRRNGHI range.
681          */
682         static const Msg        tags_config_cf[] = {
683                 MSG_DT_CONFIG_CF,               MSG_DT_DEPAUDIT_CF,
684                 MSG_DT_AUDIT_CF,                MSG_DT_PLTPAD_CF,
685                 MSG_DT_MOVETAB_CF,              MSG_DT_SYMINFO_CF
686         };
687         static const Msg        tags_config_cfnp[] = {
688                 MSG_DT_CONFIG_CFNP,             MSG_DT_DEPAUDIT_CFNP,
689                 MSG_DT_AUDIT_CFNP,              MSG_DT_PLTPAD_CFNP,
690                 MSG_DT_MOVETAB_CFNP,            MSG_DT_SYMINFO_CFNP
691         };
692         static const Msg        tags_config_nf[] = {
693                 MSG_DT_CONFIG_NF,               MSG_DT_DEPAUDIT_NF,
694                 MSG_DT_AUDIT_NF,                MSG_DT_PLTPAD_NF,
695                 MSG_DT_MOVETAB_NF,              MSG_DT_SYMINFO_NF
696         };
697         static const conv_ds_msg_t ds_config_cf = {
698             CONV_DS_MSG_INIT(DT_CONFIG, tags_config_cf) };
699         static const conv_ds_msg_t ds_config_cfnp = {
700             CONV_DS_MSG_INIT(DT_CONFIG, tags_config_cfnp) };
701         static const conv_ds_msg_t ds_config_nf = {
702             CONV_DS_MSG_INIT(DT_CONFIG, tags_config_nf) };

704         /*
705          * SUNW: generic range. Note hole between DT_VERSYM and DT_RELACOUNT.
706          */
707         static const Msg        tags_versym_cf[] = { MSG_DT_VERSYM_CF };
708         static const Msg        tags_versym_cfnp[] = { MSG_DT_VERSYM_CFNP };
709         static const Msg        tags_versym_nf[] = { MSG_DT_VERSYM_NF };
710         static const conv_ds_msg_t ds_versym_cf = {
711             CONV_DS_MSG_INIT(DT_VERSYM, tags_versym_cf) };
712         static const conv_ds_msg_t ds_versym_cfnp = {
713             CONV_DS_MSG_INIT(DT_VERSYM, tags_versym_cfnp) };
714         static const conv_ds_msg_t ds_versym_nf = {
715             CONV_DS_MSG_INIT(DT_VERSYM, tags_versym_nf) };

717         static const Msg        tags_relacount_cf[] = {
718                 MSG_DT_RELACOUNT_CF,            MSG_DT_RELCOUNT_CF,
719                 MSG_DT_FLAGS_1_CF,              MSG_DT_VERDEF_CF,
720                 MSG_DT_VERDEFNUM_CF,            MSG_DT_VERNEED_CF,
721                 MSG_DT_VERNEEDNUM_CF
722         };
723         static const Msg        tags_relacount_cfnp[] = {
724                 MSG_DT_RELACOUNT_CFNP,          MSG_DT_RELCOUNT_CFNP,
725                 MSG_DT_FLAGS_1_CFNP,            MSG_DT_VERDEF_CFNP,
726                 MSG_DT_VERDEFNUM_CFNP,          MSG_DT_VERNEED_CFNP,
727                 MSG_DT_VERNEEDNUM_CFNP
728         };
729         static const Msg        tags_relacount_nf[] = {
730                 MSG_DT_RELACOUNT_NF,            MSG_DT_RELCOUNT_NF,
731                 MSG_DT_FLAGS_1_NF,              MSG_DT_VERDEF_NF,
732                 MSG_DT_VERDEFNUM_NF,            MSG_DT_VERNEED_NF,
733                 MSG_DT_VERNEEDNUM_NF
734         };
735         static const conv_ds_msg_t ds_relacount_cf = {
736             CONV_DS_MSG_INIT(DT_RELACOUNT, tags_relacount_cf) };
737         static const conv_ds_msg_t ds_relacount_cfnp = {
738             CONV_DS_MSG_INIT(DT_RELACOUNT, tags_relacount_cfnp) };
```



```
739         static const conv_ds_msg_t ds_relacount_nf = {
740             CONV_DS_MSG_INIT(DT_RELACOUNT, tags_relacount_nf) };

742         /*
743          * DT_LOPROC - DT_HIPROC range: solaris/sparc-only
744          */
745         static const Msg tags_sparc_reg_cf[] = { MSG_DT_SPARC_REGISTER_CF };
746         static const Msg tags_sparc_reg_cfnp[] = { MSG_DT_SPARC_REGISTER_CFNP };
747         static const Msg tags_sparc_reg_nf[] = { MSG_DT_SPARC_REGISTER_NF };
748         static const Msg tags_sparc_reg_dmp[] = { MSG_DT_SPARC_REGISTER_DMP };
749         static const conv_ds_msg_t ds_sparc_reg_cf = {
750             CONV_DS_MSG_INIT(DT_SPARC_REGISTER, tags_sparc_reg_cf) };
751         static const conv_ds_msg_t ds_sparc_reg_cfnp = {
752             CONV_DS_MSG_INIT(DT_SPARC_REGISTER, tags_sparc_reg_cfnp) };
753         static const conv_ds_msg_t ds_sparc_reg_nf = {
754             CONV_DS_MSG_INIT(DT_SPARC_REGISTER, tags_sparc_reg_nf) };
755         static const conv_ds_msg_t ds_sparc_reg_dmp = {
756             CONV_DS_MSG_INIT(DT_SPARC_REGISTER, tags_sparc_reg_dmp) };

758         /*
759          * DT_LOPROC - DT_HIPROC range: Solaris osabi, all hardware
760          */
761         static const Msg        tags_auxiliary_cf[] = {
762                 MSG_DT_AUXILIARY_CF,    MSG_DT_USED_CF,
763                 MSG_DT_FILTER_CF
764         };
765         static const Msg        tags_auxiliary_cfnp[] = {
766                 MSG_DT_AUXILIARY_CFNP,  MSG_DT_USED_CFNP,
767                 MSG_DT_FILTER_CFNP
768         };
769         static const Msg        tags_auxiliary_nf[] = {
770                 MSG_DT_AUXILIARY_NF,    MSG_DT_USED_NF,
771                 MSG_DT_FILTER_NF
772         };
773         static const conv_ds_msg_t ds_auxiliary_cf = {
774             CONV_DS_MSG_INIT(DT_AUXILIARY, tags_auxiliary_cf) };
775         static const conv_ds_msg_t ds_auxiliary_cfnp = {
776             CONV_DS_MSG_INIT(DT_AUXILIARY, tags_auxiliary_cfnp) };
777         static const conv_ds_msg_t ds_auxiliary_nf = {
778             CONV_DS_MSG_INIT(DT_AUXILIARY, tags_auxiliary_nf) };


781         static const conv_ds_t  *retarr[MAX_RET];

783         int     ndx = 0;
784         int     fmt_osabi = CONV_TYPE_FMT_ALT(fmt_flags);
785         int     mach_sparc, osabi_solaris, osabi_linux;



789         osabi_solaris = (osabi == ELFOSABI_NONE) ||
790             (osabi == ELFOSABI_SOLARIS) || (osabi == CONV_OSABI_ALL);
791         osabi_linux = (osabi == ELFOSABI_LINUX) || (osabi == CONV_OSABI_ALL);
792         mach_sparc = (mach == EM_SPARC) || (mach == EM_SPARCV9) ||
793             (mach == EM_SPARC32PLUS) || (mach == CONV_MACH_ALL);

795         /*
796          * Fill in retarr with the descriptors for the messages that
797          * apply to the current osabi. Note that we order these items such
798          * that the more common are placed at the beginning, and the less
799          * likely at the end. This should speed the common case.
800          *
801          * Note that the CFNP and DMP styles are very similar, so they
802          * are combined in 'default', and fmt_osabi is consulted when there
803          * are differences.
804          */
```

```
805         switch (fmt_osabi) {
806         case CONV_FMT_ALT_CF:
807                 retarr[ndx++] = CONV_DS_ADDR(ds_null_cf);
808                 if (osabi_solaris)
809                         retarr[ndx++] = CONV_DS_ADDR(ds_sunw_auxiliary_cf);
810                 retarr[ndx++] = CONV_DS_ADDR(ds_checksum_cf);
811                 retarr[ndx++] = CONV_DS_ADDR(ds_config_cf);
812                 retarr[ndx++] = CONV_DS_ADDR(ds_versym_cf);
813                 retarr[ndx++] = CONV_DS_ADDR(ds_relacount_cf);
814                 if (osabi_solaris) {
815                         retarr[ndx++] = CONV_DS_ADDR(ds_auxiliary_cf);
816                         if (mach_sparc) {
817                                 retarr[ndx++] = CONV_DS_ADDR(ds_sparc_reg_cf);
818                                 retarr[ndx++] = CONV_DS_ADDR(ds_sdreg_cf);
819                         }
820                 }
821                 if (osabi_linux) {
822                         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_prelinked_cf);
823                         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_hash_cf);
824                 }
825                 break;

827         case CONV_FMT_ALT_NF:
828                 retarr[ndx++] = CONV_DS_ADDR(ds_null_nf);
829                 if (osabi_solaris)
830                         retarr[ndx++] = CONV_DS_ADDR(ds_sunw_auxiliary_nf);
831                 retarr[ndx++] = CONV_DS_ADDR(ds_checksum_nf);
832                 retarr[ndx++] = CONV_DS_ADDR(ds_config_nf);
833                 retarr[ndx++] = CONV_DS_ADDR(ds_versym_nf);
834                 retarr[ndx++] = CONV_DS_ADDR(ds_relacount_nf);
835                 if (osabi_solaris) {
836                         retarr[ndx++] = CONV_DS_ADDR(ds_auxiliary_nf);
837                         if (mach_sparc) {
838                                 retarr[ndx++] = CONV_DS_ADDR(ds_sparc_reg_nf);
839                                 retarr[ndx++] = CONV_DS_ADDR(ds_sdreg_nf);
840                         }
841                 }
842                 if (osabi_linux) {
843                         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_prelinked_nf);
844                         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_hash_nf);
845                 }
846                 break;
847         default:
848                 /*
849                  * The default style for the generic range is CFNP,
850                  * while dump has a couple of different strings.
851                  */

853                 retarr[ndx++] = (fmt_osabi == CONV_FMT_ALT_DUMP) ?
854                     CONV_DS_ADDR(ds_null_dmp) : CONV_DS_ADDR(ds_null_cfnp);
855                 if (osabi_solaris)
856                         retarr[ndx++] = CONV_DS_ADDR(ds_sunw_auxiliary_cfnp);
857                 retarr[ndx++] = CONV_DS_ADDR(ds_checksum_cfnp);
858                 retarr[ndx++] = CONV_DS_ADDR(ds_config_cfnp);
859                 retarr[ndx++] = CONV_DS_ADDR(ds_versym_cfnp);
860                 retarr[ndx++] = CONV_DS_ADDR(ds_relacount_cfnp);
861                 if (osabi_solaris) {
862                         retarr[ndx++] = CONV_DS_ADDR(ds_auxiliary_cfnp);
863                         if (mach_sparc) {
864                                 /*
865                                  * The default style for DT_SPARC_REGISTER
866                                  * is the dump style, which omits the 'SPARC_'.
867                                  * CFNP keeps the prefix.
868                                  */
869                                 retarr[ndx++] =
870                                     (fmt_osabi == CONV_FMT_ALT_CFNP) ?
```



```
871                                         CONV_DS_ADDR(ds_sparc_reg_cfnp) :
872                                         CONV_DS_ADDR(ds_sparc_reg_dmp);
873                                 retarr[ndx++] = CONV_DS_ADDR(ds_sdreg_cfnp);
874                         }
875                 }
876                 if (osabi_linux) {
877                         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_prelinked_cfnp);
878                         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_hash_cfnp);
879                 }
880                 break;
881         }

883         retarr[ndx++] = NULL;
884         assert(ndx <= MAX_RET);
885         return (retarr);
886 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   16695 Sun Feb 24 19:19:08 2019
new/usr/src/cmd/sgs/libconv/common/dynamic.msg
ld: implement -ztype and rework option parsing
**********************************************************
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright (c) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
  24 #

  26 @ MSG_DT_NULL_CF               "DT_NULL"                       # 0
  27 @ MSG_DT_NULL_CFNP                     "NULL"
  28 @ MSG_DT_NULL_NF                      "null"
  29 @ MSG_DT_NEEDED_CF             "DT_NEEDED"                     # 1
  30 @ MSG_DT_NEEDED_CFNP                 "NEEDED"
  31 @ MSG_DT_NEEDED_NF                  "needed"
  32 @ MSG_DT_PLTRELSZ_CF           "DT_PLTRELSZ"                   # 2
  33 @ MSG_DT_PLTRELSZ_CFNP               "PLTRELSZ"
  34 @ MSG_DT_PLTRELSZ_NF                "pltrelsz"
  35 @ MSG_DT_PLTRELSZ_DMP               "PLTSZ"
  36 @ MSG_DT_PLTGOT_CF             "DT_PLTGOT"                     # 3
  37 @ MSG_DT_PLTGOT_CFNP                 "PLTGOT"
  38 @ MSG_DT_PLTGOT_NF                  "pltgot"
  39 @ MSG_DT_HASH_CF               "DT_HASH"                       # 4
  40 @ MSG_DT_HASH_CFNP                   "HASH"
  41 @ MSG_DT_HASH_NF                    "hash"
  42 @ MSG_DT_STRTAB_CF             "DT_STRTAB"                     # 5
  43 @ MSG_DT_STRTAB_CFNP                 "STRTAB"
  44 @ MSG_DT_STRTAB_NF                  "strtab"
  45 @ MSG_DT_SYMTAB_CF             "DT_SYMTAB"                     # 6
  46 @ MSG_DT_SYMTAB_CFNP                 "SYMTAB"
  47 @ MSG_DT_SYMTAB_NF                  "symtab"
  48 @ MSG_DT_RELA_CF               "DT_RELA"                       # 7
  49 @ MSG_DT_RELA_CFNP                   "RELA"
  50 @ MSG_DT_RELA_NF                    "rela"
  51 @ MSG_DT_RELASZ_CF             "DT_RELASZ"                     # 8
  52 @ MSG_DT_RELASZ_CFNP                 "RELASZ"
  53 @ MSG_DT_RELASZ_NF                  "relasz"
  54 @ MSG_DT_RELAENT_CF            "DT_RELAENT"                    # 9
  55 @ MSG_DT_RELAENT_CFNP                "RELAENT"
  56 @ MSG_DT_RELAENT_NF                 "relaent"
  57 @ MSG_DT_STRSZ_CF              "DT_STRSZ"                      # 10
  58 @ MSG_DT_STRSZ_CFNP                  "STRSZ"
  59 @ MSG_DT_STRSZ_NF                   "strsz"
  60 @ MSG_DT_SYMENT_CF             "DT_SYMENT"                     # 11
  61 @ MSG_DT_SYMENT_CFNP                 "SYMENT"
```



```
  62 @ MSG_DT_SYMENT_NF                   "syment"
  63 @ MSG_DT_INIT_CF               "DT_INIT"                       # 12
  64 @ MSG_DT_INIT_CFNP                   "INIT"
  65 @ MSG_DT_INIT_NF                    "init"
  66 @ MSG_DT_FINI_CF               "DT_FINI"                       # 13
  67 @ MSG_DT_FINI_CFNP                   "FINI"
  68 @ MSG_DT_FINI_NF                    "fini"
  69 @ MSG_DT_SONAME_CF             "DT_SONAME"                     # 14
  70 @ MSG_DT_SONAME_CFNP                 "SONAME"
  71 @ MSG_DT_SONAME_NF                  "soname"
  72 @ MSG_DT_RPATH_CF              "DT_RPATH"                      # 15
  73 @ MSG_DT_RPATH_CFNP                  "RPATH"
  74 @ MSG_DT_RPATH_NF                   "rpath"
  75 @ MSG_DT_SYMBOLIC_CF           "DT_SYMBOLIC"                   # 16
  76 @ MSG_DT_SYMBOLIC_CFNP               "SYMBOLIC"
  77 @ MSG_DT_SYMBOLIC_NF                "symbolic"
  78 @ MSG_DT_SYMBOLIC_DMP               "SYMB"
  79 @ MSG_DT_REL_CF                "DT_REL"                        # 17
  80 @ MSG_DT_REL_CFNP                    "REL"
  81 @ MSG_DT_REL_NF                     "rel"
  82 @ MSG_DT_RELSZ_CF              "DT_RELSZ"                      # 18
  83 @ MSG_DT_RELSZ_CFNP                  "RELSZ"
  84 @ MSG_DT_RELSZ_NF                   "relsz"
  85 @ MSG_DT_RELENT_CF             "DT_RELENT"                     # 19
  86 @ MSG_DT_RELENT_CFNP                 "RELENT"
  87 @ MSG_DT_RELENT_NF                  "relent"
  88 @ MSG_DT_PLTREL_CF             "DT_PLTREL"                     # 20
  89 @ MSG_DT_PLTREL_CFNP                 "PLTREL"
  90 @ MSG_DT_PLTREL_NF                  "pltrel"
  91 @ MSG_DT_DEBUG_CF              "DT_DEBUG"                      # 21
  92 @ MSG_DT_DEBUG_CFNP                  "DEBUG"
  93 @ MSG_DT_DEBUG_NF                   "debug"
  94 @ MSG_DT_TEXTREL_CF            "DT_TEXTREL"                    # 22
  95 @ MSG_DT_TEXTREL_CFNP                "TEXTREL"
  96 @ MSG_DT_TEXTREL_NF                 "textrel"
  97 @ MSG_DT_JMPREL_CF             "DT_JMPREL"                     # 23
  98 @ MSG_DT_JMPREL_CFNP                 "JMPREL"
  99 @ MSG_DT_JMPREL_NF                  "jmprel"
 100 @ MSG_DT_BIND_NOW_CF           "DT_BIND_NOW"                   # 24
 101 @ MSG_DT_BIND_NOW_CFNP              "BIND_NOW"
 102 @ MSG_DT_BIND_NOW_NF               "bind_now"
 103 @ MSG_DT_INIT_ARRAY_CF         "DT_INIT_ARRAY"                 # 25
 104 @ MSG_DT_INIT_ARRAY_CFNP            "INIT_ARRAY"
 105 @ MSG_DT_INIT_ARRAY_NF             "init_array"
 106 @ MSG_DT_FINI_ARRAY_CF         "DT_FINI_ARRAY"                 # 26
 107 @ MSG_DT_FINI_ARRAY_CFNP            "FINI_ARRAY"
 108 @ MSG_DT_FINI_ARRAY_NF             "fini_array"
 109 @ MSG_DT_INIT_ARRAYSZ_CF       "DT_INIT_ARRAYSZ"               # 27
 110 @ MSG_DT_INIT_ARRAYSZ_CFNP          "INIT_ARRAYSZ"
 111 @ MSG_DT_INIT_ARRAYSZ_NF           "init_arraysz"
 112 @ MSG_DT_FINI_ARRAYSZ_CF       "DT_FINI_ARRAYSZ"               # 28
 113 @ MSG_DT_FINI_ARRAYSZ_CFNP          "FINI_ARRAYSZ"
 114 @ MSG_DT_FINI_ARRAYSZ_NF           "fini_arraysz"
 115 @ MSG_DT_RUNPATH_CF            "DT_RUNPATH"                    # 29
 116 @ MSG_DT_RUNPATH_CFNP               "RUNPATH"
 117 @ MSG_DT_RUNPATH_NF                "runpath"
 118 @ MSG_DT_FLAGS_CF              "DT_FLAGS"                      # 30
 119 @ MSG_DT_FLAGS_CFNP                 "FLAGS"
 120 @ MSG_DT_FLAGS_NF                  "flags"
 121 @ MSG_DT_PREINIT_ARRAY_CF      "DT_PREINIT_ARRAY"              # 32
 122 @ MSG_DT_PREINIT_ARRAY_CFNP         "PREINIT_ARRAY"
 123 @ MSG_DT_PREINIT_ARRAY_NF          "preinit_array"
 124 @ MSG_DT_PREINIT_ARRAYSZ_CF    "DT_PREINIT_ARRAYSZ"            # 33
 125 @ MSG_DT_PREINIT_ARRAYSZ_CFNP       "PREINIT_ARRAYSZ"
 126 @ MSG_DT_PREINIT_ARRAYSZ_NF        "preinit_arraysz"
 127 @ MSG_DT_DEP_SPARC_REG_CF    "DT_DEPRECATED_SPARC_REGISTER"  # 0x07000001
```

```
128 @ MSG_DT_DEP_SPARC_REG_CFNP             "DEPRECATED_SPARC_REGISTER"
129 @ MSG_DT_DEP_SPARC_REG_NF               "deprecated_sparc_register"
130 @ MSG_DT_SUNW_AUXILIARY_CF      "DT_SUNW_AUXILIARY"             # 0x6000000d
131 @ MSG_DT_SUNW_AUXILIARY_CFNP             "SUNW_AUXILIARY"
132 @ MSG_DT_SUNW_AUXILIARY_NF               "sunw_auxiliary"
133 @ MSG_DT_SUNW_RTLDINF_CF         "DT_SUNW_RTLDINF"               # 0x6000000e
134 @ MSG_DT_SUNW_RTLDINF_CFNP               "SUNW_RTLDINF"
135 @ MSG_DT_SUNW_RTLDINF_NF                 "sunw_rtldinf"
136 @ MSG_DT_SUNW_FILTER_CF          "DT_SUNW_FILTER"                # 0x6000000f
137 @ MSG_DT_SUNW_FILTER_CFNP                "SUNW_FILTER"
138 @ MSG_DT_SUNW_FILTER_NF                  "sunw_filter"
139 @ MSG_DT_SUNW_CAP_CF             "DT_SUNW_CAP"                   # 0x60000010
140 @ MSG_DT_SUNW_CAP_CFNP                   "SUNW_CAP"
141 @ MSG_DT_SUNW_CAP_NF                     "sunw_cap"
142 @ MSG_DT_SUNW_SYMTAB_CF          "DT_SUNW_SYMTAB"                # 0x60000011
143 @ MSG_DT_SUNW_SYMTAB_CFNP               "SUNW_SYMTAB"
144 @ MSG_DT_SUNW_SYMTAB_NF                 "sunw_symtab"
145 @ MSG_DT_SUNW_SYMSZ_CF           "DT_SUNW_SYMSZ"                 # 0x60000012
146 @ MSG_DT_SUNW_SYMSZ_CFNP                 "SUNW_SYMSZ"
147 @ MSG_DT_SUNW_SYMSZ_NF                   "sunw_symsz"
148 @ MSG_DT_SUNW_SORTENT_CF         "DT_SUNW_SORTENT"               # 0x60000013
149 @ MSG_DT_SUNW_SORTENT_CFNP               "SUNW_SORTENT"
150 @ MSG_DT_SUNW_SORTENT_NF                 "sunw_sortent"
151 @ MSG_DT_SUNW_SYMSORT_CF         "DT_SUNW_SYMSORT"               # 0x60000014
152 @ MSG_DT_SUNW_SYMSORT_CFNP               "SUNW_SYMSORT"
153 @ MSG_DT_SUNW_SYMSORT_NF                 "sunw_symsort"
154 @ MSG_DT_SUNW_SYMSORTSZ_CF       "DT_SUNW_SYMSORTSZ"             # 0x60000015
155 @ MSG_DT_SUNW_SYMSORTSZ_CFNP             "SUNW_SYMSORTSZ"
156 @ MSG_DT_SUNW_SYMSORTSZ_NF               "sunw_symsortsz"
157 @ MSG_DT_SUNW_TLSSORT_CF         "DT_SUNW_TLSSORT"               # 0x60000016
158 @ MSG_DT_SUNW_TLSSORT_CFNP               "SUNW_TLSSORT"
159 @ MSG_DT_SUNW_TLSSORT_NF                 "sunw_tlssort"
160 @ MSG_DT_SUNW_TLSSORTSZ_CF       "DT_SUNW_TLSSORTSZ"             # 0x60000017
161 @ MSG_DT_SUNW_TLSSORTSZ_CFNP             "SUNW_TLSSORTSZ"
162 @ MSG_DT_SUNW_TLSSORTSZ_NF               "sunw_tlssortsz"
163 @ MSG_DT_SUNW_CAPINFO_CF         "DT_SUNW_CAPINFO"               # 0x60000018
164 @ MSG_DT_SUNW_CAPINFO_CFNP               "SUNW_CAPINFO"
165 @ MSG_DT_SUNW_CAPINFO_NF                 "sunw_capinfo"
166 @ MSG_DT_SUNW_STRPAD_CF          "DT_SUNW_STRPAD"                # 0x60000019
167 @ MSG_DT_SUNW_STRPAD_CFNP                "SUNW_STRPAD"
168 @ MSG_DT_SUNW_STRPAD_NF                  "sunw_strpad"
169 @ MSG_DT_SUNW_CAPCHAIN_CF        "DT_SUNW_CAPCHAIN"              # 0x6000001a
170 @ MSG_DT_SUNW_CAPCHAIN_CFNP              "SUNW_CAPCHAIN"
171 @ MSG_DT_SUNW_CAPCHAIN_NF                "sunw_capchain"
172 @ MSG_DT_SUNW_LDMACH_CF          "DT_SUNW_LDMACH"                # 0x6000001b
173 @ MSG_DT_SUNW_LDMACH_CFNP                "SUNW_LDMACH"
174 @ MSG_DT_SUNW_LDMACH_NF                  "sunw_ldmach"
175 @ MSG_DT_SUNW_CAPCHAINENT_CF     "DT_SUNW_CAPCHAINENT"           # 0x6000001d
176 @ MSG_DT_SUNW_CAPCHAINENT_CFNP           "SUNW_CAPCHAINENT"
177 @ MSG_DT_SUNW_CAPCHAINENT_NF             "sunw_capchainent"
178 @ MSG_DT_SUNW_CAPCHAINSZ_CF      "DT_SUNW_CAPCHAINSZ"            # 0x6000001f
178 @ MSG_DT_SUNW_CAPCHAINSZ_CF     "DT_SUNW_CAPCHAINSZ"            # 0x6000001d
179 @ MSG_DT_SUNW_CAPCHAINSZ_CFNP           "SUNW_CAPCHAINSZ"
180 @ MSG_DT_SUNW_CAPCHAINSZ_NF             "sunw_capchainsz"
181 @ MSG_DT_SUNW_ASLR_CF            "DT_SUNW_ASLR"                  # 0x60000023
182 @ MSG_DT_SUNW_ASLR_CFNP                  "SUNW_ASLR"
183 @ MSG_DT_SUNW_ASLR_NF                    "sunw_aslr"
184 @ MSG_DT_SUNW_KMOD_CF            "DT_SUNW_KMOD"                  # 0x60000027
185 @ MSG_DT_SUNW_KMOD_CFNP                  "SUNW_KMOD"
186 @ MSG_DT_SUNW_KMOD_NF                    "sunw_kmod"
187 #endif /* ! codereview */

189 @ MSG_DT_GNU_PRELINKED_CF        "DT_GNU_PRELINKED"              # 0x6ffffdf5
190 @ MSG_DT_GNU_PRELINKED_CFNP              "GNU_PRELINKED"
191 @ MSG_DT_GNU_PRELINKED_NF                "gnu_prelinked"
192 @ MSG_DT_GNU_CONFLICTSZ_CF       "DT_GNU_CONFLICTSZ"             # 0x6ffffdf6
```



```
193 @ MSG_DT_GNU_CONFLICTSZ_CFNP             "GNU_CONFLICTSZ"
194 @ MSG_DT_GNU_CONFLICTSZ_NF               "gnu_conflictsz"
195 @ MSG_DT_GNU_LIBLISTSZ_CF       "DT_GNU_LIBLISTSZ"              # 0x6ffffdf7
196 @ MSG_DT_GNU_LIBLISTSZ_CFNP             "GNU_LIBLISTSZ"
197 @ MSG_DT_GNU_LIBLISTSZ_NF               "gnu_liblistsz"
198 @ MSG_DT_CHECKSUM_CF             "DT_CHECKSUM"                   # 0x6ffffdf8
199 @ MSG_DT_CHECKSUM_CFNP                   "CHECKSUM"
200 @ MSG_DT_CHECKSUM_NF                     "checksum"
201 @ MSG_DT_PLTPADSZ_CF             "DT_PLTPADSZ"                   # 0x6ffffdf9
202 @ MSG_DT_PLTPADSZ_CFNP                   "PLTPADSZ"
203 @ MSG_DT_PLTPADSZ_NF                     "pltpadsz"
204 @ MSG_DT_MOVEENT_CF              "DT_MOVEENT"                    # 0x6ffffdfa
205 @ MSG_DT_MOVEENT_CFNP                   "MOVEENT"
206 @ MSG_DT_MOVEENT_NF                     "moveent"
207 @ MSG_DT_MOVESZ_CF                "DT_MOVESZ"                     # 0x6ffffdfb
208 @ MSG_DT_MOVESZ_CFNP                   "MOVESZ"
209 @ MSG_DT_MOVESZ_NF                     "movesz"
210 @ MSG_DT_FEATURE_1_CF             "DT_FEATURE_1"                  # 0x6ffffdfc
211 @ MSG_DT_FEATURE_1_CFNP                   "FEATURE_1"
212 @ MSG_DT_FEATURE_1_NF                     "feature_1"
213 @ MSG_DT_POSFLAG_1_CF             "DT_POSFLAG_1"                  # 0x6ffffdfd
214 @ MSG_DT_POSFLAG_1_CFNP                   "POSFLAG_1"
215 @ MSG_DT_POSFLAG_1_NF                     "posflag_1"
216 @ MSG_DT_SYMINSZ_CF               "DT_SYMINSZ"                    # 0x6ffffdfe
217 @ MSG_DT_SYMINSZ_CFNP                   "SYMINSZ"
218 @ MSG_DT_SYMINSZ_NF                     "syminsz"
219 @ MSG_DT_SYMINENT_CF              "DT_SYMINENT"                   # 0x6ffffdff
220 @ MSG_DT_SYMINENT_CFNP                   "SYMINENT"
221 @ MSG_DT_SYMINENT_NF                     "syminent"
222 @ MSG_DT_GNU_HASH_CF              "DT_GNU_HASH"                   # 0x6ffffef5
223 @ MSG_DT_GNU_HASH_CFNP                   "GNU_HASH"
224 @ MSG_DT_GNU_HASH_NF                     "gnu_hash"
225 @ MSG_DT_TLSDESC_PLT_CF           "DT_TLSDESC_PLT"                # 0x6ffffef6
226 @ MSG_DT_TLSDESC_PLT_CFNP                 "TLSDESC_PLT"
227 @ MSG_DT_TLSDESC_PLT_NF                   "tlsdesc_plt"
228 @ MSG_DT_TLSDESC_GOT_CF           "DT_TLSDESC_GOT"                # 0x6ffffef7
229 @ MSG_DT_TLSDESC_GOT_CFNP                 "TLSDESC_GOT"
230 @ MSG_DT_TLSDESC_GOT_NF                   "tlsdesc_got"
231 @ MSG_DT_GNU_CONFLICT_CF          "DT_GNU_CONFLICT"               # 0x6ffffef8
232 @ MSG_DT_GNU_CONFLICT_CFNP                "GNU_CONFLICT"
233 @ MSG_DT_GNU_CONFLICT_NF                  "gnu_conflict"
234 @ MSG_DT_GNU_LIBLIST_CF           "DT_GNU_LIBLIST"                # 0x6ffffef9
235 @ MSG_DT_GNU_LIBLIST_CFNP                 "GNU_LIBLIST"
236 @ MSG_DT_GNU_LIBLIST_NF                   "gnu_liblist"
237 @ MSG_DT_CONFIG_CF                "DT_CONFIG"                     # 0x6ffffefa
238 @ MSG_DT_CONFIG_CFNP                     "CONFIG"
239 @ MSG_DT_CONFIG_NF                       "config"
240 @ MSG_DT_DEPAUDIT_CF              "DT_DEPAUDIT"                   # 0x6ffffefb
241 @ MSG_DT_DEPAUDIT_CFNP                   "DEPAUDIT"
242 @ MSG_DT_DEPAUDIT_NF                     "depaudit"
243 @ MSG_DT_AUDIT_CF                 "DT_AUDIT"                      # 0x6ffffefc
244 @ MSG_DT_AUDIT_CFNP                      "AUDIT"
245 @ MSG_DT_AUDIT_NF                        "audit"
246 @ MSG_DT_PLTPAD_CF                "DT_PLTPAD"                     # 0x6ffffefd
247 @ MSG_DT_PLTPAD_CFNP                     "PLTPAD"
248 @ MSG_DT_PLTPAD_NF                       "pltpad"
249 @ MSG_DT_MOVETAB_CF               "DT_MOVETAB"                    # 0x6ffffefe
250 @ MSG_DT_MOVETAB_CFNP                    "MOVETAB"
251 @ MSG_DT_MOVETAB_NF                      "movetab"
252 @ MSG_DT_SYMINFO_CF               "DT_SYMINFO"                    # 0x6ffffeff
253 @ MSG_DT_SYMINFO_CFNP                    "SYMINFO"
254 @ MSG_DT_SYMINFO_NF                      "syminfo"
255 @ MSG_DT_VERSYM_CF                "DT_VERSYM"                     # 0x6fffffff0
256 @ MSG_DT_VERSYM_CFNP                    "VERSYM"
257 @ MSG_DT_VERSYM_NF                      "versym"
258 @ MSG_DT_RELACOUNT_CF             "DT_RELACOUNT"                  # 0x6fffffff9
```

```
259 @ MSG_DT_RELACOUNT_CFNP                "RELACOUNT"
260 @ MSG_DT_RELACOUNT_NF                  "relacount"
261 @ MSG_DT_RELCOUNT_CF          "DT_RELCOUNT"                   # 0x6ffffffa
262 @ MSG_DT_RELCOUNT_CFNP                "RELCOUNT"
263 @ MSG_DT_RELCOUNT_NF                  "relcount"
264 @ MSG_DT_FLAGS_1_CF            "DT_FLAGS_1"                  # 0x6ffffffb
265 @ MSG_DT_FLAGS_1_CFNP                "FLAGS_1"
266 @ MSG_DT_FLAGS_1_NF                  "flags_1"
267 @ MSG_DT_VERDEF_CF             "DT_VERDEF"                   # 0x6ffffffc
268 @ MSG_DT_VERDEF_CFNP                "VERDEF"
269 @ MSG_DT_VERDEF_NF                  "verdef"
270 @ MSG_DT_VERDEFNUM_CF          "DT_VERDEFNUM"                # 0x6ffffffd
271 @ MSG_DT_VERDEFNUM_CFNP              "VERDEFNUM"
272 @ MSG_DT_VERDEFNUM_NF                "verdefnum"
273 @ MSG_DT_VERNEED_CF            "DT_VERNEED"                  # 0x6ffffffe
274 @ MSG_DT_VERNEED_CFNP                "VERNEED"
275 @ MSG_DT_VERNEED_NF                  "verneed"
276 @ MSG_DT_VERNEEDNUM_CF         "DT_VERNEEDNUM"               # 0x6fffffff
277 @ MSG_DT_VERNEEDNUM_CFNP             "VERNEEDNUM"
278 @ MSG_DT_VERNEEDNUM_NF               "verneednum"
279 @ MSG_DT_SPARC_REGISTER_CF     "DT_SPARC_REGISTER"           # 0x70000001
280 @ MSG_DT_SPARC_REGISTER_CFNP           "SPARC_REGISTER"
281 @ MSG_DT_SPARC_REGISTER_NF             "sparc_register"
282 @ MSG_DT_SPARC_REGISTER_DMP            "REGISTER"
283 @ MSG_DT_AUXILIARY_CF          "DT_AUXILIARY"                # 0x7ffffffd
284 @ MSG_DT_AUXILIARY_CFNP              "AUXILIARY"
285 @ MSG_DT_AUXILIARY_NF                "auxiliary"
286 @ MSG_DT_USED_CF               "DT_USED"                     # 0x7ffffffe
287 @ MSG_DT_USED_CFNP                   "USED"
288 @ MSG_DT_USED_NF                     "used"
289 @ MSG_DT_FILTER_CF             "DT_FILTER"                   # 0x7fffffff
290 @ MSG_DT_FILTER_CFNP                 "FILTER"
291 @ MSG_DT_FILTER_NF                   "filter"


294 @ MSG_DF_ORIGIN_CF             "DF_ORIGIN"             # 0x00000001
295 @ MSG_DF_ORIGIN_CFNP                "ORIGIN"
296 @ MSG_DF_ORIGIN_NF                  "origin"
297 @ MSG_DF_SYMBOLIC_CF           "DF_SYMBOLIC"           # 0x00000002
298 @ MSG_DF_SYMBOLIC_CFNP              "SYMBOLIC"
299 @ MSG_DF_SYMBOLIC_NF                "symbolic"
300 @ MSG_DF_TEXTREL_CF             "DF_TEXTREL"            # 0x00000004
301 @ MSG_DF_TEXTREL_CFNP                "TEXTREL"
302 @ MSG_DF_TEXTREL_NF                  "textrel"
303 @ MSG_DF_BIND_NOW_CF            "DF_BIND_NOW"           # 0x00000008
304 @ MSG_DF_BIND_NOW_CFNP               "BIND_NOW"
305 @ MSG_DF_BIND_NOW_NF                 "bind_now"
306 @ MSG_DF_STATIC_TLS_CF          "DF_STATIC_TLS"         # 0x00000010
307 @ MSG_DF_STATIC_TLS_CFNP             "STATIC_TLS"
308 @ MSG_DF_STATIC_TLS_NF               "static_tls"


311 @ MSG_DF_1_NOW_CF               "DF_1_NOW"              # 0x00000001
312 @ MSG_DF_1_NOW_CFNP                  "NOW"
313 @ MSG_DF_1_NOW_NF                    "now"
314 @ MSG_DF_1_GLOBAL_CF            "DF_1_GLOBAL"           # 0x00000002
315 @ MSG_DF_1_GLOBAL_CFNP               "GLOBAL"
316 @ MSG_DF_1_GLOBAL_NF                 "global"
317 @ MSG_DF_1_GROUP_CF              "DF_1_GROUP"            # 0x00000004
318 @ MSG_DF_1_GROUP_CFNP                 "GROUP"
319 @ MSG_DF_1_GROUP_NF                   "group"
320 @ MSG_DF_1_NODELETE_CF           "DF_1_NODELETE"         # 0x00000008
321 @ MSG_DF_1_NODELETE_CFNP              "NODELETE"
322 @ MSG_DF_1_NODELETE_NF                "nodelete"
323 @ MSG_DF_1_LOADFLTR_CF           "DF_1_LOADFLTR"         # 0x00000010
324 @ MSG_DF_1_LOADFLTR_CFNP              "LOADFLTR"
```



```
325 @ MSG_DF_1_LOADFLTR_NF                "loadfltr"
326 @ MSG_DF_1_INITFIRST_CF         "DF_1_INITFIRST"        # 0x00000020
327 @ MSG_DF_1_INITFIRST_CFNP             "INITFIRST"
328 @ MSG_DF_1_INITFIRST_NF               "initfirst"
329 @ MSG_DF_1_NOOPEN_CF             "DF_1_NOOPEN"           # 0x00000040
330 @ MSG_DF_1_NOOPEN_CFNP                "NOOPEN"
331 @ MSG_DF_1_NOOPEN_NF                  "noopen"
332 @ MSG_DF_1_ORIGIN_CF             "DF_1_ORIGIN"           # 0x00000080
333 @ MSG_DF_1_ORIGIN_CFNP                "ORIGIN"
334 @ MSG_DF_1_ORIGIN_NF                  "origin"
335 @ MSG_DF_1_DIRECT_CF             "DF_1_DIRECT"           # 0x00000100
336 @ MSG_DF_1_DIRECT_CFNP                "DIRECT"
337 @ MSG_DF_1_DIRECT_NF                  "direct"
338 @ MSG_DF_1_TRANS_CF              "DF_1_TRANS"            # 0x00000200
339 @ MSG_DF_1_TRANS_CFNP                 "TRANS"
340 @ MSG_DF_1_TRANS_NF                   "trans"
341 @ MSG_DF_1_INTERPOSE_CF          "DF_1_INTERPOSE"        # 0x00000400
342 @ MSG_DF_1_INTERPOSE_CFNP             "INTERPOSE"
343 @ MSG_DF_1_INTERPOSE_NF               "interpose"
344 @ MSG_DF_1_INTERPOSE_DEF              "OBJECT-INTERPOSE"
345 @ MSG_DF_1_NODEFLIB_CF           "DF_1_NODEFLIB"         # 0x00000800
346 @ MSG_DF_1_NODEFLIB_CFNP              "NODEFLIB"
347 @ MSG_DF_1_NODEFLIB_NF                "nodeflib"
348 @ MSG_DF_1_NODUMP_CF             "DF_1_NODUMP"           # 0x00001000
349 @ MSG_DF_1_NODUMP_CFNP                "NODUMP"
350 @ MSG_DF_1_NODUMP_NF                  "nodump"
351 @ MSG_DF_1_CONFALT_CF            "DF_1_CONFALT"          # 0x00002000
352 @ MSG_DF_1_CONFALT_CFNP               "CONFALT"
353 @ MSG_DF_1_CONFALT_NF                 "confalt"
354 @ MSG_DF_1_ENDFILTEE_CF          "DF_1_ENDFILTEE"        # 0x00004000
355 @ MSG_DF_1_ENDFILTEE_CFNP             "ENDFILTEE"
356 @ MSG_DF_1_ENDFILTEE_NF               "endfiltee"
357 @ MSG_DF_1_DISPRELDNE_CF         "DF_1_DISPRELDNE"       # 0x00008000
358 @ MSG_DF_1_DISPRELDNE_CFNP            "DISPRELDNE"
359 @ MSG_DF_1_DISPRELDNE_NF              "dispreldne"
360 @ MSG_DF_1_DISPRELDNE_DEF             "DISPLACE-RELOCS-DONE"
361 @ MSG_DF_1_DISPRELPND_CF         "DF_1_DISPRELPND"       # 0x00010000
362 @ MSG_DF_1_DISPRELPND_CFNP            "DISPRELPND"
363 @ MSG_DF_1_DISPRELPND_NF              "disprelpnd"
364 @ MSG_DF_1_DISPRELPND_DEF             "DISPLACE-RELOCS-PEND"
365 @ MSG_DF_1_NODIRECT_CF           "DF_1_NODIRECT"         # 0x00020000
366 @ MSG_DF_1_NODIRECT_CFNP              "NODIRECT"
367 @ MSG_DF_1_NODIRECT_NF                "nodirect"
368 @ MSG_DF_1_IGNMULDEF_CF          "DF_1_IGNMULDEF"        # 0x00040000
369 @ MSG_DF_1_IGNMULDEF_CFNP             "IGNMULDEF"
370 @ MSG_DF_1_IGNMULDEF_NF               "ignmuldef"
371 @ MSG_DF_1_IGNMULDEF_DEF              "IGNORE-MULDEFS"
372 @ MSG_DF_1_NOKSYMS_CF            "DF_1_NOKSYMS"          # 0x00080000
373 @ MSG_DF_1_NOKSYMS_CFNP               "NOKSYMS"
374 @ MSG_DF_1_NOKSYMS_NF                 "noksyms"
375 @ MSG_DF_1_NOHDR_CF              "DF_1_NOHDR"            # 0x00100000
376 @ MSG_DF_1_NOHDR_CFNP                 "NOHDR"
377 @ MSG_DF_1_NOHDR_NF                   "nohdr"
378 @ MSG_DF_1_EDITED_CF             "DF_1_EDITED"           # 0x00200000
379 @ MSG_DF_1_EDITED_CFNP                "EDITED"
380 @ MSG_DF_1_EDITED_NF                  "edited"
381 @ MSG_DF_1_NORELOC_CF            "DF_1_NORELOC"          # 0x00400000
382 @ MSG_DF_1_NORELOC_CFNP               "NORELOC"
383 @ MSG_DF_1_NORELOC_NF                 "noreloc"
384 @ MSG_DF_1_SYMINTPOSE_CF         "DF_1_SYMINTPOSE"       # 0x00800000
385 @ MSG_DF_1_SYMINTPOSE_CFNP            "SYMINTPOSE"
386 @ MSG_DF_1_SYMINTPOSE_NF              "symintpose"
387 @ MSG_DF_1_SYMINTPOSE_DEF             "SYMBOL-INTERPOSE"
388 @ MSG_DF_1_GLOBAUDIT_CF          "DF_1_GLOBAUDIT"        # 0x01000000
389 @ MSG_DF_1_GLOBAUDIT_CFNP             "GLOBAUDIT"
390 @ MSG_DF_1_GLOBAUDIT_NF               "globaudit"
```

```
391 @ MSG_DF_1_GLOBAUDIT_DEF                "GLOBAL-AUDITING"
392 @ MSG_DF_1_SINGLETON_CF         "DF_1_SINGLETON"        # 0x02000000
393 @ MSG_DF_1_SINGLETON_CFNP               "SINGLETON"
394 @ MSG_DF_1_SINGLETON_NF                 "singleton"
395 @ MSG_DF_1_SINGLETON_DEF                "SINGLETON-EXISTS"


398 @ MSG_DF_P1_LAZYLOAD_CF         "DF_P1_LAZYLOAD"        # 0x00000001
399 @ MSG_DF_P1_LAZYLOAD_CFNP               "LAZYLOAD"
400 @ MSG_DF_P1_LAZYLOAD_NF                 "lazyload"
401 @ MSG_DF_P1_LAZYLOAD_DEF                "LAZY"
402 @ MSG_DF_P1_GROUPPERM_CF         "DF_P1_GROUPPERM"       # 0x00000002
403 @ MSG_DF_P1_GROUPPERM_CFNP               "GROUPPERM"
404 @ MSG_DF_P1_GROUPPERM_NF                "groupperm"
405 @ MSG_DF_P1_GROUPPERM_DEF                "GROUP"
406 @ MSG_DF_P1_DEFERRED_CF         "DF_P1_DEFERRED"        # 0x00000004
407 @ MSG_DF_P1_DEFERRED_CFNP               "DEFERRED"
408 @ MSG_DF_P1_DEFERRED_NF                 "deferred"
409 @ MSG_DF_P1_DEFERRED_DEF                "DEFERRED"


412 @ MSG_DTF_1_PARINIT_CF          "DTF_1_PARINIT"         # 0x00000001
413 @ MSG_DTF_1_PARINIT_CFNP               "PARINIT"
414 @ MSG_DTF_1_PARINIT_NF                 "parinit"
415 @ MSG_DTF_1_CONFEXP_CF          "DTF_1_CONFEXP"         # 0x00000002
416 @ MSG_DTF_1_CONFEXP_CFNP               "CONFEXP"
417 @ MSG_DTF_1_CONFEXP_NF                 "confexp"


420 @ MSG_BND_NEEDED        "NEEDED"
421 @ MSG_BND_REFER         "REFERENCED"
422 @ MSG_BND_FILTER        "FILTER"


425 @ MSG_BND_ADDED         "OBJECTS-ADDED"
426 @ MSG_BND_REEVAL        "OBJECTS-REEVALUATED"
427 @ MSG_BND_DELETED       "OBJECTS-DELETED"
428 @ MSG_BND_ATEXIT        "ATEXIT-PROCESSING"
429 @ MSG_BND_REVISIT       "(revisiting)"

431 @ MSG_STR_EMPTY         ""

433 @ MSG_GBL_ZERO          "0"
```

****
38228 Sun Feb 24 19:19:09 2019
new/usr/src/cmd/sgs/libconv/common/elf.c
ld should reject kernel modules as input
****
_____unchanged_portion_omitted_

```
1182 /*
1183  * A generic means of returning additional information for a rejected file in
1184  * terms of a string. ELFOSABI_SOLARIS is assummed.
1185  */
1186 const char *
1187 conv_reject_desc(Rej_desc * rej, Conv_reject_desc_buf_t *reject_desc_buf,
1188     Half mach)
1189 {
1190         ushort_t        type = rej->rej_type;
1191         uint_t          info = rej->rej_info;

1193         switch (type) {
1194         case SGS_REJ_MACH:
1195                 return (conv_ehdr_mach((Half)info, 0,
1196                     &reject_desc_buf->inv_buf));
1197         case SGS_REJ_CLASS:
1198                 return (conv_ehdr_class((uchar_t)info, 0,
1199                     &reject_desc_buf->inv_buf));
1200         case SGS_REJ_DATA:
1201                 return (conv_ehdr_data((uchar_t)info, 0,
1202                     &reject_desc_buf->inv_buf));
1203         case SGS_REJ_TYPE:
1204                 return (conv_ehdr_type(ELFOSABI_SOLARIS, (Half)info, 0,
1205                     &reject_desc_buf->inv_buf));
1206         case SGS_REJ_BADFLAG:
1207         case SGS_REJ_MISFLAG:
1208         case SGS_REJ_HAL:
1209         case SGS_REJ_US3:
1210                 return (conv_ehdr_flags(mach, (Word)info, 0,
1211                     &reject_desc_buf->flags_buf));
1212         case SGS_REJ_UNKFILE:
1213         case SGS_REJ_ARCHIVE:
1214         case SGS_REJ_KMOD:
1215 #endif /* ! codereview */
1216                 return (NULL);
1217         case SGS_REJ_STR:
1218         case SGS_REJ_HWCAP_1:
1219         case SGS_REJ_SFCAP_1:
1220         case SGS_REJ_HWCAP_2:
1221         case SGS_REJ_MACHCAP:
1222         case SGS_REJ_PLATCAP:
1223                 if (rej->rej_str)
1224                         return ((const char *)rej->rej_str);
1225                 else
1226                         return (MSG_ORIG(MSG_STR_EMPTY));
1227         default:
1228                 return (conv_invalid_val(&reject_desc_buf->inv_buf, info,
1229                     CONV_FMT_DECIMAL));
1230         }
1231 }
```

```
**********************************************************
   23848 Sun Feb 24 19:19:09 2019
new/usr/src/cmd/sgs/libelf/common/gelf.c
ld should reject kernel modules as input
**********************************************************
______unchanged_portion_omitted_

1088 /*
1089  * If the specified object has a dynamic section, and that section
1090  * contains a DT_FLAGS_1 entry, then return the value of that entry.
1091  * Otherwise, return 0.
1092  */
1093 GElf_Xword
1094 _gelf_getdynval(Elf *elf, GElf_Sxword tag)
1094 _gelf_getdyndtflags_1(Elf *elf)
1095 {
1096         Elf_Scn *scn = NULL;
1097         Elf_Data *data;
1098         GElf_Shdr shdr;
1099         GElf_Dyn dyn;
1100         int i, n;

1102         while (scn = elf_nextscn(elf, scn)) {
1103                 if (gelf_getshdr(scn, &shdr) == NULL)
1104                         break;
1105                 if (shdr.sh_type != SHT_DYNAMIC)
1106                         continue;
1107                 if (data = elf_getdata(scn, NULL)) {
1108                         n = shdr.sh_size / shdr.sh_entsize;
1109                         for (i = 0; i < n; i++) {
1110                                 (void) gelf_getdyn(data, i, &dyn);
1111                                 if (dyn.d_tag == tag) {
1111                                 if (dyn.d_tag == DT_FLAGS_1) {
1112                                         return (dyn.d_un.d_val);
1113                                 }
1114                         }
1115                 }
1116                 break;
1117         }
1118         return (0);
1119 }

1121 GElf_Xword
1122 _gelf_getdyndtflags_1(Elf *elf)
1123 {
1124         return (_gelf_getdynval(elf, DT_FLAGS_1));
1125 #endif /* ! codereview */
1126 }
```

```
**********************************************************
    3258 Sun Feb 24 19:19:10 2019
new/usr/src/cmd/sgs/libelf/common/mapfile-vers
ld should reject kernel modules as input
**********************************************************
______unchanged_portion_omitted_

 162 SYMBOL_VERSION SUNWprivate_1.1 {
 163     global:
 164         _elf_execfill;
 165         _elf_getarhdrbase;
 166         _elf_getarsymwordsize;
 167         _elf_getnextoff;
 168         _elf_getxoff;
 169         _elf_outsync;
 170         _elf_sys_encoding;
 171         _elf_swap_wrimage;
 172         _gelf_getdyndtflags_1;
 173         _gelf_getdynval;
 174 #endif /* ! codereview */

 176 $if _ELF32
 177         elf_demangle;
 178 $endif
 179 };
```

```
**********************************************************
   67390 Sun Feb 24 19:19:10 2019
new/usr/src/cmd/sgs/libld/common/args.c
ld: implement -ztype and rework option parsing
**********************************************************
______unchanged_portion_omitted_

  98 static Setstate dflag   = SET_UNKNOWN;
  99 static Setstate zdflag  = SET_UNKNOWN;
 100 static Setstate Qflag   = SET_UNKNOWN;
 101 static Setstate Bdflag  = SET_UNKNOWN;
 102 static Setstate zfwflag = SET_UNKNOWN;

 104 static Boolean  aflag   = FALSE;
 105 static Boolean  bflag   = FALSE;
 106 static Boolean  rflag   = FALSE;
 106 static Boolean  sflag   = FALSE;
 107 static Boolean  zinflag = FALSE;
 108 static Boolean  zlflag  = FALSE;
 109 static Boolean  Bgflag  = FALSE;
 110 static Boolean  Blflag  = FALSE;
 111 static Boolean  Beflag  = FALSE;
 112 static Boolean  Bsflag  = FALSE;
 113 static Boolean  Dflag   = FALSE;
 115 static Boolean  Gflag   = FALSE;
 114 static Boolean  Vflag   = FALSE;

 116 enum output_type {
 117         OT_RELOC,               /* relocatable object */
 118         OT_SHARED,              /* shared object */
 119         OT_EXEC,                /* dynamic executable */
 120         OT_KMOD,                /* kernel module */
 121 };

 123 static enum output_type otype = OT_EXEC;

 125 #endif /* ! codereview */
 126 /*
 127  * ztflag's state is set by pointing it to the matching string:
 128  *      text | textoff | textwarn
 129  */
 130 static const char       *ztflag = NULL;

 132 /*
 133  * Remember the guidance flags that result from the initial -z guidance
 134  * option, so that they can be compared to any that follow. We only want
 135  * to issue a warning when they differ.
 136  */
 137 static ofl_guideflag_t  initial_guidance_flags  = 0;

 139 static uintptr_t process_files_com(Ofl_desc *, int, char **);
 140 static uintptr_t process_flags_com(Ofl_desc *, int, char **, int *);

 142 /*
 143  * Print usage message to stderr - 2 modes, summary message only,
 144  * and full usage message.
 145  */
 146 static void
 147 usage_mesg(Boolean detail)
 148 {
 149         (void) fprintf(stderr, MSG_INTL(MSG_ARG_USAGE),
 150             MSG_ORIG(MSG_STR_OPTIONS));

 152         if (detail == FALSE)
 153                 return;
```



```
 155         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_3));
 156         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_6));
 157         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_A));
 158         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_B));
 159         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CBDR));
 160         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CBDY));
 161         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CBE));
 162         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CBG));
 163         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CBL));
 164         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CBR));
 165         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CBS));
 166         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_C));
 167         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CC));
 168         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_D));
 169         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CD));
 170         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_E));
 171         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_F));
 172         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CF));
 173         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CG));
 174         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_H));
 175         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_I));
 176         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CI));
 177         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_L));
 178         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CL));
 179         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_M));
 180         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CM));
 181         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CN));
 182         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_O));
 183         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_P));
 184         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CP));
 185         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CQ));
 186         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_R));
 187         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CR));
 188         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_S));
 189         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CS));
 190         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_T));
 191         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_U));
 192         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CV));
 193         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_CY));
 194         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZA));
 195         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZAE));
 196         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZAL));
 197         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZADLIB));
 198         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZC));
 199         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZDEF));
 200         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZDFS));
 201         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZDRS));
 202         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZE));
 203         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZFATW));
 204         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZFA));
 205         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZGP));
 206         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZGUIDE));
 207         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZH));
 208         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZIG));
 209         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZINA));
 210         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZINI));
 211         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZINT));
 212         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZLAZY));
 213         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZLD32));
 214         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZLD64));
 215         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZLO));
 216         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZM));
 217         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZNC));
 218         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZNDFS));
 219         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZNDEF));
 220         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZNDEL));
```

```
221         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZNDLO));
222         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZNDU));
223         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZNLD));
224         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZNOW));
225         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZNPA));
226         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZNV));
227         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZO));
228         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZPIA));
229         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZRL));
230         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZRREL));
231         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZRS));
232         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZRSN));
233         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZRSGRP));
234         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZSCAP));
235         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZTARG));
236         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZT));
237         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZTO));
238         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZTW));
239         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZTY));
240 #endif /* ! codereview */
241         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZWRAP));
242         (void) fprintf(stderr, MSG_INTL(MSG_ARG_DETAIL_ZVER));
243 }

245 /*
246  * Rescan the archives seen on the command line in order
247  * to handle circularly dependent archives, stopping when
248  * no further member extraction occurs.
249  *
250  * entry:
251  *      ofl - Output file descriptor
252  *      isgrp - True if this is a an archive group search, False
253  *              to search starting with argv[1] through end_arg_ndx
254  *      end_arg_ndx - Index of final argv element to consider.
255  */
256 static uintptr_t
257 ld_rescan_archives(Ofl_desc *ofl, int isgrp, int end_arg_ndx)
258 {
259         ofl->ofl_flags1 |= FLG_OF1_EXTRACT;

261         while (ofl->ofl_flags1 & FLG_OF1_EXTRACT) {
262                 Aliste          idx;
263                 Ar_desc         *adp;
264                 Word            start_ndx = isgrp ? ofl->ofl_ars_gsndx : 0;
265                 Word            ndx = 0;

267                 ofl->ofl_flags1 &= ~FLG_OF1_EXTRACT;

269                 DBG_CALL(Dbg_file_ar_rescan(ofl->ofl_lml,
270                     isgrp ? ofl->ofl_ars_gsandx : 1, end_arg_ndx));

272                 for (APLIST_TRAVERSE(ofl->ofl_ars, idx, adp)) {
273                         /* If not to starting index yet, skip it */
274                         if (ndx++ < start_ndx)
275                                 continue;

277                         /*
278                          * If this archive was processed with -z allextract,
279                          * then all members have already been extracted.
280                          */
281                         if (adp->ad_elf == NULL)
282                                 continue;

284                         /*
285                          * Reestablish any archive specific command line flags.
286                          */
```



```
287                         ofl->ofl_flags1 &= ~MSK_OF1_ARCHIVE;
288                         ofl->ofl_flags1 |= (adp->ad_flags & MSK_OF1_ARCHIVE);

290                         /*
291                          * Re-process the archive.  Note that a file descriptor
292                          * is unnecessary, as the file is already available in
293                          * memory.
294                          */
295                         if (!ld_process_archive(adp->ad_name, -1, adp, ofl))
296                                 return (S_ERROR);
297                         if (ofl->ofl_flags & FLG_OF_FATAL)
298                                 return (1);
299                 }
300         }

302         return (1);
303 }

305 /*
306  * Checks the command line option flags for consistency.
307  */
308 static uintptr_t
309 check_flags(Ofl_desc * ofl, int argc)
310 {
311         /*
312          * If the user specified -zguidance=noall, then we can safely disable
313          * the entire feature. The purpose of -zguidance=noall is to allow
314          * the user to override guidance specified from a makefile via
315          * the LD_OPTIONS environment variable, and so, we want to behave
316          * in exactly the same manner we would have if no option were present.
317          */
318         if ((ofl->ofl_guideflags & (FLG_OFG_ENABLE | FLG_OFG_NO_ALL)) ==
319             (FLG_OFG_ENABLE | FLG_OFG_NO_ALL))
320                 ofl->ofl_guideflags &= ~FLG_OFG_ENABLE;

322         if (Plibpath && (Llibdir || Ulibdir))
323                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_YP),
324                     Llibdir ? 'L' : 'U');

326         if ((otype == OT_RELOC) || (otype == OT_KMOD)) {
327                 if (otype == OT_RELOC) {
118         if (rflag) {
328                         if (dflag == SET_UNKNOWN)
329                                 dflag = SET_FALSE;
330                         if ((dflag == SET_TRUE) &&
331                             OFL_GUIDANCE(ofl, FLG_OFG_NO_KMOD)) {
332                                 ld_eprintf(ofl, ERR_GUIDANCE,
333                                     MSG_INTL(MSG_GUIDE_KMOD));
334                         }
335                 } else if (otype == OT_KMOD) {
336                         if (dflag != SET_UNKNOWN) {
337                                 ld_eprintf(ofl, ERR_FATAL,
338                                     MSG_INTL(MSG_MARG_INCOMP),
339                                     MSG_INTL(MSG_MARG_TYPE_KMOD),
340                                     MSG_ORIG(MSG_ARG_D));
341                         }

343                         dflag = SET_TRUE;
344                 }

346 #endif /* ! codereview */
347                 /*
348                  * Combining relocations when building a relocatable
349                  * object isn't allowed.  Warn the user, but proceed.
350                  */
351                 if (ofl->ofl_flags & FLG_OF_COMREL) {
```

```
352                         const char *msg;

354                         if (otype == OT_RELOC) {
355                                 msg = MSG_INTL(MSG_MARG_REL);
356                         } else {
357                                 msg = MSG_INTL(MSG_MARG_TYPE_KMOD);
358                         }
121                 if (ofl->ofl_flags & FLG_OF_COMREL)
359                         ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_MARG_INCOMP),
360                             msg,
123                             MSG_INTL(MSG_MARG_REL),
361                             MSG_ORIG(MSG_ARG_ZCOMBRELOC));
362                 }
363 #endif /* ! codereview */
364                 ofl->ofl_flags |= FLG_OF_RELOBJ;

366                 if (otype == OT_KMOD)
367                         ofl->ofl_flags |= FLG_OF_KMOD;
368 #endif /* ! codereview */
369         } else {
370                 /*
371                  * Translating object capabilities to symbol capabilities is
372                  * only meaningful when creating a relocatable object.
373                  */
374                 if (ofl->ofl_flags & FLG_OF_OTOSCAP)
375                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_MARG_ONLY),
376                             MSG_ORIG(MSG_ARG_ZSYMBOLCAP),
377                             MSG_INTL(MSG_MARG_REL));

379                 /*
380                  * If the user hasn't explicitly requested that relocations
381                  * not be combined, combine them by default.
382                  */
383                 if ((ofl->ofl_flags & FLG_OF_NOCOMREL) == 0)
384                         ofl->ofl_flags |= FLG_OF_COMREL;
385         }

387         if (zdflag == SET_TRUE)
388                 ofl->ofl_flags |= FLG_OF_NOUNDEF;

390         if (zinflag)
391                 ofl->ofl_dtflags_1 |= DF_1_INTERPOSE;

393         if (sflag)
394                 ofl->ofl_flags |= FLG_OF_STRIP;

396         if (Qflag == SET_TRUE)
397                 ofl->ofl_flags |= FLG_OF_ADDVERS;

399         if (Blflag)
400                 ofl->ofl_flags |= FLG_OF_AUTOLCL;

402         if (Beflag)
403                 ofl->ofl_flags |= FLG_OF_AUTOELM;

405         if (Blflag && Beflag)
406                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_INCOMP),
407                     MSG_ORIG(MSG_ARG_BELIMINATE), MSG_ORIG(MSG_ARG_BLOCAL));

409         if (ofl->ofl_interp && (ofl->ofl_flags1 & FLG_OF1_NOINTRP))
410                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_INCOMP),
411                     MSG_ORIG(MSG_ARG_CI), MSG_ORIG(MSG_ARG_ZNOINTERP));

413         if ((ofl->ofl_flags1 & (FLG_OF1_NRLXREL | FLG_OF1_RLXREL)) ==
414             (FLG_OF1_NRLXREL | FLG_OF1_RLXREL))
415                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_INCOMP),
```



```
416                     MSG_ORIG(MSG_ARG_ZRELAXRELOC),
417                     MSG_ORIG(MSG_ARG_ZNORELAXRELOC));

419         if (ofl->ofl_filtees && (otype != OT_SHARED))
125         if (ofl->ofl_filtees && !Gflag)
420                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_MARG_ST_ONLYAVL),
421                     ((ofl->ofl_flags & FLG_OF_AUX) ?
422                     MSG_INTL(MSG_MARG_FILTER_AUX) : MSG_INTL(MSG_MARG_FILTER)));

424         if (dflag != SET_FALSE) {
425                 /*
426                  * Set -Bdynamic on by default, setting is rechecked as input
427                  * files are processed.
428                  */
429                 ofl->ofl_flags |=
430                     (FLG_OF_DYNAMIC | FLG_OF_DYNLIBS | FLG_OF_PROCRED);

432                 if (aflag)
433                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_INCOMP),
434                             MSG_ORIG(MSG_ARG_DY), MSG_ORIG(MSG_ARG_A));

436                 if (bflag)
437                         ofl->ofl_flags |= FLG_OF_BFLAG;

439                 if (Bgflag == TRUE) {
440                         if (zdflag == SET_FALSE)
441                                 ld_eprintf(ofl, ERR_FATAL,
442                                     MSG_INTL(MSG_ARG_INCOMP),
443                                     MSG_ORIG(MSG_ARG_BGROUP),
444                                     MSG_ORIG(MSG_ARG_ZNODEF));
445                         ofl->ofl_dtflags_1 |= DF_1_GROUP;
446                         ofl->ofl_flags |= FLG_OF_NOUNDEF;
447                 }

449                 /*
450                  * If the use of default library searching has been suppressed
451                  * but no runpaths have been provided we're going to have a hard
452                  * job running this object.
453                  */
454                 if ((ofl->ofl_dtflags_1 & DF_1_NODEFLIB) && !ofl->ofl_rpath)
455                         ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_ARG_NODEFLIB),
456                             MSG_INTL(MSG_MARG_RPATH));

458                 /*
459                  * By default, text relocation warnings are given when building
460                  * an executable unless the -b flag is specified.  This option
461                  * implies that unclean text can be created, so no warnings are
462                  * generated unless specifically asked for.
463                  */
464                 if ((ztflag == MSG_ORIG(MSG_ARG_ZTEXTOFF)) ||
465                     ((ztflag == NULL) && bflag)) {
466                         ofl->ofl_flags1 |= FLG_OF1_TEXTOFF;
467                         ofl->ofl_guideflags |= FLG_OFG_NO_TEXT;
468                 } else if (ztflag == MSG_ORIG(MSG_ARG_ZTEXT)) {
469                         ofl->ofl_flags |= FLG_OF_PURETXT;
470                         ofl->ofl_guideflags |= FLG_OFG_NO_TEXT;
471                 }

473                 if ((otype == OT_SHARED) || (otype == OT_EXEC)) {
179                 if (Gflag || !rflag) {
474                         /*
475                          * Create a dynamic object.  -Bdirect indicates that all
476                          * references should be bound directly.  This also
477                          * enables lazyloading.  Individual symbols can be
478                          * bound directly (or not) using mapfiles and the
479                          * DIRECT (NODIRECT) qualifier.  With this capability,
```

```
480                          * each syminfo entry is tagged SYMINFO_FLG_DIRECTBIND.
481                          * Prior to this per-symbol direct binding, runtime
482                          * direct binding was controlled via the DF_1_DIRECT
483                          * flag.  This flag affected all references from the
484                          * object.  -Bdirect continues to set this flag, and
485                          * thus provides a means of taking a newly built
486                          * direct binding object back to older systems.
487                          *
488                          * NOTE, any use of per-symbol NODIRECT bindings, or
489                          * -znodirect, will disable the creation of the
490                          * DF_1_DIRECT flag.  Older runtime linkers do not
491                          * have the capability to do per-symbol direct bindings.
492                          */
493                         if (Bdflag == SET_TRUE) {
494                                 ofl->ofl_dtflags_1 |= DF_1_DIRECT;
495                                 ofl->ofl_flags1 |= FLG_OF1_LAZYLD;
496                                 ofl->ofl_guideflags |= FLG_OFG_NO_LAZY;
497                                 ofl->ofl_flags |= FLG_OF_SYMINFO;
498                         }
499
500                         /*
501                          * -Bnodirect disables directly binding to any symbols
502                          * exported from the object being created.  Individual
503                          * references to external objects can still be affected
504                          * by -zdirect or mapfile DIRECT directives.
505                          */
506                         if (Bdflag == SET_FALSE) {
507                                 ofl->ofl_flags1 |= (FLG_OF1_NDIRECT |
508                                     FLG_OF1_NGLBDIR | FLG_OF1_ALNODIR);
509                                 ofl->ofl_flags |= FLG_OF_SYMINFO;
510                         }
511                 }

513                 if (otype == OT_EXEC) {
219                 if (!Gflag && !rflag) {
514                         /*
515                          * Dynamically linked executable.
516                          */
517                         ofl->ofl_flags |= FLG_OF_EXEC;

519                         if (zdflag != SET_FALSE)
520                                 ofl->ofl_flags |= FLG_OF_NOUNDEF;

522                         /*
523                          * -z textwarn is the default for executables, and
524                          * only an explicit -z text* option can change that,
525                          * so there's no need to provide additional guidance.
526                          */
527                         ofl->ofl_guideflags |= FLG_OFG_NO_TEXT;

529                         if (Bsflag)
530                                 ld_eprintf(ofl, ERR_FATAL,
531                                     MSG_INTL(MSG_ARG_DY_INCOMP),
532                                     MSG_ORIG(MSG_ARG_BSYMBOLIC));
533                         if (ofl->ofl_soname)
534                                 ld_eprintf(ofl, ERR_FATAL,
535                                     MSG_INTL(MSG_MARG_DY_INCOMP),
536                                     MSG_INTL(MSG_MARG_SONAME));
537                 } else if (otype == OT_SHARED) {
243                 } else if (!rflag) {
538                         /*
539                          * Shared library.
540                          */
541                         ofl->ofl_flags |= FLG_OF_SHAROBJ;

543                         /*
```



```
544                          * By default, print text relocation warnings for
545                          * executables but *not* for shared objects. However,
546                          * if -z guidance is on, issue warnings for shared
547                          * objects as well.
548                          *
549                          * If -z textwarn is explicitly specified, also issue
550                          * guidance messages if -z guidance is on, but not
551                          * for -z text or -z textoff.
552                          */
553                         if (ztflag == NULL) {
554                                 if (!OFL_GUIDANCE(ofl, FLG_OFG_NO_TEXT))
555                                         ofl->ofl_flags1 |= FLG_OF1_TEXTOFF;
556                         } else if ((ofl->ofl_flags & FLG_OF_PURETXT) ||
557                             (ofl->ofl_flags1 & FLG_OF1_TEXTOFF)) {
558                                 ofl->ofl_guideflags |= FLG_OFG_NO_TEXT;
559                         }

561                         if (Bsflag) {
562                                 /*
563                                  * -Bsymbolic, and -Bnodirect make no sense.
564                                  */
565                                 if (Bdflag == SET_FALSE)
566                                         ld_eprintf(ofl, ERR_FATAL,
567                                             MSG_INTL(MSG_ARG_INCOMP),
568                                             MSG_ORIG(MSG_ARG_BSYMBOLIC),
569                                             MSG_ORIG(MSG_ARG_BNODIRECT));
570                                 ofl->ofl_flags |= FLG_OF_SYMBOLIC;
571                                 ofl->ofl_dtflags |= DF_SYMBOLIC;
572                         }
573                 } else {
574                         /*
575                          * Dynamic relocatable object.
576                          */
577                         if (ztflag == NULL)
578                                 ofl->ofl_flags1 |= FLG_OF1_TEXTOFF;
579                         ofl->ofl_guideflags |= FLG_OFG_NO_TEXT;

581                         if (ofl->ofl_interp)
582                                 ld_eprintf(ofl, ERR_FATAL,
583                                     MSG_INTL(MSG_MARG_INCOMP),
584                                     MSG_INTL(MSG_MARG_REL),
585                                     MSG_ORIG(MSG_ARG_CI));
586                 }

588                 assert((ofl->ofl_flags & (FLG_OF_SHAROBJ|FLG_OF_EXEC)) !=
589                     (FLG_OF_SHAROBJ|FLG_OF_EXEC));
590 #endif /* ! codereview */
591                 } else {
592                         ofl->ofl_flags |= FLG_OF_STATIC;

594                         if (bflag)
595                                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_ST_INCOMP),
596                                     MSG_ORIG(MSG_ARG_B));
597                         if (ofl->ofl_soname)
598                                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_MARG_ST_INCOMP),
599                                     MSG_INTL(MSG_MARG_SONAME));
600                         if (ofl->ofl_depaudit)
601                                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_ST_INCOMP),
602                                     MSG_ORIG(MSG_ARG_CP));
603                         if (ofl->ofl_audit)
604                                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_ST_INCOMP),
605                                     MSG_ORIG(MSG_ARG_P));
606                         if (ofl->ofl_config)
607                                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_ST_INCOMP),
608                                     MSG_ORIG(MSG_ARG_C));
609                         if (ztflag)
```

```
610                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_ST_INCOMP),
611                             MSG_ORIG(MSG_ARG_ZTEXTALL));
612                 if (otype == OT_SHARED)
293                 if (Gflag)
613                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_MARG_ST_INCOMP),
614                             MSG_INTL(MSG_MARG_SO));
615                 if (aflag && (otype == OT_RELOC))
296                 if (aflag && rflag)
616                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_MARG_INCOMP),
617                             MSG_ORIG(MSG_ARG_A), MSG_INTL(MSG_MARG_REL));

619                 if (otype == OT_RELOC) {
300                 if (rflag) {
620                         /*
621                          * We can only strip the symbol table and string table
622                          * if no output relocations will refer to them.
623                          */
624                         if (sflag)
625                                 ld_eprintf(ofl, ERR_WARNING,
626                                     MSG_INTL(MSG_ARG_STRIP),
627                                     MSG_INTL(MSG_MARG_REL),
628                                     MSG_INTL(MSG_MARG_STRIP));

630                         if (ztflag == NULL)
631                                 ofl->ofl_flags1 |= FLG_OF1_TEXTOFF;
632                         ofl->ofl_guideflags |= FLG_OFG_NO_TEXT;

634                         if (ofl->ofl_interp)
635                                 ld_eprintf(ofl, ERR_FATAL,
636                                     MSG_INTL(MSG_MARG_INCOMP),
637                                     MSG_INTL(MSG_MARG_REL),
638                                     MSG_ORIG(MSG_ARG_CI));
639                 } else {
640                         /*
641                          * Static executable.
642                          */
643                         ofl->ofl_flags |= FLG_OF_EXEC | FLG_OF_PROCRED;

645                         if (zdflag != SET_FALSE)
646                                 ofl->ofl_flags |= FLG_OF_NOUNDEF;
647                 }
648         }

650         /*
651          * If the user didn't supply an output file name supply a default.
652          */
653         if (ofl->ofl_name == NULL)
654                 ofl->ofl_name = MSG_ORIG(MSG_STR_AOUT);

656         /*
657          * We set the entrance criteria after all input argument processing as
658          * it is only at this point we're sure what the output image will be
659          * (static or dynamic).
660          */
661         if (ld_ent_setup(ofl, ld_targ.t_m.m_segm_align) == S_ERROR)
662                 return (S_ERROR);

664         /*
665          * Does the host currently running the linker have the same
666          * byte order as the target for which the object is being produced?
667          * If not, set FLG_OF1_ENCDIFF so relocation code will know
668          * to check.
669          */
670         if (_elf_sys_encoding() != ld_targ.t_m.m_data)
671                 ofl->ofl_flags1 |= FLG_OF1_ENCDIFF;
```



```
673         /*
674          * If the target has special executable section filling requirements,
675          * register the fill function with libelf
676          */
677         if (ld_targ.t_ff.ff_execfill != NULL)
678                 _elf_execfill(ld_targ.t_ff.ff_execfill);

680         /*
681          * Initialize string tables.  Symbol definitions within mapfiles can
682          * result in the creation of input sections.
683          */
684         if (ld_init_strings(ofl) == S_ERROR)
685                 return (S_ERROR);

687         /*
688          * Process mapfiles. Mapfile can redefine or add sections/segments,
689          * so this must come after the default entrance criteria are established
690          * (above).
691          */
692         if (ofl->ofl_maps) {
693                 const char      *name;
694                 Aliste          idx;

696                 for (APLIST_TRAVERSE(ofl->ofl_maps, idx, name))
697                         if (!ld_map_parse(name, ofl))
698                                 return (S_ERROR);

700                 if (!ld_map_post_process(ofl))
701                         return (S_ERROR);
702         }

704         /*
705          * If a mapfile has been used to define a single symbolic scope of
706          * interfaces, -Bsymbolic is established.  This global setting goes
707          * beyond individual symbol protection, and ensures all relocations
708          * (even those that reference section symbols) are processed within
709          * the object being built.
710          */
711         if (((ofl->ofl_flags &
712             (FLG_OF_MAPSYMB | FLG_OF_MAPGLOB)) == FLG_OF_MAPSYMB) &&
713             (ofl->ofl_flags & (FLG_OF_AUTOLCL | FLG_OF_AUTOELM))) {
714                 ofl->ofl_flags |= FLG_OF_SYMBOLIC;
715                 ofl->ofl_dtflags |= DF_SYMBOLIC;
716         }

718         /*
719          * If -zloadfltr is set, verify that filtering is in effect.  Filters
720          * are either established from the command line, and affect the whole
721          * object, or are set on a per-symbol basis from a mapfile.
722          */
723         if (zlflag) {
724                 if ((ofl->ofl_filtees == NULL) && (ofl->ofl_dtsfltrs == NULL))
725                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_NOFLTR),
726                             MSG_ORIG(MSG_ARG_ZLOADFLTR));
727                 ofl->ofl_dtflags_1 |= DF_1_LOADFLTR;
728         }

730         /*
731          * Check that we have something to work with. This check is carried out
732          * after mapfile processing as its possible a mapfile is being used to
733          * define symbols, in which case it would be sufficient to build the
734          * output file purely from the mapfile.
735          */
736         if ((ofl->ofl_objscnt == 0) && (ofl->ofl_soscnt == 0)) {
737                 if ((Vflag ||
738                     (Dflag && (dbg_desc->d_extra & DBG_E_HELP_EXIT))) &&
```

```
 739                     (argc == 2)) {
 740                         ofl->ofl_flags1 |= FLG_OF1_DONE;
 741                 } else {
 742                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_NOFILES));
 743                         return (S_ERROR);
 744                 }
 745         }
 746         return (1);
 747 }
_____unchanged_portion_omitted_

1011 static int      optitle = 0;
1012 /*
1013  * Parsing options pass1 for process_flags().
1014  */
1015 static uintptr_t
1016 parseopt_pass1(Ofl_desc *ofl, int argc, char **argv, int *usage)
1017 {
1018         int     c, ndx = optind;

1020         /*
1021          * The -32, -64 and -ztarget options are special, in that we validate
1022          * them, but otherwise ignore them. libld.so (this code) is called
1023          * from the ld front end program. ld has already examined the
1024          * arguments to determine the output class and machine type of the
1025          * output object, as reflected in the version (32/64) of ld_main()
1026          * that was called and the value of the 'mach' argument passed.
1027          * By time execution reaches this point, these options have already
1028          * been seen and acted on.
1029          */
1030         while ((c = ld_getopt(ofl->ofl_lml, ndx, argc, argv)) != -1) {

1032                 switch (c) {
1033                 case '3':
1034                         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));

1036                         /*
1037                          * -32 is processed by ld to determine the output class.
1038                          * Here we sanity check the option incase some other
1039                          * -3* option is mistakenly passed to us.
1040                          */
1041                         if (optarg[0] != '2')
1042                                 ld_eprintf(ofl, ERR_FATAL,
1043                                     MSG_INTL(MSG_ARG_ILLEGAL),
1044                                     MSG_ORIG(MSG_ARG_3), optarg);
1045                         continue;

1047                 case '6':
1048                         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));

1050                         /*
1051                          * -64 is processed by ld to determine the output class.
1052                          * Here we sanity check the option incase some other
1053                          * -6* option is mistakenly passed to us.
1054                          */
1055                         if (optarg[0] != '4')
1056                                 ld_eprintf(ofl, ERR_FATAL,
1057                                     MSG_INTL(MSG_ARG_ILLEGAL),
1058                                     MSG_ORIG(MSG_ARG_6), optarg);
1059                         continue;

1061                 case 'a':
1062                         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1063                         aflag = TRUE;
1064                         break;
```



```
1066                 case 'b':
1067                         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1068                         bflag = TRUE;

1070                         /*
1071                          * This is a hack, and may be undone later.
1072                          * The -b option is only used to build the Unix
1073                          * kernel and its related kernel-mode modules.
1074                          * We do not want those files to get a .SUNW_ldynsym
1075                          * section. At least for now, the kernel makes no
1076                          * use of .SUNW_ldynsym, and we do not want to use
1077                          * the space to hold it. Therefore, we overload
1078                          * the use of -b to also imply -znoldynsym.
1079                          */
1080                         ofl->ofl_flags |= FLG_OF_NOLDYNSYM;
1081                         break;

1083                 case 'c':
1084                         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1085                         if (ofl->ofl_config)
1086                                 ld_eprintf(ofl, ERR_WARNING_NF,
1087                                     MSG_INTL(MSG_ARG_MTONCE),
1088                                     MSG_ORIG(MSG_ARG_C));
1089                         else
1090                                 ofl->ofl_config = optarg;
1091                         break;

1093                 case 'C':
1094                         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1095                         demangle_flag = 1;
1096                         break;

1098                 case 'd':
1099                         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1100                         if ((optarg[0] == 'n') && (optarg[1] == '\0')) {
1101                                 if (dflag != SET_UNKNOWN)
1102                                         ld_eprintf(ofl, ERR_WARNING_NF,
1103                                             MSG_INTL(MSG_ARG_MTONCE),
1104                                             MSG_ORIG(MSG_ARG_D));
1105                                 else
1106                                         dflag = SET_FALSE;
1107                         } else if ((optarg[0] == 'y') && (optarg[1] == '\0')) {
1108                                 if (dflag != SET_UNKNOWN)
1109                                         ld_eprintf(ofl, ERR_WARNING_NF,
1110                                             MSG_INTL(MSG_ARG_MTONCE),
1111                                             MSG_ORIG(MSG_ARG_D));
1112                                 else
1113                                         dflag = SET_TRUE;
1114                         } else {
1115                                 ld_eprintf(ofl, ERR_FATAL,
1116                                     MSG_INTL(MSG_ARG_ILLEGAL),
1117                                     MSG_ORIG(MSG_ARG_D), optarg);
1118                         }
1119                         break;

1121                 case 'e':
1122                         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1123                         if (ofl->ofl_entry)
1124                                 ld_eprintf(ofl, ERR_WARNING_NF,
1125                                     MSG_INTL(MSG_MARG_MTONCE),
1126                                     MSG_INTL(MSG_MARG_ENTRY));
1127                         else
1128                                 ofl->ofl_entry = (void *)optarg;
1129                         break;

1131                 case 'f':
```

```
1132                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1133                        if (ofl->ofl_filtees &&
1134                            (!(ofl->ofl_flags & FLG_OF_AUX))) {
1135                                ld_eprintf(ofl, ERR_FATAL,
1136                                    MSG_INTL(MSG_MARG_INCOMP),
1137                                    MSG_INTL(MSG_MARG_FILTER_AUX),
1138                                    MSG_INTL(MSG_MARG_FILTER));
1139                        } else {
1140                                if ((ofl->ofl_filtees =
1141                                    add_string(ofl->ofl_filtees, optarg)) ==
1142                                    (const char *)S_ERROR)
1143                                        return (S_ERROR);
1144                                ofl->ofl_flags |= FLG_OF_AUX;
1145                        }
1146                        break;

1148                case 'F':
1149                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1150                        if (ofl->ofl_filtees &&
1151                            (ofl->ofl_flags & FLG_OF_AUX)) {
1152                                ld_eprintf(ofl, ERR_FATAL,
1153                                    MSG_INTL(MSG_MARG_INCOMP),
1154                                    MSG_INTL(MSG_MARG_FILTER),
1155                                    MSG_INTL(MSG_MARG_FILTER_AUX));
1156                        } else {
1157                                if ((ofl->ofl_filtees =
1158                                    add_string(ofl->ofl_filtees, optarg)) ==
1159                                    (const char *)S_ERROR)
1160                                        return (S_ERROR);
1161                        }
1162                        break;

1164                case 'h':
1165                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1166                        if (ofl->ofl_soname)
1167                                ld_eprintf(ofl, ERR_WARNING_NF,
1168                                    MSG_INTL(MSG_MARG_MTONCE),
1169                                    MSG_INTL(MSG_MARG_SONAME));
1170                        else
1171                                ofl->ofl_soname = (const char *)optarg;
1172                        break;

1174                case 'i':
1175                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1176                        ofl->ofl_flags |= FLG_OF_IGNENV;
1177                        break;

1179                case 'I':
1180                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1181                        if (ofl->ofl_interp)
1182                                ld_eprintf(ofl, ERR_WARNING_NF,
1183                                    MSG_INTL(MSG_ARG_MTONCE),
1184                                    MSG_ORIG(MSG_ARG_CI));
1185                        else
1186                                ofl->ofl_interp = (const char *)optarg;
1187                        break;

1189                case 'l':
1190                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1191                        /*
1192                         * For now, count any library as a shared object.  This
1193                         * is used to size the internal symbol cache.  This
1194                         * value is recalculated later on actual file processing
1195                         * to get an accurate shared object count.
1196                         */
1197                        ofl->ofl_soscnt++;
```



```
1198                        break;

1200                case 'm':
1201                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1202                        ofl->ofl_flags |= FLG_OF_GENMAP;
1203                        break;

1205                case 'o':
1206                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1207                        if (ofl->ofl_name)
1208                                ld_eprintf(ofl, ERR_WARNING_NF,
1209                                    MSG_INTL(MSG_MARG_MTONCE),
1210                                    MSG_INTL(MSG_MARG_OUTFILE));
1211                        else
1212                                ofl->ofl_name = (const char *)optarg;
1213                        break;

1215                case 'p':
1216                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));

1218                        /*
1219                         * Multiple instances of this option may occur.  Each
1220                         * additional instance is effectively concatenated to
1221                         * the previous separated by a colon.
1222                         */
1223                        if (*optarg != '\0') {
1224                                if ((ofl->ofl_audit =
1225                                    add_string(ofl->ofl_audit,
1226                                    optarg)) == (const char *)S_ERROR)
1227                                        return (S_ERROR);
1228                        }
1229                        break;

1231                case 'P':
1232                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));

1234                        /*
1235                         * Multiple instances of this option may occur.  Each
1236                         * additional instance is effectively concatenated to
1237                         * the previous separated by a colon.
1238                         */
1239                        if (*optarg != '\0') {
1240                                if ((ofl->ofl_depaudit =
1241                                    add_string(ofl->ofl_depaudit,
1242                                    optarg)) == (const char *)S_ERROR)
1243                                        return (S_ERROR);
1244                        }
1245                        break;

1247                case 'r':
1248                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1249                        otype = OT_RELOC;
 930                        rflag = TRUE;
1250                        break;

1252                case 'R':
1253                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));

1255                        /*
1256                         * Multiple instances of this option may occur.  Each
1257                         * additional instance is effectively concatenated to
1258                         * the previous separated by a colon.
1259                         */
1260                        if (*optarg != '\0') {
1261                                if ((ofl->ofl_rpath =
1262                                    add_string(ofl->ofl_rpath,
```

```
1263                                    optarg)) == (const char *)S_ERROR)
1264                                        return (S_ERROR);
1265                            }
1266                            break;

1268                    case 's':
1269                            DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1270                            sflag = TRUE;
1271                            break;

1273                    case 't':
1274                            DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1275                            ofl->ofl_flags |= FLG_OF_NOWARN;
1276                            break;

1278                    case 'u':
1279                            DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1280                            break;

1282                    case 'z':
1283                            DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));

1285                            /*
1286                             * Skip comma that might be present between -z and its
1287                             * argument (e.g. if -Wl,-z,assert-deflib was passed).
1288                             */
1289                            if (strncmp(optarg, MSG_ORIG(MSG_STR_COMMA),
1290                                MSG_STR_COMMA_SIZE) == 0)
1291                                    optarg++;

1293                            /*
1294                             * For specific help, print our usage message and exit
1295                             * immediately to ensure a 0 return code.
1296                             */
1297                            if (strncmp(optarg, MSG_ORIG(MSG_ARG_HELP),
1298                                MSG_ARG_HELP_SIZE) == 0) {
1299                                    usage_mesg(TRUE);
1300                                    exit(0);
1301                            }

1303                            /*
1304                             * For some options set a flag - further consistancy
1305                             * checks will be carried out in check_flags().
1306                             */
1307                            if ((strncmp(optarg, MSG_ORIG(MSG_ARG_LD32),
1308                                MSG_ARG_LD32_SIZE) == 0) ||
1309                                (strncmp(optarg, MSG_ORIG(MSG_ARG_LD64),
1310                                MSG_ARG_LD64_SIZE) == 0)) {
1311                                    if (createargv(ofl, usage) == S_ERROR)
1312                                            return (S_ERROR);

1314                            } else if (
1315                                strcmp(optarg, MSG_ORIG(MSG_ARG_DEFS)) == 0) {
1316                                    if (zdflag != SET_UNKNOWN)
1317                                            ld_eprintf(ofl, ERR_WARNING_NF,
1318                                                MSG_INTL(MSG_ARG_MTONCE),
1319                                                MSG_ORIG(MSG_ARG_ZDEFNODEF));
1320                                    else
1321                                            zdflag = SET_TRUE;
1322                                    ofl->ofl_guideflags |= FLG_OFG_NO_DEFS;
1323                            } else if (strcmp(optarg,
1324                                MSG_ORIG(MSG_ARG_NODEFS)) == 0) {
1325                                    if (zdflag != SET_UNKNOWN)
1326                                            ld_eprintf(ofl, ERR_WARNING_NF,
1327                                                MSG_INTL(MSG_ARG_MTONCE),
1328                                                MSG_ORIG(MSG_ARG_ZDEFNODEF));
```



```
1329                                    else
1330                                            zdflag = SET_FALSE;
1331                                    ofl->ofl_guideflags |= FLG_OFG_NO_DEFS;
1332                            } else if (strcmp(optarg,
1333                                MSG_ORIG(MSG_ARG_TEXT)) == 0) {
1334                                    if (ztflag &&
1335                                        (ztflag != MSG_ORIG(MSG_ARG_ZTEXT)))
1336                                            ld_eprintf(ofl, ERR_FATAL,
1337                                                MSG_INTL(MSG_ARG_INCOMP),
1338                                                MSG_ORIG(MSG_ARG_ZTEXT),
1339                                                ztflag);
1340                                    ztflag = MSG_ORIG(MSG_ARG_ZTEXT);
1341                            } else if (strcmp(optarg,
1342                                MSG_ORIG(MSG_ARG_TEXTOFF)) == 0) {
1343                                    if (ztflag &&
1344                                        (ztflag != MSG_ORIG(MSG_ARG_ZTEXTOFF)))
1345                                            ld_eprintf(ofl, ERR_FATAL,
1346                                                MSG_INTL(MSG_ARG_INCOMP),
1347                                                MSG_ORIG(MSG_ARG_ZTEXTOFF),
1348                                                ztflag);
1349                                    ztflag = MSG_ORIG(MSG_ARG_ZTEXTOFF);
1350                            } else if (strcmp(optarg,
1351                                MSG_ORIG(MSG_ARG_TEXTWARN)) == 0) {
1352                                    if (ztflag &&
1353                                        (ztflag != MSG_ORIG(MSG_ARG_ZTEXTWARN)))
1354                                            ld_eprintf(ofl, ERR_FATAL,
1355                                                MSG_INTL(MSG_ARG_INCOMP),
1356                                                MSG_ORIG(MSG_ARG_ZTEXTWARN),
1357                                                ztflag);
1358                                    ztflag = MSG_ORIG(MSG_ARG_ZTEXTWARN);

1360                            /*
1361                             * For other options simply set the ofl flags directly.
1362                             */
1363                            } else if (strcmp(optarg,
1364                                MSG_ORIG(MSG_ARG_RESCAN)) == 0) {
1365                                    ofl->ofl_flags1 |= FLG_OF1_RESCAN;
1366                            } else if (strcmp(optarg,
1367                                MSG_ORIG(MSG_ARG_ABSEXEC)) == 0) {
1368                                    ofl->ofl_flags1 |= FLG_OF1_ABSEXEC;
1369                            } else if (strcmp(optarg,
1370                                MSG_ORIG(MSG_ARG_LOADFLTR)) == 0) {
1371                                    zlflag = TRUE;
1372                            } else if (strcmp(optarg,
1373                                MSG_ORIG(MSG_ARG_NORELOC)) == 0) {
1374                                    ofl->ofl_dtflags_1 |= DF_1_NORELOC;
1375                            } else if (strcmp(optarg,
1376                                MSG_ORIG(MSG_ARG_NOVERSION)) == 0) {
1377                                    ofl->ofl_flags |= FLG_OF_NOVERSEC;
1378                            } else if (strcmp(optarg,
1379                                MSG_ORIG(MSG_ARG_MULDEFS)) == 0) {
1380                                    ofl->ofl_flags |= FLG_OF_MULDEFS;
1381                            } else if (strcmp(optarg,
1382                                MSG_ORIG(MSG_ARG_REDLOCSYM)) == 0) {
1383                                    ofl->ofl_flags |= FLG_OF_REDLSYM;
1384                            } else if (strcmp(optarg,
1385                                MSG_ORIG(MSG_ARG_INITFIRST)) == 0) {
1386                                    ofl->ofl_dtflags_1 |= DF_1_INITFIRST;
1387                            } else if (strcmp(optarg,
1388                                MSG_ORIG(MSG_ARG_NODELETE)) == 0) {
1389                                    ofl->ofl_dtflags_1 |= DF_1_NODELETE;
1390                            } else if (strcmp(optarg,
1391                                MSG_ORIG(MSG_ARG_NOPARTIAL)) == 0) {
1392                                    ofl->ofl_flags1 |= FLG_OF1_NOPARTI;
1393                            } else if (strcmp(optarg,
1394                                MSG_ORIG(MSG_ARG_NOOPEN)) == 0) {
```

```
1395                            ofl->ofl_dtflags_1 |= DF_1_NOOPEN;
1396                    } else if (strcmp(optarg,
1397                        MSG_ORIG(MSG_ARG_NOW)) == 0) {
1398                            ofl->ofl_dtflags_1 |= DF_1_NOW;
1399                            ofl->ofl_dtflags |= DF_BIND_NOW;
1400                    } else if (strcmp(optarg,
1401                        MSG_ORIG(MSG_ARG_ORIGIN)) == 0) {
1402                            ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1403                            ofl->ofl_dtflags |= DF_ORIGIN;
1404                    } else if (strcmp(optarg,
1405                        MSG_ORIG(MSG_ARG_NODEFAULTLIB)) == 0) {
1406                            ofl->ofl_dtflags_1 |= DF_1_NODEFLIB;
1407                    } else if (strcmp(optarg,
1408                        MSG_ORIG(MSG_ARG_NODUMP)) == 0) {
1409                            ofl->ofl_dtflags_1 |= DF_1_NODUMP;
1410                    } else if (strcmp(optarg,
1411                        MSG_ORIG(MSG_ARG_ENDFILTEE)) == 0) {
1412                            ofl->ofl_dtflags_1 |= DF_1_ENDFILTEE;
1413                    } else if (strcmp(optarg,
1414                        MSG_ORIG(MSG_ARG_VERBOSE)) == 0) {
1415                            ofl->ofl_flags |= FLG_OF_VERBOSE;
1416                    } else if (strcmp(optarg,
1417                        MSG_ORIG(MSG_ARG_COMBRELOC)) == 0) {
1418                            ofl->ofl_flags |= FLG_OF_COMREL;
1419                    } else if (strcmp(optarg,
1420                        MSG_ORIG(MSG_ARG_NOCOMBRELOC)) == 0) {
1421                            ofl->ofl_flags |= FLG_OF_NOCOMREL;
1422                    } else if (strcmp(optarg,
1423                        MSG_ORIG(MSG_ARG_NOCOMPSTRTAB)) == 0) {
1424                            ofl->ofl_flags1 |= FLG_OF1_NCSTTAB;
1425                    } else if (strcmp(optarg,
1426                        MSG_ORIG(MSG_ARG_NOINTERP)) == 0) {
1427                            ofl->ofl_flags1 |= FLG_OF1_NOINTRP;
1428                    } else if (strcmp(optarg,
1429                        MSG_ORIG(MSG_ARG_INTERPOSE)) == 0) {
1430                            zinflag = TRUE;
1431                    } else if (strcmp(optarg,
1432                        MSG_ORIG(MSG_ARG_IGNORE)) == 0) {
1433                            ofl->ofl_flags1 |= FLG_OF1_IGNPRC;
1434                    } else if (strcmp(optarg,
1435                        MSG_ORIG(MSG_ARG_RELAXRELOC)) == 0) {
1436                            ofl->ofl_flags1 |= FLG_OF1_RLXREL;
1437                    } else if (strcmp(optarg,
1438                        MSG_ORIG(MSG_ARG_NORELAXRELOC)) == 0) {
1439                            ofl->ofl_flags1 |= FLG_OF1_NRLXREL;
1440                    } else if (strcmp(optarg,
1441                        MSG_ORIG(MSG_ARG_NOLDYNSYM)) == 0) {
1442                            ofl->ofl_flags |= FLG_OF_NOLDYNSYM;
1443                    } else if (strcmp(optarg,
1444                        MSG_ORIG(MSG_ARG_GLOBAUDIT)) == 0) {
1445                            ofl->ofl_dtflags_1 |= DF_1_GLOBAUDIT;
1446                    } else if (strcmp(optarg,
1447                        MSG_ORIG(MSG_ARG_NOSIGHANDLER)) == 0) {
1448                            ofl->ofl_flags1 |= FLG_OF1_NOSGHND;
1449                    } else if (strcmp(optarg,
1450                        MSG_ORIG(MSG_ARG_SYMBOLCAP)) == 0) {
1451                            ofl->ofl_flags |= FLG_OF_OTOSCAP;

1453                    /*
1454                     * Check archive group usage
1455                     *      -z rescan-start ... -z rescan-end
1456                     * to ensure they don't overlap and are well formed.
1457                     */
1458                    } else if (strcmp(optarg,
1459                        MSG_ORIG(MSG_ARG_RESCAN_START)) == 0) {
1460                            if (ofl->ofl_ars_gsandx == 0) {
```



```
1461                                    ofl->ofl_ars_gsandx = ndx;
1462                            } else if (ofl->ofl_ars_gsandx > 0) {
1463                                    /* Another group is still open */
1464                                    ld_eprintf(ofl, ERR_FATAL,
1465                                        MSG_INTL(MSG_ARG_AR_GRP_OLAP),
1466                                        MSG_INTL(MSG_MARG_AR_GRPS));
1467                                    /* Don't report cascading errors */
1468                                    ofl->ofl_ars_gsandx = -1;
1469                            }
1470                    } else if (strcmp(optarg,
1471                        MSG_ORIG(MSG_ARG_RESCAN_END)) == 0) {
1472                            if (ofl->ofl_ars_gsandx > 0) {
1473                                    ofl->ofl_ars_gsandx = 0;
1474                            } else if (ofl->ofl_ars_gsandx == 0) {
1475                                    /* There was no matching begin */
1476                                    ld_eprintf(ofl, ERR_FATAL,
1477                                        MSG_INTL(MSG_ARG_AR_GRP_BAD),
1478                                        MSG_INTL(MSG_MARG_AR_GRP_END),
1479                                        MSG_INTL(MSG_MARG_AR_GRP_START));
1480                                    /* Don't report cascading errors */
1481                                    ofl->ofl_ars_gsandx = -1;
1482                            }

1484                    /*
1485                     * If -z wrap is seen, enter the symbol to be wrapped
1486                     * into the wrap AVL tree.
1487                     */
1488                    } else if (strncmp(optarg, MSG_ORIG(MSG_ARG_WRAP),
1489                        MSG_ARG_WRAP_SIZE) == 0) {
1490                            if (ld_wrap_enter(ofl,
1491                                optarg + MSG_ARG_WRAP_SIZE) == NULL)
1492                                    return (S_ERROR);
1493                    } else if (strncmp(optarg, MSG_ORIG(MSG_ARG_ASLR),
1494                        MSG_ARG_ASLR_SIZE) == 0) {
1495                            char *p = optarg + MSG_ARG_ASLR_SIZE;
1496                            if (*p == '\0') {
1497                                    ofl->ofl_aslr = 1;
1498                            } else if (*p == '=') {
1499                                    p++;

1501                                    if ((strcmp(p,
1502                                        MSG_ORIG(MSG_ARG_ENABLED)) == 0) ||
1503                                        (strcmp(p,
1504                                        MSG_ORIG(MSG_ARG_ENABLE)) == 0)) {
1505                                            ofl->ofl_aslr = 1;
1506                                    } else if ((strcmp(p,
1507                                        MSG_ORIG(MSG_ARG_DISABLED)) == 0) ||
1508                                        (strcmp(p,
1509                                        MSG_ORIG(MSG_ARG_DISABLE)) == 0)) {
1510                                            ofl->ofl_aslr = -1;
1511                                    } else {
1512                                            ld_eprintf(ofl, ERR_FATAL,
1513                                                MSG_INTL(MSG_ARG_ILLEGAL),
1514                                                MSG_ORIG(MSG_ARG_ZASLR), p);
1515                                            return (S_ERROR);
1516                                    }
1517                            } else {
1518                                    ld_eprintf(ofl, ERR_FATAL,
1519                                        MSG_INTL(MSG_ARG_ILLEGAL),
1520                                        MSG_ORIG(MSG_ARG_Z), optarg);
1521                                    return (S_ERROR);
1522                            }
1523                    } else if ((strncmp(optarg, MSG_ORIG(MSG_ARG_GUIDE),
1524                        MSG_ARG_GUIDE_SIZE) == 0) &&
1525                        ((optarg[MSG_ARG_GUIDE_SIZE] == '=') ||
1526                        (optarg[MSG_ARG_GUIDE_SIZE] == '\0'))) {
```

```
1527                                    if (!guidance_parse(ofl, optarg))
1528                                            return (S_ERROR);
1529                            } else if (strcmp(optarg,
1530                                MSG_ORIG(MSG_ARG_FATWARN)) == 0) {
1531                                    if (zfwflag  == SET_FALSE) {
1532                                            ld_eprintf(ofl, ERR_WARNING_NF,
1533                                                MSG_INTL(MSG_ARG_MTONCE),
1534                                                MSG_ORIG(MSG_ARG_ZFATWNOFATW));
1535                                    } else {
1536                                            zfwflag = SET_TRUE;
1537                                            ofl->ofl_flags |= FLG_OF_FATWARN;
1538                                    }
1539                            } else if (strcmp(optarg,
1540                                MSG_ORIG(MSG_ARG_NOFATWARN)) == 0) {
1541                                    if (zfwflag  == SET_TRUE)
1542                                            ld_eprintf(ofl, ERR_WARNING_NF,
1543                                                MSG_INTL(MSG_ARG_MTONCE),
1544                                                MSG_ORIG(MSG_ARG_ZFATWNOFATW));
1545                                    else
1546                                            zfwflag = SET_FALSE;

1548                            /*
1549                             * Process everything related to -z assert-deflib. This
1550                             * must be done in pass 1 because it gets used in pass
1551                             * 2.
1552                             */
1553                            } else if (strncmp(optarg, MSG_ORIG(MSG_ARG_ASSDEFLIB),
1554                                MSG_ARG_ASSDEFLIB_SIZE) == 0) {
1555                                    if (assdeflib_parse(ofl, optarg) != TRUE)
1556                                            return (S_ERROR);
1557                            } else if (strncmp(optarg, MSG_ORIG(MSG_ARG_TYPE),
1558                                MSG_ARG_TYPE_SIZE) == 0) {
1559                                    char *p = optarg + MSG_ARG_TYPE_SIZE;
1560                                    if (*p != '=') {
1561                                            ld_eprintf(ofl, ERR_FATAL,
1562                                                MSG_INTL(MSG_ARG_ILLEGAL),
1563                                                MSG_ORIG(MSG_ARG_Z), optarg);
1564                                            return (S_ERROR);
1565                                    }

1567                                    p++;
1568                                    if (strcmp(p,
1569                                        MSG_ORIG(MSG_ARG_TYPE_RELOC)) == 0) {
1570                                            otype = OT_RELOC;
1571                                    } else if (strcmp(p,
1572                                        MSG_ORIG(MSG_ARG_TYPE_EXEC)) == 0) {
1573                                            otype = OT_EXEC;
1574                                    } else if (strcmp(p,
1575                                        MSG_ORIG(MSG_ARG_TYPE_SHARED)) == 0) {
1576                                            otype = OT_SHARED;
1577                                    } else if (strcmp(p,
1578                                        MSG_ORIG(MSG_ARG_TYPE_KMOD)) == 0) {
1579                                            otype = OT_KMOD;
1580                                    } else {
1581                                            ld_eprintf(ofl, ERR_FATAL,
1582                                                MSG_INTL(MSG_ARG_ILLEGAL),
1583                                                MSG_ORIG(MSG_ARG_Z), optarg);
1584                                            return (S_ERROR);
1585                                    }
1586 #endif /* ! codereview */
1587                            /*
1588                             * The following options just need validation as they
1589                             * are interpreted on the second pass through the
1590                             * command line arguments.
1591                             */
1592                            } else if (
```



```
1593                                strncmp(optarg, MSG_ORIG(MSG_ARG_INITARRAY),
1594                                MSG_ARG_INITARRAY_SIZE) &&
1595                                strncmp(optarg, MSG_ORIG(MSG_ARG_FINIARRAY),
1596                                MSG_ARG_FINIARRAY_SIZE) &&
1597                                strncmp(optarg, MSG_ORIG(MSG_ARG_PREINITARRAY),
1598                                MSG_ARG_PREINITARRAY_SIZE) &&
1599                                strncmp(optarg, MSG_ORIG(MSG_ARG_RTLDINFO),
1600                                MSG_ARG_RTLDINFO_SIZE) &&
1601                                strncmp(optarg, MSG_ORIG(MSG_ARG_DTRACE),
1602                                MSG_ARG_DTRACE_SIZE) &&
1603                                strcmp(optarg, MSG_ORIG(MSG_ARG_ALLEXTRT)) &&
1604                                strcmp(optarg, MSG_ORIG(MSG_ARG_DFLEXTRT)) &&
1605                                strcmp(optarg, MSG_ORIG(MSG_ARG_DIRECT)) &&
1606                                strcmp(optarg, MSG_ORIG(MSG_ARG_NODIRECT)) &&
1607                                strcmp(optarg, MSG_ORIG(MSG_ARG_GROUPPERM)) &&
1608                                strcmp(optarg, MSG_ORIG(MSG_ARG_LAZYLOAD)) &&
1609                                strcmp(optarg, MSG_ORIG(MSG_ARG_NOGROUPPERM)) &&
1610                                strcmp(optarg, MSG_ORIG(MSG_ARG_NOLAZYLOAD)) &&
1611                                strcmp(optarg, MSG_ORIG(MSG_ARG_NODEFERRED)) &&
1612                                strcmp(optarg, MSG_ORIG(MSG_ARG_RECORD)) &&
1613                                strcmp(optarg, MSG_ORIG(MSG_ARG_ALTEXEC64)) &&
1614                                strcmp(optarg, MSG_ORIG(MSG_ARG_WEAKEXT)) &&
1615                                strncmp(optarg, MSG_ORIG(MSG_ARG_TARGET),
1616                                MSG_ARG_TARGET_SIZE) &&
1617                                strcmp(optarg, MSG_ORIG(MSG_ARG_RESCAN_NOW)) &&
1618                                strcmp(optarg, MSG_ORIG(MSG_ARG_DEFERRED))) {
1619                                    ld_eprintf(ofl, ERR_FATAL,
1620                                        MSG_INTL(MSG_ARG_ILLEGAL),
1621                                        MSG_ORIG(MSG_ARG_Z), optarg);
1622                            }

1624                            break;

1626                    case 'D':
1627                            /*
1628                             * If we have not yet read any input files go ahead
1629                             * and process any debugging options (this allows any
1630                             * argument processing, entrance criteria and library
1631                             * initialization to be displayed).  Otherwise, if an
1632                             * input file has been seen, skip interpretation until
1633                             * process_files (this allows debugging to be turned
1634                             * on and off around individual groups of files).
1635                             */
1636                            Dflag = 1;
1637                            if (ofl->ofl_objscnt == 0) {
1638                                    if (dbg_setup(ofl, optarg, 2) == 0)
1639                                            return (S_ERROR);
1640                            }

1642                            /*
1643                             * A diagnostic can only be provided after dbg_setup().
1644                             * As this is the first diagnostic that can be produced
1645                             * by ld(1), issue a title for timing and basic output.
1646                             */
1647                            if ((optitle == 0) && DBG_ENABLED) {
1648                                    optitle++;
1649                                    DBG_CALL(Dbg_basic_options(ofl->ofl_lml));
1650                            }
1651                            DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1652                            break;

1654                    case 'B':
1655                            DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1656                            if (strcmp(optarg, MSG_ORIG(MSG_ARG_DIRECT)) == 0) {
1657                                    if (Bdflag == SET_FALSE) {
1658                                            ld_eprintf(ofl, ERR_FATAL,
```

```
1659                                        MSG_INTL(MSG_ARG_INCOMP),
1660                                        MSG_ORIG(MSG_ARG_BNODIRECT),
1661                                        MSG_ORIG(MSG_ARG_BDIRECT));
1662                                } else {
1663                                        Bdflag = SET_TRUE;
1664                                        ofl->ofl_guideflags |= FLG_OFG_NO_DB;
1665                                }
1666                        } else if (strcmp(optarg,
1667                            MSG_ORIG(MSG_ARG_NODIRECT)) == 0) {
1668                                if (Bdflag == SET_TRUE) {
1669                                        ld_eprintf(ofl, ERR_FATAL,
1670                                            MSG_INTL(MSG_ARG_INCOMP),
1671                                            MSG_ORIG(MSG_ARG_BDIRECT),
1672                                            MSG_ORIG(MSG_ARG_BNODIRECT));
1673                                } else {
1674                                        Bdflag = SET_FALSE;
1675                                        ofl->ofl_guideflags |= FLG_OFG_NO_DB;
1676                                }
1677                        } else if (strcmp(optarg,
1678                            MSG_ORIG(MSG_STR_SYMBOLIC)) == 0)
1679                                Bsflag = TRUE;
1680                        else if (strcmp(optarg, MSG_ORIG(MSG_ARG_REDUCE)) == 0)
1681                                ofl->ofl_flags |= FLG_OF_PROCRED;
1682                        else if (strcmp(optarg, MSG_ORIG(MSG_STR_LOCAL)) == 0)
1683                                Blflag = TRUE;
1684                        else if (strcmp(optarg, MSG_ORIG(MSG_ARG_GROUP)) == 0)
1685                                Bgflag = TRUE;
1686                        else if (strcmp(optarg,
1687                            MSG_ORIG(MSG_STR_ELIMINATE)) == 0)
1688                                Beflag = TRUE;
1689                        else if (strcmp(optarg,
1690                            MSG_ORIG(MSG_ARG_TRANSLATOR)) == 0) {
1691                                ld_eprintf(ofl, ERR_WARNING,
1692                                    MSG_INTL(MSG_ARG_UNSUPPORTED),
1693                                    MSG_ORIG(MSG_ARG_BTRANSLATOR));
1694                        } else if (strcmp(optarg,
1695                            MSG_ORIG(MSG_STR_LD_DYNAMIC)) &&
1696                            strcmp(optarg, MSG_ORIG(MSG_ARG_STATIC))) {
1697                                ld_eprintf(ofl, ERR_FATAL,
1698                                    MSG_INTL(MSG_ARG_ILLEGAL),
1699                                    MSG_ORIG(MSG_ARG_CB), optarg);
1700                        }
1701                        break;

1703                case 'G':
1704                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1705                        otype = OT_SHARED;
1238                        Gflag = TRUE;
1706                        break;

1708                case 'L':
1709                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1710                        break;

1712                case 'M':
1713                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1714                        if (aplist_append(&(ofl->ofl_maps), optarg,
1715                            AL_CNT_OFL_MAPFILES) == NULL)
1716                                return (S_ERROR);
1717                        break;

1719                case 'N':
1720                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1721                        break;

1723                case 'Q':
```



```
1724                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1725                        if ((optarg[0] == 'n') && (optarg[1] == '\0')) {
1726                                if (Qflag != SET_UNKNOWN)
1727                                        ld_eprintf(ofl, ERR_WARNING_NF,
1728                                            MSG_INTL(MSG_ARG_MTONCE),
1729                                            MSG_ORIG(MSG_ARG_CQ));
1730                                else
1731                                        Qflag = SET_FALSE;
1732                        } else if ((optarg[0] == 'y') && (optarg[1] == '\0')) {
1733                                if (Qflag != SET_UNKNOWN)
1734                                        ld_eprintf(ofl, ERR_WARNING_NF,
1735                                            MSG_INTL(MSG_ARG_MTONCE),
1736                                            MSG_ORIG(MSG_ARG_CQ));
1737                                else
1738                                        Qflag = SET_TRUE;
1739                        } else {
1740                                ld_eprintf(ofl, ERR_FATAL,
1741                                    MSG_INTL(MSG_ARG_ILLEGAL),
1742                                    MSG_ORIG(MSG_ARG_CQ), optarg);
1743                        }
1744                        break;

1746                case 'S':
1747                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1748                        if (aplist_append(&lib_support, optarg,
1749                            AL_CNT_SUPPORT) == NULL)
1750                                return (S_ERROR);
1751                        break;

1753                case 'V':
1754                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1755                        if (!Vflag)
1756                                (void) fprintf(stderr, MSG_ORIG(MSG_STR_STRNL),
1757                                    ofl->ofl_sgsid);
1758                        Vflag = TRUE;
1759                        break;

1761                case 'Y':
1762                        DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1763                        if (strncmp(optarg, MSG_ORIG(MSG_ARG_LCOM), 2) == 0) {
1764                                if (Llibdir)
1765                                        ld_eprintf(ofl, ERR_WARNING_NF,
1766                                            MSG_INTL(MSG_ARG_MTONCE),
1767                                            MSG_ORIG(MSG_ARG_CYL));
1768                                else
1769                                        Llibdir = optarg + 2;
1770                        } else if (strncmp(optarg,
1771                            MSG_ORIG(MSG_ARG_UCOM), 2) == 0) {
1772                                if (Ulibdir)
1773                                        ld_eprintf(ofl, ERR_WARNING_NF,
1774                                            MSG_INTL(MSG_ARG_MTONCE),
1775                                            MSG_ORIG(MSG_ARG_CYU));
1776                                else
1777                                        Ulibdir = optarg + 2;
1778                        } else if (strncmp(optarg,
1779                            MSG_ORIG(MSG_ARG_PCOM), 2) == 0) {
1780                                if (Plibpath)
1781                                        ld_eprintf(ofl, ERR_WARNING_NF,
1782                                            MSG_INTL(MSG_ARG_MTONCE),
1783                                            MSG_ORIG(MSG_ARG_CYP));
1784                                else
1785                                        Plibpath = optarg + 2;
1786                        } else {
1787                                ld_eprintf(ofl, ERR_FATAL,
1788                                    MSG_INTL(MSG_ARG_ILLEGAL),
1789                                    MSG_ORIG(MSG_ARG_CY), optarg);
```

```
1790                            }
1791                            break;

1793                    case '?':
1794                            DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1795                            /*
1796                             * If the option character is '-', we're looking at a
1797                             * long option which couldn't be translated, display a
1798                             * more useful error.
1799                             */
1800                            if (optopt == '-') {
1801                                    eprintf(ofl->ofl_lml, ERR_FATAL,
1802                                        MSG_INTL(MSG_ARG_LONG_UNKNOWN),
1803                                        argv[optind-1]);
1804                            } else {
1805                                    eprintf(ofl->ofl_lml, ERR_FATAL,
1806                                        MSG_INTL(MSG_ARG_UNKNOWN), optopt);
1807                            }
1808                            (*usage)++;
1809                            break;

1811                    default:
1812                            break;
1813                    }

1815                    /*
1816                     * Update the argument index for the next getopt() iteration.
1817                     */
1818                    ndx = optind;
1819            }
1820            return (1);
1821 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  107936 Sun Feb 24 19:19:11 2019
new/usr/src/cmd/sgs/libld/common/files.c
ld should reject kernel modules as input
**********************************************************
______unchanged_portion_omitted_

3023 /*
3024  * Process the current input file.  There are basically three types of files
3025  * that come through here:
3026  *
3027  *  -   files explicitly defined on the command line (ie. foo.o or bar.so),
3028  *      in this case only the 'name' field is valid.
3029  *
3030  *  -   libraries determined from the -l command line option (ie. -lbar),
3031  *      in this case the 'soname' field contains the basename of the located
3032  *      file.
3033  *
3034  * Any shared object specified via the above two conventions must be recorded
3035  * as a needed dependency.
3036  *
3037  *  -   libraries specified as dependencies of those libraries already obtained
3038  *      via the command line (ie. bar.so has a DT_NEEDED entry of fred.so.1),
3039  *      in this case the 'soname' field contains either a full pathname (if the
3040  *      needed entry contained a '/'), or the basename of the located file.
3041  *      These libraries are processed to verify symbol binding but are not
3042  *      recorded as dependencies of the output file being generated.
3043  *
3044  * entry:
3045  *      name - File name
3046  *      soname - SONAME for needed sharable library, as described above
3047  *      fd - Open file descriptor
3048  *      elf - Open ELF handle
3049  *      flags - FLG_IF_ flags applicable to file
3050  *      ofl - Output file descriptor
3051  *      rej - Rejection descriptor used to record rejection reason
3052  *      ifl_ret - NULL, or address of pointer to receive reference to
3053  *              resulting input descriptor for file. If ifl_ret is non-NULL,
3054  *              the file cannot be an archive or it will be rejected.
3055  *
3056  * exit:
3057  *      If a error occurs in examining the file, S_ERROR is returned.
3058  *      If the file can be examined, but is not suitable, *rej is updated,
3059  *      and 0 is returned. If the file is acceptable, 1 is returned, and if
3060  *      ifl_ret is non-NULL, *ifl_ret is set to contain the pointer to the
3061  *      resulting input descriptor.
3062  */
3063 uintptr_t
3064 ld_process_ifl(const char *name, const char *soname, int fd, Elf *elf,
3065     Word flags, Ofl_desc *ofl, Rej_desc *rej, Ifl_desc **ifl_ret)
3066 {
3067         Ifl_desc        *ifl;
3068         Ehdr            *ehdr;
3069         uintptr_t       error = 0;
3070         struct stat     status;
3071         Ar_desc         *adp;
3072         Rej_desc        _rej;

3074         /*
3075          * If this file was not extracted from an archive obtain its device
3076          * information.  This will be used to determine if the file has already
3077          * been processed (rather than simply comparing filenames, the device
3078          * information provides a quicker comparison and detects linked files).
3079          */
3080         if (fd && ((flags & FLG_IF_EXTRACT) == 0))
3081                 (void) fstat(fd, &status);
```



```
3082         else {
3083                 status.st_dev = 0;
3084                 status.st_ino = 0;
3085         }

3087         switch (elf_kind(elf)) {
3088         case ELF_K_AR:
3089                 /*
3090                  * If the caller has supplied a non-NULL ifl_ret, then
3091                  * we cannot process archives, for there will be no
3092                  * input file descriptor for us to return. In this case,
3093                  * reject the attempt.
3094                  */
3095                 if (ifl_ret != NULL) {
3096                         _rej.rej_type = SGS_REJ_ARCHIVE;
3097                         _rej.rej_name = name;
3098                         DBG_CALL(Dbg_file_rejected(ofl->ofl_lml, &_rej,
3099                             ld_targ.t_m.m_mach));
3100                         if (rej->rej_type == 0) {
3101                                 *rej = _rej;
3102                                 rej->rej_name = strdup(_rej.rej_name);
3103                         }
3104                         return (0);
3105                 }

3107                 /*
3108                  * Determine if we've already come across this archive file.
3109                  */
3110                 if (!(flags & FLG_IF_EXTRACT)) {
3111                         Aliste  idx;

3113                         for (APLIST_TRAVERSE(ofl->ofl_ars, idx, adp)) {
3114                                 if ((adp->ad_stdev != status.st_dev) ||
3115                                     (adp->ad_stino != status.st_ino))
3116                                         continue;

3118                                 /*
3119                                  * We've seen this file before so reuse the
3120                                  * original archive descriptor and discard the
3121                                  * new elf descriptor.  Note that a file
3122                                  * descriptor is unnecessary, as the file is
3123                                  * already available in memory.
3124                                  */
3125                                 DBG_CALL(Dbg_file_reuse(ofl->ofl_lml, name,
3126                                     adp->ad_name));
3127                                 (void) elf_end(elf);
3128                                 if (!ld_process_archive(name, -1, adp, ofl))
3129                                         return (S_ERROR);
3130                                 return (1);
3131                         }
3132                 }

3134                 /*
3135                  * As we haven't processed this file before establish a new
3136                  * archive descriptor.
3137                  */
3138                 adp = ld_ar_setup(name, elf, ofl);
3139                 if ((adp == NULL) || (adp == (Ar_desc *)S_ERROR))
3140                         return ((uintptr_t)adp);
3141                 adp->ad_stdev = status.st_dev;
3142                 adp->ad_stino = status.st_ino;

3144                 ld_sup_file(ofl, name, ELF_K_AR, flags, elf);

3146                 /*
3147                  * Indicate that the ELF descriptor no longer requires a file
```

```
3148                  * descriptor by reading the entire file.  The file is already
3149                  * read via the initial mmap(2) behind elf_begin(3elf), thus
3150                  * this operation is effectively a no-op.  However, a side-
3151                  * effect is that the internal file descriptor, maintained in
3152                  * the ELF descriptor, is set to -1.  This setting will not
3153                  * be compared with any file descriptor that is passed to
3154                  * elf_begin(), should this archive, or one of the archive
3155                  * members, be processed again from the command line or
3156                  * because of a -z rescan.
3157                  */
3158                 if (elf_cntl(elf, ELF_C_FDREAD) == -1) {
3159                         ld_eprintf(ofl, ERR_ELF, MSG_INTL(MSG_ELF_CNTL),
3160                             name);
3161                         return (0);
3162                 }

3164                 if (!ld_process_archive(name, -1, adp, ofl))
3165                         return (S_ERROR);
3166                 return (1);

3168         case ELF_K_ELF:
3169                 /*
3170                  * Obtain the elf header so that we can determine what type of
3171                  * elf ELF_K_ELF file this is.
3172                  */
3173                 if ((ehdr = elf_getehdr(elf)) == NULL) {
3174                         int     _class = gelf_getclass(elf);

3176                         /*
3177                          * This can fail for a number of reasons. Typically
3178                          * the object class is incorrect (ie. user is building
3179                          * 64-bit but managed to point at 32-bit libraries).
3180                          * Other ELF errors can include a truncated or corrupt
3181                          * file. Try to get the best error message possible.
3182                          */
3183                         if (ld_targ.t_m.m_class != _class) {
3184                                 _rej.rej_type = SGS_REJ_CLASS;
3185                                 _rej.rej_info = (uint_t)_class;
3186                         } else {
3187                                 _rej.rej_type = SGS_REJ_STR;
3188                                 _rej.rej_str = elf_errmsg(-1);
3189                         }
3190                         _rej.rej_name = name;
3191                         DBG_CALL(Dbg_file_rejected(ofl->ofl_lml, &_rej,
3192                             ld_targ.t_m.m_mach));
3193                         if (rej->rej_type == 0) {
3194                                 *rej = _rej;
3195                                 rej->rej_name = strdup(_rej.rej_name);
3196                         }
3197                         return (0);
3198                 }

3200                 if (_gelf_getdynval(elf, DT_SUNW_KMOD) == 1) {
3201                         _rej.rej_name = name;
3202                         DBG_CALL(Dbg_file_rejected(ofl->ofl_lml, &_rej,
3203                             ld_targ.t_m.m_mach));
3204                         _rej.rej_type = SGS_REJ_KMOD;
3205                         _rej.rej_str = elf_errmsg(-1);

3207 #endif /* ! codereview */
3208                         if (rej->rej_type == 0) {
3209                                 *rej = _rej;
3210                                 rej->rej_name = strdup(_rej.rej_name);
3211                         }
3212                         return (0);
3213                 }
```



```
3215                 /*
3216                  * Determine if we've already come across this file.
3217                  */
3218                 if (!(flags & FLG_IF_EXTRACT)) {
3219                         APlist  *apl;
3220                         Aliste  idx;

3222                         if (ehdr->e_type == ET_REL)
3223                                 apl = ofl->ofl_objs;
3224                         else
3225                                 apl = ofl->ofl_sos;

3227                         /*
3228                          * Traverse the appropriate file list and determine if
3229                          * a dev/inode match is found.
3230                          */
3231                         for (APLIST_TRAVERSE(apl, idx, ifl)) {
3232                                 /*
3233                                  * Ifl_desc generated via -Nneed, therefore no
3234                                  * actual file behind it.
3235                                  */
3236                                 if (ifl->ifl_flags & FLG_IF_NEEDSTR)
3237                                         continue;

3239                                 if ((ifl->ifl_stino != status.st_ino) ||
3240                                     (ifl->ifl_stdev != status.st_dev))
3241                                         continue;

3243                                 /*
3244                                  * Disregard (skip) this image.
3245                                  */
3246                                 DBG_CALL(Dbg_file_skip(ofl->ofl_lml,
3247                                     ifl->ifl_name, name));
3248                                 (void) elf_end(elf);

3250                                 /*
3251                                  * If the file was explicitly defined on the
3252                                  * command line (this is always the case for
3253                                  * relocatable objects, and is true for shared
3254                                  * objects when they weren't specified via -l or
3255                                  * were dragged in as an implicit dependency),
3256                                  * then warn the user.
3257                                  */
3258                                 if ((flags & FLG_IF_CMDLINE) ||
3259                                     (ifl->ifl_flags & FLG_IF_CMDLINE)) {
3260                                         const char      *errmsg;

3262                                         /*
3263                                          * Determine whether this is the same
3264                                          * file name as originally encountered
3265                                          * so as to provide the most
3266                                          * descriptive diagnostic.
3267                                          */
3268                                         errmsg =
3269                                             (strcmp(name, ifl->ifl_name) == 0) ?
3270                                             MSG_INTL(MSG_FIL_MULINC_1) :
3271                                             MSG_INTL(MSG_FIL_MULINC_2);
3272                                         ld_eprintf(ofl, ERR_WARNING,
3273                                             errmsg, name, ifl->ifl_name);
3274                                 }
3275                                 if (ifl_ret)
3276                                         *ifl_ret = ifl;
3277                                 return (1);
3278                         }
3279                 }
```

```
3281                 /*
3282                  * At this point, we know we need the file.  Establish an input
3283                  * file descriptor and continue processing.
3284                  */
3285                 ifl = ifl_setup(name, ehdr, elf, flags, ofl, rej);
3286                 if ((ifl == NULL) || (ifl == (Ifl_desc *)S_ERROR))
3287                         return ((uintptr_t)ifl);
3288                 ifl->ifl_stdev = status.st_dev;
3289                 ifl->ifl_stino = status.st_ino;

3291                 /*
3292                  * If -zignore is in effect, mark this file as a potential
3293                  * candidate (the files use isn't actually determined until
3294                  * symbol resolution and relocation processing are completed).
3295                  */
3296                 if (ofl->ofl_flags1 & FLG_OF1_IGNORE)
3297                         ifl->ifl_flags |= FLG_IF_IGNORE;

3299                 switch (ehdr->e_type) {
3300                 case ET_REL:
3301                         (*ld_targ.t_mr.mr_mach_eflags)(ehdr, ofl);
3302                         error = process_elf(ifl, elf, ofl);
3303                         break;
3304                 case ET_DYN:
3305                         if ((ofl->ofl_flags & FLG_OF_STATIC) ||
3306                             !(ofl->ofl_flags & FLG_OF_DYNLIBS)) {
3307                                 ld_eprintf(ofl, ERR_FATAL,
3308                                     MSG_INTL(MSG_FIL_SOINSTAT), name);
3309                                 return (0);
3310                         }

3312                         /*
3313                          * Record any additional shared object information.
3314                          * If no soname is specified (eg. this file was
3315                          * derived from a explicit filename declaration on the
3316                          * command line, ie. bar.so) use the pathname.
3317                          * This entry may be overridden if the files dynamic
3318                          * section specifies an DT_SONAME value.
3319                          */
3320                         if (soname == NULL)
3321                                 ifl->ifl_soname = ifl->ifl_name;
3322                         else
3323                                 ifl->ifl_soname = soname;

3325                         /*
3326                          * If direct bindings, lazy loading, group permissions,
3327                          * or deferred dependencies need to be established, mark
3328                          * this object.
3329                          */
3330                         if (ofl->ofl_flags1 & FLG_OF1_ZDIRECT)
3331                                 ifl->ifl_flags |= FLG_IF_DIRECT;
3332                         if (ofl->ofl_flags1 & FLG_OF1_LAZYLD)
3333                                 ifl->ifl_flags |= FLG_IF_LAZYLD;
3334                         if (ofl->ofl_flags1 & FLG_OF1_GRPPRM)
3335                                 ifl->ifl_flags |= FLG_IF_GRPPRM;
3336                         if (ofl->ofl_flags1 & FLG_OF1_DEFERRED)
3337                                 ifl->ifl_flags |=
3338                                     (FLG_IF_LAZYLD | FLG_IF_DEFERRED);

3340                         error = process_elf(ifl, elf, ofl);

3342                         /*
3343                          * Determine whether this dependency requires a syminfo.
3344                          */
3345                         if (ifl->ifl_flags & MSK_IF_SYMINFO)
```



```
3346                                 ofl->ofl_flags |= FLG_OF_SYMINFO;

3348                         /*
3349                          * Guidance: Use -z lazyload/nolazyload.
3350                          * libc is exempt from this advice, because it cannot
3351                          * be lazy loaded, and requests to do so are ignored.
3352                          */
3353                         if (OFL_GUIDANCE(ofl, FLG_OFG_NO_LAZY) &&
3354                             ((ifl->ifl_flags & FLG_IF_RTLDINF) == 0)) {
3355                                 ld_eprintf(ofl, ERR_GUIDANCE,
3356                                     MSG_INTL(MSG_GUIDE_LAZYLOAD));
3357                                 ofl->ofl_guideflags |= FLG_OFG_NO_LAZY;
3358                         }

3360                         /*
3361                          * Guidance: Use -B direct/nodirect or
3362                          * -z direct/nodirect.
3363                          */
3364                         if (OFL_GUIDANCE(ofl, FLG_OFG_NO_DB)) {
3365                                 ld_eprintf(ofl, ERR_GUIDANCE,
3366                                     MSG_INTL(MSG_GUIDE_DIRECT));
3367                                 ofl->ofl_guideflags |= FLG_OFG_NO_DB;
3368                         }

3370                         break;
3371                 default:
3372                         (void) elf_errno();
3373                         _rej.rej_type = SGS_REJ_UNKFILE;
3374                         _rej.rej_name = name;
3375                         DBG_CALL(Dbg_file_rejected(ofl->ofl_lml, &_rej,
3376                             ld_targ.t_m.m_mach));
3377                         if (rej->rej_type == 0) {
3378                                 *rej = _rej;
3379                                 rej->rej_name = strdup(_rej.rej_name);
3380                         }
3381                         return (0);
3382                 }
3383                 break;
3384         default:
3385                 (void) elf_errno();
3386                 _rej.rej_type = SGS_REJ_UNKFILE;
3387                 _rej.rej_name = name;
3388                 DBG_CALL(Dbg_file_rejected(ofl->ofl_lml, &_rej,
3389                     ld_targ.t_m.m_mach));
3390                 if (rej->rej_type == 0) {
3391                         *rej = _rej;
3392                         rej->rej_name = strdup(_rej.rej_name);
3393                 }
3394                 return (0);
3395         }
3396         if ((error == 0) || (error == S_ERROR))
3397                 return (error);

3399         if (ifl_ret)
3400                 *ifl_ret = ifl;
3401         return (1);
3402 }

3404 /*
3405  * Having successfully opened a file, set up the necessary elf structures to
3406  * process it further.  This small section of processing is slightly different
3407  * from the elf initialization required to process a relocatable object from an
3408  * archive (see libs.c: ld_process_archive()).
3409  */
3410 uintptr_t
3411 ld_process_open(const char *opath, const char *ofile, int *fd, Ofl_desc *ofl,
```

```
3412     Word flags, Rej_desc *rej, Ifl_desc **ifl_ret)
3413 {
3414         Elf             *elf;
3415         const char      *npath = opath;
3416         const char      *nfile = ofile;

3418         if ((elf = elf_begin(*fd, ELF_C_READ, NULL)) == NULL) {
3419                 ld_eprintf(ofl, ERR_ELF, MSG_INTL(MSG_ELF_BEGIN), npath);
3420                 return (0);
3421         }

3423         /*
3424          * Determine whether the support library wishes to process this open.
3425          * The support library may return:
3426          *    .  a different ELF descriptor (in which case they should have
3427          *       closed the original)
3428          *    .  a different file descriptor (in which case they should have
3429          *       closed the original)
3430          *    .  a different path and file name (presumably associated with
3431          *       a different file descriptor)
3432          *
3433          * A file descriptor of -1, or and ELF descriptor of zero indicates
3434          * the file should be ignored.
3435          */
3436         ld_sup_open(ofl, &npath, &nfile, fd, flags, &elf, NULL, 0,
3437             elf_kind(elf));

3439         if ((*fd == -1) || (elf == NULL))
3440                 return (0);

3442         return (ld_process_ifl(npath, nfile, *fd, elf, flags, ofl, rej,
3443             ifl_ret));
3444 }

3446 /*
3447  * Having successfully mapped a file, set up the necessary elf structures to
3448  * process it further.  This routine is patterned after ld_process_open() and
3449  * is only called by ld.so.1(1) to process a relocatable object.
3450  */
3451 Ifl_desc *
3452 ld_process_mem(const char *path, const char *file, char *addr, size_t size,
3453     Ofl_desc *ofl, Rej_desc *rej)
3454 {
3455         Elf             *elf;
3456         uintptr_t       open_ret;
3457         Ifl_desc        *ifl;

3459         if ((elf = elf_memory(addr, size)) == NULL) {
3460                 ld_eprintf(ofl, ERR_ELF, MSG_INTL(MSG_ELF_MEMORY), path);
3461                 return (0);
3462         }

3464         open_ret = ld_process_ifl(path, file, 0, elf, 0, ofl, rej, &ifl);
3465         if (open_ret != 1)
3466                 return ((Ifl_desc *) open_ret);
3467         return (ifl);
3468 }

3470 /*
3471  * Process a required library (i.e. the dependency of a shared object).
3472  * Combine the directory and filename, check the resultant path size, and try
3473  * opening the pathname.
3474  */
3475 static Ifl_desc *
3476 process_req_lib(Sdf_desc *sdf, const char *dir, const char *file,
3477     Ofl_desc *ofl, Rej_desc *rej)
```



```
3478 {
3479         size_t          dlen, plen;
3480         int             fd;
3481         char            path[PATH_MAX];
3482         const char      *_dir = dir;

3484         /*
3485          * Determine the sizes of the directory and filename to insure we don't
3486          * exceed our buffer.
3487          */
3488         if ((dlen = strlen(dir)) == 0) {
3489                 _dir = MSG_ORIG(MSG_STR_DOT);
3490                 dlen = 1;
3491         }
3492         dlen++;
3493         plen = dlen + strlen(file) + 1;
3494         if (plen > PATH_MAX) {
3495                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_FIL_PTHTOLONG),
3496                     _dir, file);
3497                 return (0);
3498         }

3500         /*
3501          * Build the entire pathname and try and open the file.
3502          */
3503         (void) strcpy(path, _dir);
3504         (void) strcat(path, MSG_ORIG(MSG_STR_SLASH));
3505         (void) strcat(path, file);
3506         DBG_CALL(Dbg_libs_req(ofl->ofl_lml, sdf->sdf_name,
3507             sdf->sdf_rfile, path));

3509         if ((fd = open(path, O_RDONLY)) == -1)
3510                 return (0);
3511         else {
3512                 uintptr_t       open_ret;
3513                 Ifl_desc        *ifl;
3514                 char            *_path;

3516                 if ((_path = libld_malloc(strlen(path) + 1)) == NULL)
3517                         return ((Ifl_desc *)S_ERROR);
3518                 (void) strcpy(_path, path);
3519                 open_ret = ld_process_open(_path, &_path[dlen], &fd, ofl,
3520                     0, rej, &ifl);
3521                 if (fd != -1)
3522                         (void) close(fd);
3523                 if (open_ret != 1)
3524                         return ((Ifl_desc *)open_ret);
3525                 return (ifl);
3526         }
3527 }

3529 /*
3530  * Finish any library processing.  Walk the list of so's that have been listed
3531  * as "included" by shared objects we have previously processed.  Examine them,
3532  * without adding them as explicit dependents of this program, in order to
3533  * complete our symbol definition process.  The search path rules are:
3534  *
3535  *  -   use any user supplied paths, i.e. LD_LIBRARY_PATH and -L, then
3536  *
3537  *  -   use any RPATH defined within the parent shared object, then
3538  *
3539  *  -   use the default directories, i.e. LIBPATH or -YP.
3540  */
3541 uintptr_t
3542 ld_finish_libs(Ofl_desc *ofl)
3543 {
```

```
3544        Aliste          idx1;
3545        Sdf_desc        *sdf;
3546        Rej_desc        rej = { 0 };

3548        /*
3549         * Make sure we are back in dynamic mode.
3550         */
3551        ofl->ofl_flags |= FLG_OF_DYNLIBS;

3553        for (APLIST_TRAVERSE(ofl->ofl_soneed, idx1, sdf)) {
3554                Aliste          idx2;
3555                char            *path, *slash = NULL;
3556                int             fd;
3557                Ifl_desc        *ifl;
3558                char            *file = (char *)sdf->sdf_name;

3560                /*
3561                 * See if this file has already been processed.  At the time
3562                 * this implicit dependency was determined there may still have
3563                 * been more explicit dependencies to process.  Note, if we ever
3564                 * do parse the command line three times we would be able to
3565                 * do all this checking when processing the dynamic section.
3566                 */
3567                if (sdf->sdf_file)
3568                        continue;

3570                for (APLIST_TRAVERSE(ofl->ofl_sos, idx2, ifl)) {
3571                        if (!(ifl->ifl_flags & FLG_IF_NEEDSTR) &&
3572                            (strcmp(file, ifl->ifl_soname) == 0)) {
3573                                sdf->sdf_file = ifl;
3574                                break;
3575                        }
3576                }
3577                if (sdf->sdf_file)
3578                        continue;

3580                /*
3581                 * If the current path name element embeds a "/", then it's to
3582                 * be taken "as is", with no searching involved.  Process all
3583                 * "/" occurrences, so that we can deduce the base file name.
3584                 */
3585                for (path = file; *path; path++) {
3586                        if (*path == '/')
3587                                slash = path;
3588                }
3589                if (slash) {
3590                        DBG_CALL(Dbg_libs_req(ofl->ofl_lml, sdf->sdf_name,
3591                            sdf->sdf_rfile, file));
3592                        if ((fd = open(file, O_RDONLY)) == -1) {
3593                                ld_eprintf(ofl, ERR_WARNING,
3594                                    MSG_INTL(MSG_FIL_NOTFOUND), file,
3595                                    sdf->sdf_rfile);
3596                        } else {
3597                                uintptr_t       open_ret;
3598                                Rej_desc        _rej = { 0 };

3600                                open_ret = ld_process_open(file, ++slash,
3601                                    &fd, ofl, 0, &_rej, &ifl);
3602                                if (fd != -1)
3603                                        (void) close(fd);
3604                                if (open_ret == S_ERROR)
3605                                        return (S_ERROR);

3607                                if (_rej.rej_type) {
3608                                        Conv_reject_desc_buf_t rej_buf;
```



```
3610                                        ld_eprintf(ofl, ERR_WARNING,
3611                                            MSG_INTL(reject[_rej.rej_type]),
3612                                            _rej.rej_name ? rej.rej_name :
3613                                            MSG_INTL(MSG_STR_UNKNOWN),
3614                                            conv_reject_desc(&_rej, &rej_buf,
3615                                            ld_targ.t_m.m_mach));
3616                                } else
3617                                        sdf->sdf_file = ifl;
3618                        }
3619                        continue;
3620                }

3622                /*
3623                 * Now search for this file in any user defined directories.
3624                 */
3625                for (APLIST_TRAVERSE(ofl->ofl_ulibdirs, idx2, path)) {
3626                        Rej_desc        _rej = { 0 };

3628                        ifl = process_req_lib(sdf, path, file, ofl, &_rej);
3629                        if (ifl == (Ifl_desc *)S_ERROR) {
3630                                return (S_ERROR);
3631                        }
3632                        if (_rej.rej_type) {
3633                                if (rej.rej_type == 0) {
3634                                        rej = _rej;
3635                                        rej.rej_name = strdup(_rej.rej_name);
3636                                }
3637                        }
3638                        if (ifl) {
3639                                sdf->sdf_file = ifl;
3640                                break;
3641                        }
3642                }
3643                if (sdf->sdf_file)
3644                        continue;

3646                /*
3647                 * Next use the local rules defined within the parent shared
3648                 * object.
3649                 */
3650                if (sdf->sdf_rpath != NULL) {
3651                        char    *rpath, *next;

3653                        rpath = libld_malloc(strlen(sdf->sdf_rpath) + 1);
3654                        if (rpath == NULL)
3655                                return (S_ERROR);
3656                        (void) strcpy(rpath, sdf->sdf_rpath);
3657                        DBG_CALL(Dbg_libs_path(ofl->ofl_lml, rpath,
3658                            LA_SER_RUNPATH, sdf->sdf_rfile));
3659                        if ((path = strtok_r(rpath,
3660                            MSG_ORIG(MSG_STR_COLON), &next)) != NULL) {
3661                                do {
3662                                        Rej_desc        _rej = { 0 };

3664                                        path = expand(sdf->sdf_rfile, path,
3665                                            &next);

3667                                        ifl = process_req_lib(sdf, path,
3668                                            file, ofl, &_rej);
3669                                        if (ifl == (Ifl_desc *)S_ERROR) {
3670                                                return (S_ERROR);
3671                                        }
3672                                        if ((_rej.rej_type) &&
3673                                            (rej.rej_type == 0)) {
3674                                                rej = _rej;
3675                                                rej.rej_name =
```

```
3676                                                    strdup(_rej.rej_name);
3677                                    }
3678                                    if (ifl) {
3679                                            sdf->sdf_file = ifl;
3680                                            break;
3681                                    }
3682                            } while ((path = strtok_r(NULL,
3683                                MSG_ORIG(MSG_STR_COLON), &next)) != NULL);
3684                    }
3685            }
3686            if (sdf->sdf_file)
3687                    continue;

3689            /*
3690             * Finally try the default library search directories.
3691             */
3692            for (APLIST_TRAVERSE(ofl->ofl_dlibdirs, idx2, path)) {
3693                    Rej_desc        _rej = { 0 };

3695                    ifl = process_req_lib(sdf, path, file, ofl, &rej);
3696                    if (ifl == (Ifl_desc *)S_ERROR) {
3697                            return (S_ERROR);
3698                    }
3699                    if (_rej.rej_type) {
3700                            if (rej.rej_type == 0) {
3701                                    rej = _rej;
3702                                    rej.rej_name = strdup(_rej.rej_name);
3703                            }
3704                    }
3705                    if (ifl) {
3706                            sdf->sdf_file = ifl;
3707                            break;
3708                    }
3709            }
3710            if (sdf->sdf_file)
3711                    continue;

3713            /*
3714             * If we've got this far we haven't found the shared object.
3715             * If an object was found, but was rejected for some reason,
3716             * print a diagnostic to that effect, otherwise generate a
3717             * generic "not found" diagnostic.
3718             */
3719            if (rej.rej_type) {
3720                    Conv_reject_desc_buf_t rej_buf;

3722                    ld_eprintf(ofl, ERR_WARNING,
3723                        MSG_INTL(reject[rej.rej_type]),
3724                        rej.rej_name ? rej.rej_name :
3725                        MSG_INTL(MSG_STR_UNKNOWN),
3726                        conv_reject_desc(&rej, &rej_buf,
3727                        ld_targ.t_m.m_mach));
3728            } else {
3729                    ld_eprintf(ofl, ERR_WARNING,
3730                        MSG_INTL(MSG_FIL_NOTFOUND), file, sdf->sdf_rfile);
3731            }
3732    }

3734    /*
3735     * Finally, now that all objects have been input, make sure any version
3736     * requirements have been met.
3737     */
3738    return (ld_vers_verify(ofl));
3739 }
```

```
**********************************************************
    4129 Sun Feb 24 19:19:11 2019
new/usr/src/cmd/sgs/libld/common/globals.c
ld should reject kernel modules as input
**********************************************************
```

```
/*
 * 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.
 */

/*
 * Global variables
 */
#include        <sys/elf.h>
#include        "msg.h"
#include        "_libld.h"

Ld_heap         *ld_heap;       /* list of allocated blocks for */
                                /*      link-edit dynamic allocations */
APlist          *lib_support;   /* list of support libraries specified */
                                /*      (-S option) */
int             demangle_flag;  /* symbol demangling required */

/*
 * Paths and directories for library searches.  These are used to set up
 * linked lists of directories which are maintained in the ofl structure.
 */
char            *Plibpath;      /* User specified -YP or defaults to LIBPATH */
char            *Llibdir;       /* User specified -YL */
char            *Ulibdir;       /* User specified -YU */

/*
 * A default library search path is used if one was not supplied on the command
 * line.  Note: these strings can not use MSG_ORIG() since they are modified as
 * part of the path processing.
 */
char            def64_Plibpath[] = "/lib/64:/usr/lib/64";
char            def32_Plibpath[] = "/usr/ccs/lib:/lib:/usr/lib";

/*
 * Rejected file error messages (indexed to match SGS_REJ_ values).
 */
const Msg
```



```
reject[SGS_REJ_NUM] = {
                MSG_STR_EMPTY,
                MSG_REJ_MACH,           /* MSG_INTL(MSG_REJ_MACH) */
                MSG_REJ_CLASS,          /* MSG_INTL(MSG_REJ_CLASS) */
                MSG_REJ_DATA,           /* MSG_INTL(MSG_REJ_DATA) */
                MSG_REJ_TYPE,           /* MSG_INTL(MSG_REJ_TYPE) */
                MSG_REJ_BADFLAG,        /* MSG_INTL(MSG_REJ_BADFLAG) */
                MSG_REJ_MISFLAG,        /* MSG_INTL(MSG_REJ_MISFLAG) */
                MSG_REJ_VERSION,        /* MSG_INTL(MSG_REJ_VERSION) */
                MSG_REJ_HAL,            /* MSG_INTL(MSG_REJ_HAL) */
                MSG_REJ_US3,            /* MSG_INTL(MSG_REJ_US3) */
                MSG_REJ_STR,            /* MSG_INTL(MSG_REJ_STR) */
                MSG_REJ_UNKFILE,        /* MSG_INTL(MSG_REJ_UNKFILE) */
                MSG_REJ_UNKCAP,         /* MSG_INTL(MSG_REJ_UNKCAP) */
                MSG_REJ_HWCAP_1,        /* MSG_INTL(MSG_REJ_HWCAP_1) */
                MSG_REJ_SFCAP_1,        /* MSG_INTL(MSG_REJ_SFCAP_1) */
                MSG_REJ_MACHCAP,        /* MSG_INTL(MSG_REJ_MACHCAP) */
                MSG_REJ_PLATCAP,        /* MSG_INTL(MSG_REJ_PLATCAP) */
                MSG_REJ_HWCAP_2,        /* MSG_INTL(MSG_REJ_HWCAP_2) */
                MSG_REJ_ARCHIVE,        /* MSG_INTL(MSG_REJ_ARCHIVE) */
                MSG_REJ_KMOD            /* MSG_INTL(MSG_REJ_KMOD) */
                MSG_REJ_ARCHIVE         /* MSG_INTL(MSG_REJ_ARCHIVE) */
        };
#if SGS_REJ_NUM != (SGS_REJ_KMOD + 1)
#if SGS_REJ_NUM != (SGS_REJ_ARCHIVE + 1)
#error SGS_REJ_NUM has changed
#endif

/*
 * Symbol types that we include in .SUNW_ldynsym sections
 * (indexed by STT_ values).
 */
const int
ldynsym_symtype[] = {
                0,                      /* STT_NOTYPE (not counting 1st slot) */
                0,                      /* STT_OBJECT */
                1,                      /* STT_FUNC */
                0,                      /* STT_SECTION */
                1,                      /* STT_FILE */
                0,                      /* STT_COMMON */
                0,                      /* STT_TLS */
                0,                      /* 7 */
                0,                      /* 8 */
                0,                      /* 9 */
                0,                      /* 10 */
                0,                      /* 11 */
                0,                      /* 12 */
                0,                      /* STT_SPARC_REGISTER */
                0,                      /* 14 */
                0,                      /* 15 */
};
______unchanged_portion_omitted_
```

```
**********************************************************
   60944 Sun Feb 24 19:19:11 2019
new/usr/src/cmd/sgs/libld/common/libld.msg
ld should reject kernel modules as input
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)
ld: implement -ztype and rework option parsing
**********************************************************
```

```
#
# 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\
```

```
124                          \t\t\tspecify a link-edit support library\n"
125 @ 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\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_ZTY    "\t[-z type=type]\tspecify the type of object \
255                          (exec, kmod, reloc, shared)\n"
```

```
#endif /* ! codereview */
@ MSG_ARG_DETAIL_ZWRAP  "\t[-z wrap=symbol], [-wrap=symbol], [--wrap=symbol]\n\
                         \t\t\twrap symbol references\n"
@ MSG_ARG_DETAIL_ZVER   "\t[-z verbose]\t\
                         generate warnings for suspicious processings\n"

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

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

@ MSG_REL_NOFIT         "relocation error: %s: file %s: symbol %s: \
                         value 0x%llx does not fit"
@ MSG_REL_NONALIGN      "relocation error: %s: file %s: symbol %s: \
                         offset 0x%llx is non-aligned"
@ MSG_REL_NULL          "relocation error: file %s: section [%u]%s: \
                         skipping null relocation record"
@ MSG_REL_NOTSUP        "relocation error: %s: file %s: section [%u]%s: \
                         relocation not currently supported"
@ MSG_REL_PICREDLOC     "relocation error: %s: file %s symbol %s: \
                         -z redlocsym may not be used for pic code"
@ MSG_REL_TLSLE         "relocation error: %s: file %s: symbol %s: \
                         relocation illegal when building a shared object"
@ MSG_REL_TLSBND        "relocation error: %s: file %s: symbol %s: \
                         bound to: %s: relocation illegal when not bound \
                         to object being created"
@ MSG_REL_TLSSTAT       "relocation error: %s: file %s: symbol %s: \
                         relocation illegal when building a static object"
@ MSG_REL_TLSBADSYM     "relocation error: %s: file %s: symbol %s: \
                         bad symbol type %s: symbol type must be TLS"
@ MSG_REL_BADTLS        "relocation error: %s: file %s: symbol %s: \
                         relocation illegal for TLS symbol"
@ MSG_REL_BADGOTBASED   "relocation error: %s: file %s: symbol %s: a GOT \
                         relative relocation must reference a local symbol"
@ MSG_REL_UNKNWSYM      "relocation error: %s: file %s: section [%u]%s: \
                         attempt to relocate with respect to unknown \
                         symbol %s: offset 0x%llx, symbol index %d"
@ MSG_REL_UNSUPSZ       "relocation error: %s: file %s: symbol %s: \
                         offset size (%d bytes) is not supported"
@ MSG_REL_INVALOFFSET   "relocation error: %s: file %s section [%u]%s: \
                         invalid offset symbol '%s': offset 0x%llx"
@ MSG_REL_INVALRELT     "relocation error: file %s: section [%u]%s: \
                         invalid relocation type: 0x%x"
@ MSG_REL_EMPTYSEC      "relocation error: %s: file %s: symbol %s: \
                         attempted against empty section [%u]%s"
@ MSG_REL_EXTERNSYM     "relocation error: %s: file %s: symbol %s: \
                         external symbolic relocation against non-allocatable \
                         section %s; cannot be processed at runtime: \
                         relocation ignored"
@ MSG_REL_UNEXPREL      "relocation error: %s: file %s: symbol %s: \
                         unexpected relocation; generic processing performed"
@ MSG_REL_UNEXPSYM      "relocation error: %s: file %s: symbol %s: \
                         unexpected symbol referenced from file %s"
@ MSG_REL_SYMDISC       "relocation error: %s: file %s: section [%u]%s: \
                         symbol %s: symbol has been discarded with discarded \
                         section: [%u]%s"
@ MSG_REL_NOSYMBOL      "relocation error: %s: file %s: section: [%u]%s: \
                         offset: 0x%llx: relocation requires reference symbol"
@ MSG_REL_DISPREL1      "relocation error: %s: file %s: symbol %s: \
                         displacement relocation applied to the symbol \
                         %s at 0x%llx: symbol %s is a copy relocated symbol"
```



```
@ MSG_REL_UNSUPSIZE     "relocation error: %s: file %s: section [%u]%s: \
                         relocation against section symbol unsupported"

@ MSG_REL_DISPREL2      "relocation warning: %s: file %s: symbol %s: \
                         may contain displacement relocation"
@ MSG_REL_DISPREL3      "relocation warning: %s: file %s: symbol %s: \
                         displacement relocation applied to the symbol \
                         %s: at 0x%llx: displacement relocation will not be \
                         visible in output image"
@ MSG_REL_DISPREL4      "relocation warning: %s: file %s: symbol %s: \
                         displacement relocation to be applied to the symbol \
                         %s: at 0x%llx: displacement relocation will be \
                         visible in output image"
@ MSG_REL_COPY          "relocation warning: %s: file %s: symbol %s: \
                         relocation bound to a symbol with STV_PROTECTED \
                         visibility"
@ MSG_RELINVSEC         "relocation warning: %s: file %s: section: [%u]%s: \
                         against suspicious section [%u]%s; relocation ignored"
@ MSG_REL_TLSIE         "relocation warning: %s: file %s: symbol %s: \
                         relocation has restricted use when building a shared \
                         object"

@ MSG_REL_SLOPCDATNONAM "relocation warning: %s: file %s: section [%u]%s: \
                         relocation against discarded COMDAT section [%u]%s: \
                         redirected to file %s"
@ MSG_REL_SLOPCDATNAM   "relocation warning: %s: file %s: section [%u]%s: \
                         symbol %s: relocation against discarded COMDAT \
                         section [%u]%s: redirected to file %s"
@ MSG_REL_SLOPCDATNOSYM "relocation warning: %s: file %s: section [%u]%s: \
                         symbol %s: relocation against discarded COMDAT \
                         section [%u]%s: symbol not found, relocation ignored"

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

#
# TRANSLATION_NOTE
#       The following 7 messages are the message to print the
#       following example messages.
#
#Text relocation remains                        referenced
#    against symbol                offset      in file
#str                                0x14        main.o
#printf                             0x1c        main.o
#
#       The first two lines are the header, and the next msgid
#       is the format string for the header.
#       Tabs and spaces are used for alignment.
#       The first and third %s are for: "Text relocation remains against symbol"
#       The second %s and fourth %s are for: "referenced in file"
#       The third %s is for: "offset"
#
@ MSG_REL_REMAIN_FMT_1  "%-40s\t%s\n    %s\t\t    %s\t%s"
#
# TRANSLATION_NOTE
#       The next two msdid make a sentence. So translate:
#               "Text relocation remain against symbol"
#       And separate them into two msgstr considering the proper
#       alignment.
@ MSG_REL_RMN_ITM_11    "Text relocation remains"
@ MSG_REL_RMN_ITM_12    "against symbol"
@ MSG_REL_RMN_ITM_13    "warning: Text relocation remains"

@ MSG_REL_RMN_ITM_2     "offset"

#
```

```
388 # TRANSLATION_NOTE
389 #       The next two msdid make a sentence. So translate:
390 #               "referenced in file"
391 #       And separate them into two msgstr considering the proper
392 #       alignment.
393 @ MSG_REL_RMN_ITM_31    "referenced"
394 @ MSG_REL_RMN_ITM_32    "in file"
395 @ MSG_REL_REMAIN_2      "%-35s 0x%-8llx\t%s"
396 @ MSG_REL_REMAIN_3      "relocations remain against allocatable but \
397                          non-writable sections"

399 # Files processing messages

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

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

436 # Recording name conflicts

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

444 # System call messages

446 @ MSG_SYS_OPEN          "file %s: open failed: %s"
447 @ MSG_SYS_UNLINK        "file %s: unlink failed: %s"
448 @ MSG_SYS_MMAPANON      "mmap anon failed: %s"
449 @ MSG_SYS_MALLOC        "malloc failed: %s"


452 # Messages related to platform support
```



```
454 @ MSG_TARG_UNSUPPORTED  "unsupported ELF machine type: %s"


457 # ELF processing messages

459 @ MSG_ELF_LIBELF        "libelf: version not supported: %d"

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

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

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

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

472 # Section processing errors

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

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

480 @ MSG_SCN_DWFOVRFLW     "%s: section %s: encoded DWARF data exceeds \
481                          section size"
482 @ MSG_SCN_DWFBADENC     "%s: section %s: invalid DWARF encoding: %#x"

484 # Symbol processing errors

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

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

```
@ MSG_PSYM_INVMINFO1    "file %s: section [%u]%s: entry[%d] has invalid m_info:
                         0x%llx for symbol index"
@ MSG_PSYM_INVMINFO2    "file %s: section [%u]%s: entry[%d] has invalid m_info:
                         0x%llx for size"
@ MSG_PSYM_INVMREPEAT   "file %s: section [%u]%s: entry[%d] has invalid m_repeat
                         0x%llx"
@ MSG_PSYM_CANNOTEXPND  "file %s: section [%u]%s: entry[%d] can not be expanded:
                         associated symbol size is unknown %s"
@ MSG_PSYM_NOSTATIC     "and partial initialization cannot be deferred to \
                         a static object"
@ MSG_MOVE_OVERLAP      "file %s: section [%u]%s: symbol '%s' overlapping move \
                         initialization: start=0x%llx, length=0x%llx: \
                         start=0x%llx, length=0x%llx"
@ MSG_PSYM_EXPREASON1   "output file is static object"
@ MSG_PSYM_EXPREASON2   "-z nopartial option in effect"
@ MSG_PSYM_EXPREASON3   "move infrastructure size is greater than move data"

#
# Support library failures
#
@ MSG_SUP_NOLOAD        "dlopen() of support library (%s) failed with \
                         error: %s"
@ MSG_SUP_BADVERSION    "initialization of support library (%s) failed with \
                         bad version.  supported: %d returned: %d"


#
# TRANSLATION_NOTE
#       The following 7 messages are the message to print the
#       following example messages.
#
#Undefined                      first referenced
# symbol                            in file
#inquire                            halt_hold.o
#
@ MSG_SYM_FMT_UNDEF     "%s\t\t\t%s\
                         \n %s  \t\t\t    %s"

#
# TRANSLATION_NOTE
#       The next two msdid make a sentence. So translate:
#               "Undefined symbol"
#       And separate them into two msgstr considering the proper
#       alignment.
@ MSG_SYM_UNDEF_ITM_11  "Undefined"
@ MSG_SYM_UNDEF_ITM_12  "symbol"
#
# TRANSLATION_NOTE
#       The next two msdid make a sentence. So translate:
#               "first referenced in file"
#       And separate them into two msgstr considering the proper
#       alignment.
@ MSG_SYM_UNDEF_ITM_21  "first referenced"
@ MSG_SYM_UNDEF_ITM_22  "in file"
#

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

@ MSG_SYM_ENTRY         "entry point"
@ MSG_SYM_UNDEF         "%s symbol '%s' is undefined"
```



```
@ MSG_SYM_EXTERN        "%s symbol '%s' is undefined  (symbol belongs to \
                         dependency %s)"
@ MSG_SYM_NOCRT         "symbol '%s' not found, but %s section exists - \
                         possible link-edit without using the compiler driver"

# Output file update messages

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

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

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


# Version processing messages

@ MSG_VER_HIGHER        "file %s: version revision %d is higher than \
                         expected %d"
@ MSG_VER_NOEXIST       "file %s: version '%s' does not exist:\n\
                         \trequired by file %s"
@ MSG_VER_UNDEF         "version '%s' undefined, referenced by version '%s':\n\
                         \trequired by file %s"
@ MSG_VER_UNAVAIL       "file %s: version '%s' is unavailable:\n\
                         \trequired by file %s"
@ MSG_VER_DEFINED       "version symbol '%s' already defined in file %s"
@ MSG_VER_INVALNDX      "version symbol '%s' from file %s has an invalid \
                         version index (%d)"
@ MSG_VER_ADDVERS       "unused $ADDVERS specification from file '%s' \
                         for object '%s'\nversion(s):"
@ MSG_VER_ADDVER        "\t%s"
@ MSG_VER_CYCLIC        "following versions generate cyclic dependency:"

# Capabilities messages

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

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

# Basic strings

@ MSG_STR_ALIGNMENTS    "alignments"
@ MSG_STR_COMMAND       "(command line)"
@ MSG_STR_TLSREL        "(internal TLS relocation requirement)"
@ MSG_STR_SIZES         "sizes"
@ MSG_STR_UNKNOWN       "<unknown>"
@ MSG_STR_SECTION       "%s (section)"
@ MSG_STR_SECTION_MSTR  "%s (merged string section)"
```

```
#
# TRANSLATION_NOTE
#       The elf_ function name represents a man page reference and should not
#       be translated.
@ MSG_ELF_BEGIN         "file %s: elf_begin"
@ MSG_ELF_CNTL          "file %s: elf_cntl"
@ MSG_ELF_GETARHDR      "file %s: elf_getarhdr"
@ MSG_ELF_GETARSYM      "file %s: elf_getarsym"
@ MSG_ELF_GETDATA       "file %s: elf_getdata"
@ MSG_ELF_GETEHDR       "file %s: elf_getehdr"
@ MSG_ELF_GETPHDR       "file %s: elf_getphdr"
@ MSG_ELF_GETSCN        "file %s: elf_getscn: scnndx: %d"
@ MSG_ELF_GETSHDR       "file %s: elf_getshdr"
@ MSG_ELF_MEMORY        "file %s: elf_memory"
@ MSG_ELF_NDXSCN        "file %s: elf_ndxscn"
@ MSG_ELF_NEWDATA       "file %s: elf_newdata"
@ MSG_ELF_NEWEHDR       "file %s: elf_newehdr"
@ MSG_ELF_NEWSCN        "file %s: elf_newscn"
@ MSG_ELF_NEWPHDR       "file %s: elf_newphdr"
@ MSG_ELF_STRPTR        "file %s: elf_strptr"
@ MSG_ELF_UPDATE        "file %s: elf_update"
@ MSG_ELF_SWAP_WRIMAGE  "file %s: _elf_swap_wrimage"


@ MSG_REJ_MACH          "file %s: wrong ELF machine type: %s"
@ MSG_REJ_CLASS         "file %s: wrong ELF class: %s"
@ MSG_REJ_DATA          "file %s: wrong ELF data format: %s"
@ MSG_REJ_TYPE          "file %s: bad ELF type: %s"
@ MSG_REJ_BADFLAG       "file %s: bad ELF flags value: %s"
@ MSG_REJ_MISFLAG       "file %s: mismatched ELF flags value: %s"
@ MSG_REJ_VERSION       "file %s: mismatched ELF/lib version: %s"
@ MSG_REJ_HAL           "file %s: HAL R1 extensions required"
@ MSG_REJ_US3           "file %s: Sun UltraSPARC III extensions required"
@ MSG_REJ_STR           "file %s: %s"
@ MSG_REJ_UNKFILE       "file %s: unknown file type"
@ MSG_REJ_UNKCAP        "file=%s; unknown capability: %d"
@ MSG_REJ_HWCAP_1       "file %s: hardware capability (CA_SUNW_HW_1) \
                         unsupported: %s"
@ MSG_REJ_SFCAP_1       "file %s: software capability (CA_SUNW_SF_1) \
                         unsupported: %s"
@ MSG_REJ_MACHCAP       "file %s: machine capability (CA_SUNW_MACH) \
                         unsupported: %s"
@ MSG_REJ_PLATCAP       "file %s: platform capability (CA_SUNW_PLAT) \
                         unsupported: %s"
@ MSG_REJ_HWCAP_2       "file %s: hardware capability (CA_SUNW_HW_2) \
                         unsupported: %s"
@ MSG_REJ_ARCHIVE       "file %s: invalid archive use"
@ MSG_REJ_KMOD          "file %s: kernel modules can't be link-edit input"
#endif /* ! codereview */

# Guidance messages
@ MSG_GUIDE_SUMMARY     "see ld(1) -z guidance for more information"
@ MSG_GUIDE_DEFS        "-z defs option recommended for shared objects"
@ MSG_GUIDE_DIRECT      "-B direct or -z direct option recommended before \
                         first dependency"
@ MSG_GUIDE_LAZYLOAD    "-z lazyload option recommended before \
                         first dependency"
@ MSG_GUIDE_MAPFILE     "version 2 mapfile syntax recommended: %s"
@ MSG_GUIDE_TEXT        "position independent (PIC) code recommended for \
                         shared objects"
@ MSG_GUIDE_UNUSED      "removal of unused dependency recommended: %s"
@ MSG_GUIDE_KMOD        "use -z type=kmod, not -r -dy"
#endif /* ! codereview */

@ _END_
```



```


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

@ MSG_STR_EOF           "<eof>"
@ MSG_STR_ERROR         "<error>"
@ MSG_STR_EMPTY         ""
@ MSG_QSTR_BANG         "'!'"
@ MSG_STR_COLON         ":"
@ MSG_QSTR_COLON        "':'"
@ MSG_QSTR_SEMICOLON    "';'"
@ MSG_QSTR_EQUAL        "'='"
@ MSG_QSTR_PLUSEQ       "'+='"
@ MSG_QSTR_MINUSEQ      "'-='"
@ MSG_QSTR_ATSIGN       "'@'"
@ MSG_QSTR_DASH         "'-'"
@ MSG_QSTR_LEFTBKT      "'{'"
@ MSG_QSTR_RIGHTBKT     "'}'"
@ MSG_QSTR_PIPE         "'|'"
@ MSG_QSTR_STAR         "'*'"
@ MSG_STR_DOT           "."
@ MSG_STR_SLASH         "/"
@ MSG_STR_COMMA         ","
@ MSG_STR_DYNAMIC       "(.dynamic)"
@ MSG_STR_ORIGIN        "$ORIGIN"
@ MSG_STR_MACHINE       "$MACHINE"
@ MSG_STR_PLATFORM      "$PLATFORM"
@ MSG_STR_ISALIST       "$ISALIST"
@ MSG_STR_OSNAME        "$OSNAME"
@ MSG_STR_OSREL         "$OSREL"
@ MSG_STR_UU_REAL_U     "__real_"
@ MSG_STR_UU_WRAP_U     "__wrap_"
@ MSG_STR_UELF32        "_ELF32"
@ MSG_STR_UELF64        "_ELF64"
@ MSG_STR_USPARC        "_sparc"
@ MSG_STR_UX86          "_x86"
@ MSG_STR_TRUE          "true"

@ MSG_STR_CDIR_ADD      "$add"
@ MSG_STR_CDIR_CLEAR    "$clear"
@ MSG_STR_CDIR_ERROR    "$error"
@ MSG_STR_CDIR_MFVER    "$mapfile_version"
@ MSG_STR_CDIR_IF       "$if"
@ MSG_STR_CDIR_ELIF     "$elif"
@ MSG_STR_CDIR_ELSE     "$else"
@ MSG_STR_CDIR_ENDIF    "$endif"

@ MSG_STR_GROUP         "GROUP"
@ MSG_STR_SUNW_COMDAT   "SUNW_COMDAT"

@ MSG_FMT_ARMEM         "%s(%s)"
@ MSG_FMT_COLPATH       "%s:%s"
@ MSG_FMT_SYMNAM        "'%s'"
@ MSG_FMT_NULLSYMNAM    "%s[%d]"
@ MSG_FMT_STRCAT        "%s%s"

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

@ MSG_SUNW_OST_SGS      "SUNW_OST_SGS"


# Section strings

@ MSG_SCN_BSS           ".bss"
@ MSG_SCN_DATA          ".data"
```

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

838 @ MSG_SYM_FINIARRAY     "finiarray"
839 @ MSG_SYM_INITARRAY     "initarray"
840 @ MSG_SYM_PREINITARRAY  "preinitarray"

842 #
843 # GNU section names
844 #
845 @ MSG_SCN_CTORS         ".ctors"
846 @ MSG_SCN_DTORS         ".dtors"
847 @ MSG_SCN_EHFRAME       ".eh_frame"
848 @ MSG_SCN_EHFRAME_HDR   ".eh_frame_hdr"
849 @ MSG_SCN_GCC_X_TBL     ".gcc_except_table"
```



```
850 @ MSG_SCN_JCR           ".jcr"

852 # Segment names for segments referenced by entrance criteria

854 @ MSG_ENT_BSS           "bss"
855 @ MSG_ENT_DATA          "data"
856 @ MSG_ENT_EXTRA         "extra"
857 @ MSG_ENT_LDATA         "ldata"
858 @ MSG_ENT_LRODATA       "lrodata"
859 @ MSG_ENT_NOTE          "note"
860 @ MSG_ENT_TEXT          "text"

862 # Symbol names

864 @ MSG_SYM_START         "_start"
865 @ MSG_SYM_MAIN          "main"

867 @ MSG_SYM_FINI_U        "_fini"
868 @ MSG_SYM_INIT_U        "_init"
869 @ MSG_SYM_DYNAMIC       "DYNAMIC"
870 @ MSG_SYM_DYNAMIC_U     "_DYNAMIC"
871 @ MSG_SYM_EDATA         "edata"
872 @ MSG_SYM_EDATA_U       "_edata"
873 @ MSG_SYM_END           "end"
874 @ MSG_SYM_END_U         "_end"
875 @ MSG_SYM_ETEXT         "etext"
876 @ MSG_SYM_ETEXT_U       "_etext"
877 @ MSG_SYM_GOFTBL        "GLOBAL_OFFSET_TABLE_"
878 @ MSG_SYM_GOFTBL_U      "_GLOBAL_OFFSET_TABLE_"
879 @ MSG_SYM_PLKTBL        "PROCEDURE_LINKAGE_TABLE_"
880 @ MSG_SYM_PLKTBL_U      "_PROCEDURE_LINKAGE_TABLE_"
881 @ MSG_SYM_TLSGETADDR_U  "__tls_get_addr"
882 @ MSG_SYM_TLSGETADDR_UU "___tls_get_addr"

884 @ MSG_SYM_L_END         "END_"
885 @ MSG_SYM_L_END_U       "_END_"
886 @ MSG_SYM_L_START       "START_"
887 @ MSG_SYM_L_START_U     "_START_"

889 @ MSG_SYM_SECBOUND_START "__start_"
890 @ MSG_SYM_SECBOUND_STOP "__stop_"

892 #endif /* ! codereview */
893 # Support functions

895 @ MSG_SUP_VERSION       "ld_version"
896 @ MSG_SUP_INPUT_DONE    "ld_input_done"

898 @ MSG_SUP_START_64      "ld_start64"
899 @ MSG_SUP_ATEXIT_64     "ld_atexit64"
900 @ MSG_SUP_OPEN_64       "ld_open64"
901 @ MSG_SUP_FILE_64       "ld_file64"
902 @ MSG_SUP_INSEC_64      "ld_input_section64"
903 @ MSG_SUP_SEC_64        "ld_section64"

905 @ MSG_SUP_START         "ld_start"
906 @ MSG_SUP_ATEXIT        "ld_atexit"
907 @ MSG_SUP_OPEN          "ld_open"
908 @ MSG_SUP_FILE          "ld_file"
909 @ MSG_SUP_INSEC         "ld_input_section"
910 @ MSG_SUP_SEC           "ld_section"

912 #
913 # Message previously in ’ld’
914 #
915 #
```

```
916 @ _START_

918 # System error messages

920 @ MSG_SYS_STAT          "file %s: stat failed: %s"
921 @ MSG_SYS_READ          "file %s: read failed: %s"
922 @ MSG_SYS_NOTREG        "file %s: is not a regular file"

924 # Argument processing messages

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


955 @ MSG_ARG_FLAGS         "flags processing errors"
956 @ MSG_ARG_FILES         "file processing errors. No output written to %s"
957 @ MSG_ARG_SYM_WARN      "symbol referencing errors"
958 @ MSG_ARG_SYM_FATAL     "symbol referencing errors. No output written to %s"
959 @ MSG_ARG_AR_GRP_OLAP   "%s cannot be nested"
960 @ MSG_ARG_AR_GRP_BAD    "%s used without corresponding %s"


963 # Messages used to refer to options where there is more than
964 # one name accepted.

966 @ MSG_MARG_AR_GRPS      "archive rescan groups \
967                          (-z rescan-start, -(, --start-group)"
968 @ MSG_MARG_AR_GRP_END   "archive rescan group end option \
969                          (-z rescan-end, -), --end-group)"
970 @ MSG_MARG_AR_GRP_START "archive rescan group start option \
971                          (-z rescan-start, -(, --start-group)"
972 @ MSG_MARG_ENTRY        "entry point option (-e, --entry)"
973 @ MSG_MARG_FILTER_AUX   "auxiliary filter option (-f, --auxiliary)"
974 @ MSG_MARG_FILTER       "filter option (-F, --filter)"
975 @ MSG_MARG_OUTFILE      "output object option (-o, --output)"
976 @ MSG_MARG_REL          "relocatable object option (-r, --relocatable, \
977                          -z type=reloc)"
254 @ MSG_MARG_REL          "relocatable object option (-r, --relocatable)"
978 @ MSG_MARG_RPATH        "runpath option (-R, -rpath)"
979 @ MSG_MARG_SO           "shared object option (-G, -shared, -z type=shared)"
256 @ MSG_MARG_SO           "shared object option (-G, -shared)"
```



```
980 @ MSG_MARG_SONAME       "soname option (-h, --soname)"
981 @ MSG_MARG_STRIP        "strip option (-s, --strip-all)"
982 @ MSG_MARG_TYPE_KMOD    "-z type=kmod"
983 #endif /* ! codereview */

985 # Entrance criteria messages

987 @ MSG_ENT_MAP_FMT_TIL_1 "\t\t%s\n\n"
988 @ MSG_ENT_MAP_TITLE_1   "LINK EDITOR MEMORY MAP"

990 #
991 # TRANSLATION_NOTE -- Entry map header
992 #
993 # The next message is a format string for a title. The title is composed of
994 # two lines. In C locale, it would look like:
995 #
996 #       output          input           new
997 #       section         section         displacement    size
998 #
999 # The \t characters are used for alignment.  (output section), (input section),
1000 # and (new displacement) have to be aligned.
1001 #
1002 @ 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"
1003 @ 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"
1004 @ MSG_ENT_ITM_OUTPUT    "output"
1005 @ MSG_ENT_ITM_INPUT     "input"
1006 @ MSG_ENT_ITM_NEW       "new"
1007 @ MSG_ENT_ITM_SECTION   "section"
1008 @ MSG_ENT_ITM_DISPMNT   "displacement"
1009 @ MSG_ENT_ITM_SIZE      "size"
1010 @ MSG_ENT_ITM_VIRTUAL   "virtual"
1011 @ MSG_ENT_ITM_ADDRESS   "address"

1013 @ MSG_ENT_MAP_ENTRY_1   "%-8.8s\t\t\t%08.2llx\t%08.2llx\n"
1014 @ MSG_ENT_MAP_ENTRY_2   "\t\t%-8.8s\t%08.2llx\t%08.2llx %s\n"

1016 #
1017 # TRANSLATION_NOTE -- multiple defined symbol table header
1018 #
1019 # In C locale, an example output is:
1020 #
1021 #               MULTIPLY DEFINED SYMBOLS
1022 #
1023 #
1024 #symbol                          definition used     also defined in
1025 #
1026 #variable1                          main.o
1027 #                                                      ./libfred.so
1028 @ MSG_ENT_MUL_FMT_TIL_0 "\n\n\t\t%s\n\n\n"
1029 @ MSG_ENT_MUL_TIL_0     "MULTIPLY DEFINED SYMBOLS"

1031 #
1032 # TRANSLATION_NOTE -- This is the format string for:
1033 #
1034 #symbol                          definition used     also defined in
1035 #
1036 @ MSG_ENT_MUL_FMT_TIL_1 "%s\t\t\t\t  %s     %s\n\n"
1037 @ MSG_ENT_MUL_ITM_SYM   "symbol"
1038 @ MSG_ENT_MUL_ITM_DEF_0 "definition used"
1039 @ MSG_ENT_MUL_ITM_DEF_1 "also defined in"

1041 #
1042 # TRANSLATION_NOTE -- This is the format string for the second item:
1043 #
1044 @ MSG_ENT_MUL_ENTRY_1   "%-35s %s\n"
```

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

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

# Library messages

@ MSG_LIB_NOTFOUND      "library -l%s: not found"
@ 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.
```

```
1311 @ MSG_DBG_FOPEN_MODE    "w"

1313 @ MSG_DBG_CLS32_FMT     "32: "
1314 @ MSG_DBG_CLS64_FMT     "64: "

1316 @ MSG_STR_PATHTOK       ";:"
1317 @ MSG_STR_AOUT          "a.out"

1319 @ MSG_STR_LIB_A         "%s/lib%s.a"
1320 @ MSG_STR_LIB_SO        "%s/lib%s.so"
1321 @ MSG_STR_PATH          "%s/%s"
1322 @ MSG_STR_STRNL         "%s\n"
1323 @ MSG_STR_NL            "\n"
1324 @ MSG_STR_CAPGROUPID    "CAP_GROUP_%d"

1326 @ MSG_STR_LD_DYNAMIC    "dynamic"
1327 @ MSG_STR_SYMBOLIC      "symbolic"
1328 @ MSG_STR_ELIMINATE     "eliminate"
1329 @ MSG_STR_LOCAL         "local"
1330 @ MSG_STR_PROGBITS      "progbits"
1331 @ MSG_STR_SYMTAB        "symtab"
1332 @ MSG_STR_DYNSYM        "dynsym"
1333 @ MSG_STR_REL           "rel"
1334 @ MSG_STR_RELA          "rela"
1335 @ MSG_STR_STRTAB        "strtab"
1336 @ MSG_STR_HASH          "hash"
1337 @ MSG_STR_LIB           "lib"
1338 @ MSG_STR_NOTE          "note"
1339 @ MSG_STR_NOBITS        "nobits"
1340 @ MSG_STR_HWCAP_1       "hwcap_1"
1341 @ MSG_STR_SFCAP_1       "sfcap_1"
1342 @ MSG_STR_SOEXT         ".so"

1344 @ 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:\
1345                          S:VW:Y:?"

1347 # Argument processing strings

1349 @ MSG_ARG_3             "-3"
1350 @ MSG_ARG_6             "-6"
1351 @ MSG_ARG_A             "-a"
1352 @ MSG_ARG_B             "-b"
1353 @ MSG_ARG_CB            "-B"
1354 @ MSG_ARG_BDIRECT       "-Bdirect"
1355 @ MSG_ARG_BDYNAMIC      "-Bdynamic"
1356 @ MSG_ARG_BELIMINATE    "-Beliminate"
1357 @ MSG_ARG_BGROUP        "-Bgroup"
1358 @ MSG_ARG_BLOCAL        "-Blocal"
1359 @ MSG_ARG_BNODIRECT     "-Bnodirect"
1360 @ MSG_ARG_BSYMBOLIC     "-Bsymbolic"
1361 @ MSG_ARG_BTRANSLATOR   "-Btranslator"
1362 @ MSG_ARG_C             "-c"
1363 @ MSG_ARG_D             "-d"
1364 @ MSG_ARG_DY            "-dy"
1365 @ MSG_ARG_CI            "-I"
1366 @ MSG_ARG_CN            "-N"
1367 @ MSG_ARG_P             "-p"
1368 @ MSG_ARG_CP            "-P"
1369 @ MSG_ARG_CQ            "-Q"
1370 @ MSG_ARG_CY            "-Y"
1371 @ MSG_ARG_CYL           "-YL"
1372 @ MSG_ARG_CYP           "-YP"
1373 @ MSG_ARG_CYU           "-YU"
1374 @ MSG_ARG_Z             "-z"
1375 @ MSG_ARG_ZDEFNODEF     "-z[defs|nodefs]"
```



```
1376 @ MSG_ARG_ZASLR         "-zaslr"
1377 @ MSG_ARG_ZGUIDE        "-zguidance"
1378 @ MSG_ARG_ZNODEF        "-znodefs"
1379 @ MSG_ARG_ZNOINTERP     "-znointerp"
1380 @ MSG_ARG_ZRELAXRELOC   "-zrelaxreloc"
1381 @ MSG_ARG_ZNORELAXRELOC "-znorelaxreloc"
1382 @ MSG_ARG_ZTEXT         "-ztext"
1383 @ MSG_ARG_ZTEXTOFF      "-ztextoff"
1384 @ MSG_ARG_ZTEXTWARN     "-ztextwarn"
1385 @ MSG_ARG_ZTEXTALL      "-z[text|textwarn|textoff]"
1386 @ MSG_ARG_ZLOADFLTR     "-zloadfltr"
1387 @ MSG_ARG_ZCOMBRELOC    "-zcombreloc"
1388 @ MSG_ARG_ZSYMBOLCAP    "-zsymbolcap"
1389 @ MSG_ARG_ZFATWNOFATW   "-z[fatal-warnings|nofatalwarnings]"

1391 @ MSG_ARG_ABSEXEC       "absexec"
1392 @ MSG_ARG_ALTEXEC64     "altexec64"
1393 @ MSG_ARG_ASLR          "aslr"
1394 @ MSG_ARG_NOCOMPSTRTAB  "nocompstrtab"
1395 @ MSG_ARG_GROUPPERM     "groupperm"
1396 @ MSG_ARG_NOGROUPPERM   "nogroupperm"
1397 @ MSG_ARG_LAZYLOAD      "lazyload"
1398 @ MSG_ARG_NOLAZYLOAD    "nolazyload"
1399 @ MSG_ARG_INTERPOSE     "interpose"
1400 @ MSG_ARG_DIRECT        "direct"
1401 @ MSG_ARG_NODIRECT      "nodirect"
1402 @ MSG_ARG_IGNORE        "ignore"
1403 @ MSG_ARG_RECORD        "record"
1404 @ MSG_ARG_INITFIRST     "initfirst"
1405 @ MSG_ARG_INITARRAY     "initarray="
1406 @ MSG_ARG_FINIARRAY     "finiarray="
1407 @ MSG_ARG_PREINITARRAY  "preinitarray="
1408 @ MSG_ARG_RTLDINFO      "rtldinfo="
1409 @ MSG_ARG_DTRACE        "dtrace="
1410 @ MSG_ARG_TRANSLATOR    "translator"
1411 @ MSG_ARG_NOOPEN        "nodlopen"
1412 @ MSG_ARG_NOW           "now"
1413 @ MSG_ARG_ORIGIN        "origin"
1414 @ MSG_ARG_DEFS          "defs"
1415 @ MSG_ARG_NODEFS        "nodefs"
1416 @ MSG_ARG_NODUMP        "nodump"
1417 @ MSG_ARG_NOVERSION     "noversion"
1418 @ MSG_ARG_TEXT          "text"
1419 @ MSG_ARG_TEXTOFF       "textoff"
1420 @ MSG_ARG_TEXTWARN      "textwarn"
1421 @ MSG_ARG_MULDEFS       "muldefs"
1422 @ MSG_ARG_NODELETE      "nodelete"
1423 @ MSG_ARG_NOINTERP      "nointerp"
1424 @ MSG_ARG_NOPARTIAL     "nopartial"
1425 @ MSG_ARG_NORELOC       "noreloc"
1426 @ MSG_ARG_REDLOCSYM     "redlocsym"
1427 @ MSG_ARG_VERBOSE       "verbose"
1428 @ MSG_ARG_WEAKEXT       "weakextract"
1429 @ MSG_ARG_LOADFLTR      "loadfltr"
1430 @ MSG_ARG_ALLEXTRT      "allextract"
1431 @ MSG_ARG_DFLEXTRT      "defaultextract"
1432 @ MSG_ARG_COMBRELOC     "combreloc"
1433 @ MSG_ARG_NOCOMBRELOC   "nocombreloc"
1434 @ MSG_ARG_NODEFAULTLIB  "nodefaultlib"
1435 @ MSG_ARG_ENDFILTEE     "endfiltee"
1436 @ MSG_ARG_LD32          "ld32="
1437 @ MSG_ARG_LD64          "ld64="
1438 @ MSG_ARG_RESCAN        "rescan"
1439 @ MSG_ARG_RESCAN_NOW    "rescan-now"
1440 @ MSG_ARG_RESCAN_START  "rescan-start"
1441 @ MSG_ARG_RESCAN_END    "rescan-end"
```

```
@ MSG_ARG_GUIDE          "guidance"
@ MSG_ARG_NOLDYNSYM      "noldynsym"
@ MSG_ARG_RELAXRELOC     "relaxreloc"
@ MSG_ARG_NORELAXRELOC   "norelaxreloc"
@ MSG_ARG_NOSIGHANDLER   "nosighandler"
@ MSG_ARG_GLOBAUDIT      "globalaudit"
@ MSG_ARG_TARGET         "target="
@ MSG_ARG_WRAP           "wrap="
@ MSG_ARG_FATWARN        "fatal-warnings"
@ MSG_ARG_NOFATWARN      "nofatal-warnings"
@ MSG_ARG_HELP           "help"
@ MSG_ARG_GROUP          "group"
@ MSG_ARG_REDUCE         "reduce"
@ MSG_ARG_STATIC         "static"
@ MSG_ARG_SYMBOLCAP      "symbolcap"
@ MSG_ARG_DEFERRED       "deferred"
@ MSG_ARG_NODEFERRED     "nodeferred"
@ MSG_ARG_ASSDEFLIB      "assert-deflib"
@ MSG_ARG_TYPE           "type"
#endif /* ! codereview */

@ MSG_ARG_LCOM           "L,"
@ MSG_ARG_PCOM           "P,"
@ MSG_ARG_UCOM           "U,"

@ MSG_ARG_T_RPATH        "rpath"
@ MSG_ARG_T_SHARED       "shared"
@ MSG_ARG_T_SONAME       "soname"
@ MSG_ARG_T_WL           "l,-"

@ MSG_ARG_T_AUXFLTR      "-auxiliary"
@ MSG_ARG_T_MULDEFS      "-allow-multiple-definition"
@ MSG_ARG_T_INTERP       "-dynamic-linker"
@ MSG_ARG_T_ENDGROUP     "-end-group"
@ MSG_ARG_T_ENTRY        "-entry"
@ MSG_ARG_T_STDFLTR      "-filter"
@ MSG_ARG_T_FATWARN      "-fatal-warnings"
@ MSG_ARG_T_NOFATWARN    "-no-fatal-warnings"
@ MSG_ARG_T_HELP         "-help"
@ MSG_ARG_T_LIBRARY      "-library"
@ MSG_ARG_T_LIBPATH      "-library-path"
@ MSG_ARG_T_NOUNDEF      "-no-undefined"
@ MSG_ARG_T_NOWHOLEARC   "-no-whole-archive"
@ MSG_ARG_T_OUTPUT       "-output"
@ MSG_ARG_T_RELOCATABLE  "-relocatable"
@ MSG_ARG_T_STARTGROUP   "-start-group"
@ MSG_ARG_T_STRIP        "-strip-all"
@ MSG_ARG_T_UNDEF        "-undefined"
@ MSG_ARG_T_VERSION      "-version"
@ MSG_ARG_T_WHOLEARC     "-whole-archive"
@ MSG_ARG_T_WRAP         "-wrap"
@ MSG_ARG_T_OPAR         "("
@ MSG_ARG_T_CPAR         ")"

@ MSG_ARG_ENABLED        "enabled"
@ MSG_ARG_DISABLED       "disabled"
@ MSG_ARG_ENABLE         "enable"
@ MSG_ARG_DISABLE        "disable"

# -z guidance=item strings
@ MSG_ARG_GUIDE_DELIM            ",: \t"
@ MSG_ARG_GUIDE_NO_ALL           "noall"
@ MSG_ARG_GUIDE_NO_DEFS          "nodefs"
@ MSG_ARG_GUIDE_NO_DIRECT        "nodirect"
@ MSG_ARG_GUIDE_NO_LAZYLOAD      "nolazyload"
@ MSG_ARG_GUIDE_NO_MAPFILE       "nomapfile"
```



```
@ MSG_ARG_GUIDE_NO_TEXT          "notext"
@ MSG_ARG_GUIDE_NO_UNUSED        "nounused"

# -z type= strings
@ MSG_ARG_TYPE_RELOC     "reloc"
@ MSG_ARG_TYPE_EXEC      "exec"
@ MSG_ARG_TYPE_SHARED    "shared"
@ MSG_ARG_TYPE_KMOD      "kmod"
#endif /* ! codereview */

# Environment variable strings

@ MSG_LD_RUN_PATH        "LD_RUN_PATH"
@ MSG_LD_LIBPATH_32      "LD_LIBRARY_PATH_32"
@ MSG_LD_LIBPATH_64      "LD_LIBRARY_PATH_64"
@ MSG_LD_LIBPATH         "LD_LIBRARY_PATH"

@ MSG_LD_NOVERSION_32    "LD_NOVERSION_32"
@ MSG_LD_NOVERSION_64    "LD_NOVERSION_64"
@ MSG_LD_NOVERSION       "LD_NOVERSION"

@ MSG_SGS_SUPPORT_32     "SGS_SUPPORT_32"
@ MSG_SGS_SUPPORT_64     "SGS_SUPPORT_64"
@ MSG_SGS_SUPPORT        "SGS_SUPPORT"


# Symbol names

@ MSG_SYM_LIBVER_U       "_lib_version"


# Mapfile tokens

@ MSG_MAP_LOAD           "load"
@ MSG_MAP_NOTE           "note"
@ MSG_MAP_NULL           "null"
@ MSG_MAP_STACK          "stack"
@ MSG_MAP_ADDVERS        "addvers"
@ MSG_MAP_FUNCTION       "function"
@ MSG_MAP_DATA           "data"
@ MSG_MAP_COMMON         "common"
@ MSG_MAP_PARENT         "parent"
@ MSG_MAP_EXTERN         "extern"
@ MSG_MAP_DIRECT         "direct"
@ MSG_MAP_NODIRECT       "nodirect"
@ MSG_MAP_FILTER         "filter"
@ MSG_MAP_AUXILIARY      "auxiliary"
@ MSG_MAP_OVERRIDE       "override"
@ MSG_MAP_INTERPOSE      "interpose"
@ MSG_MAP_DYNSORT        "dynsort"
@ MSG_MAP_NODYNSORT      "nodynsort"

@ MSG_MAPKW_ALIGN                "ALIGN"
@ MSG_MAPKW_ALLOC                "ALLOC"
@ MSG_MAPKW_ALLOW                "ALLOW"
@ MSG_MAPKW_AMD64_LARGE          "AMD64_LARGE"
@ MSG_MAPKW_ASSIGN_SECTION       "ASSIGN_SECTION"
@ MSG_MAPKW_AUX                  "AUXILIARY"
@ MSG_MAPKW_CAPABILITY           "CAPABILITY"
@ MSG_MAPKW_COMMON               "COMMON"
@ MSG_MAPKW_DATA                 "DATA"
@ MSG_MAPKW_DEFAULT              "DEFAULT"
@ MSG_MAPKW_DEPEND_VERSIONS      "DEPEND_VERSIONS"
@ MSG_MAPKW_DIRECT               "DIRECT"
@ MSG_MAPKW_DISABLE              "DISABLE"
@ MSG_MAPKW_DYNSORT              "DYNSORT"
```

```
1574 @ MSG_MAPKW_ELIMINATE           "ELIMINATE"
1575 @ MSG_MAPKW_EXECUTE             "EXECUTE"
1576 @ MSG_MAPKW_EXPORTED            "EXPORTED"
1577 @ MSG_MAPKW_EXTERN              "EXTERN"
1578 @ MSG_MAPKW_FILTER              "FILTER"
1579 @ MSG_MAPKW_FILE_BASENAME       "FILE_BASENAME"
1580 @ MSG_MAPKW_FILE_PATH           "FILE_PATH"
1581 @ MSG_MAPKW_FILE_OBJNAME        "FILE_OBJNAME"
1582 @ MSG_MAPKW_FUNCTION            "FUNCTION"
1583 @ MSG_MAPKW_FLAGS               "FLAGS"
1584 @ MSG_MAPKW_GLOBAL              "GLOBAL"
1585 @ MSG_MAPKW_INTERPOSE           "INTERPOSE"
1586 @ MSG_MAPKW_HIDDEN              "HIDDEN"
1587 @ MSG_MAPKW_HDR_NOALLOC         "HDR_NOALLOC"
1588 @ MSG_MAPKW_HW                  "HW"
1589 @ MSG_MAPKW_HW_1                "HW_1"
1590 @ MSG_MAPKW_HW_2                "HW_2"
1591 @ MSG_MAPKW_IS_NAME             "IS_NAME"
1592 @ MSG_MAPKW_IS_ORDER            "IS_ORDER"
1593 @ MSG_MAPKW_LOAD_SEGMENT        "LOAD_SEGMENT"
1594 @ MSG_MAPKW_LOCAL               "LOCAL"
1595 @ MSG_MAPKW_MACHINE             "MACHINE"
1596 @ MSG_MAPKW_MAX_SIZE            "MAX_SIZE"
1597 @ MSG_MAPKW_NOHDR               "NOHDR"
1598 @ MSG_MAPKW_NODIRECT            "NODIRECT"
1599 @ MSG_MAPKW_NODYNSORT           "NODYNSORT"
1600 @ MSG_MAPKW_NOTE_SEGMENT        "NOTE_SEGMENT"
1601 @ MSG_MAPKW_NULL_SEGMENT        "NULL_SEGMENT"
1602 @ MSG_MAPKW_OS_ORDER            "OS_ORDER"
1603 @ MSG_MAPKW_PADDR               "PADDR"
1604 @ MSG_MAPKW_PARENT              "PARENT"
1605 @ MSG_MAPKW_PHDR_ADD_NULL       "PHDR_ADD_NULL"
1606 @ MSG_MAPKW_PLATFORM            "PLATFORM"
1607 @ MSG_MAPKW_PROTECTED           "PROTECTED"
1608 @ MSG_MAPKW_READ                "READ"
1609 @ MSG_MAPKW_ROUND               "ROUND"
1610 @ MSG_MAPKW_REQUIRE             "REQUIRE"
1611 @ MSG_MAPKW_SEGMENT_ORDER       "SEGMENT_ORDER"
1612 @ MSG_MAPKW_SF                  "SF"
1613 @ MSG_MAPKW_SF_1                "SF_1"
1614 @ MSG_MAPKW_SINGLETON           "SINGLETON"
1615 @ MSG_MAPKW_SIZE                "SIZE"
1616 @ MSG_MAPKW_SIZE_SYMBOL         "SIZE_SYMBOL"
1617 @ MSG_MAPKW_STACK               "STACK"
1618 @ MSG_MAPKW_SYMBOL_SCOPE        "SYMBOL_SCOPE"
1619 @ MSG_MAPKW_SYMBOL_VERSION      "SYMBOL_VERSION"
1620 @ MSG_MAPKW_SYMBOLIC            "SYMBOLIC"
1621 @ MSG_MAPKW_TYPE                "TYPE"
1622 @ MSG_MAPKW_VADDR               "VADDR"
1623 @ MSG_MAPKW_VALUE               "VALUE"
1624 @ MSG_MAPKW_WRITE               "WRITE"


1627 @ MSG_STR_DTRACE        "PT_SUNWDTRACE"
```

```
**********************************************************
   94536 Sun Feb 24 19:19:12 2019
new/usr/src/cmd/sgs/libld/common/relocate.c
ld: implement -ztype and rework option parsing
**********************************************************
______unchanged_portion_omitted_
```

```
1473 uintptr_t
1474 ld_process_sym_reloc(Ofl_desc *ofl, Rel_desc *reld, Rel *reloc, Is_desc *isp,
1475     const char *isname, Word isscnndx)
1476 {
1477         Word            rtype = reld->rel_rtype;
1478         ofl_flag_t      flags = ofl->ofl_flags;
1479         Sym_desc        *sdp = reld->rel_sym;
1480         Sym_aux         *sap;
1481         Boolean         local;
1482         Conv_inv_buf_t  inv_buf;

1484         DBG_CALL(Dbg_reloc_in(ofl->ofl_lml, ELF_DBG_LD, ld_targ.t_m.m_mach,
1485             ld_targ.t_m.m_rel_sht_type, (void *)reloc, isname, isscnndx,
1486             ld_reloc_sym_name(reld)));

1488         /*
1489          * Indicate this symbol is being used for relocation and therefore must
1490          * have its output address updated accordingly (refer to update_osym()).
1491          */
1492         sdp->sd_flags |= FLG_SY_UPREQD;

1494         /*
1495          * Indicate the section this symbol is defined in has been referenced,
1496          * therefor it *is not* a candidate for elimination.
1497          */
1498         if (sdp->sd_isc) {
1499                 sdp->sd_isc->is_flags |= FLG_IS_SECTREF;
1500                 sdp->sd_isc->is_file->ifl_flags |= FLG_IF_FILEREF;
1501         }

1503         if (!ld_reloc_set_aux_usym(ofl, reld, sdp))
1504                 return (S_ERROR);

1506         /*
1507          * Determine if this symbol is actually an alias to another symbol.  If
1508          * so, and the alias is not REF_DYN_SEEN, set ra_usym to point to the
1509          * weak symbols strong counter-part.  The one exception is if the
1510          * FLG_SY_MVTOCOMM flag is set on the weak symbol.  If this is the case,
1511          * the strong is only here because of its promotion, and the weak symbol
1512          * should still be used for the relocation reference (see reloc_exec()).
1513          */
1514         sap = sdp->sd_aux;
1515         if (sap && sap->sa_linkndx &&
1516             ((ELF_ST_BIND(sdp->sd_sym->st_info) == STB_WEAK) ||
1517             (sdp->sd_flags & FLG_SY_WEAKDEF)) &&
1518             (!(sdp->sd_flags & FLG_SY_MVTOCOMM))) {
1519                 Sym_desc        *_sdp;

1521                 _sdp = sdp->sd_file->ifl_oldndx[sap->sa_linkndx];
1522                 if ((_sdp->sd_ref != REF_DYN_SEEN) &&
1523                     !ld_reloc_set_aux_usym(ofl, reld, _sdp))
1524                         return (S_ERROR);
1525         }

1527         /*
1528          * Determine whether this symbol should be bound locally or not.
1529          * Symbols are bound locally if one of the following is true:
1530          *
1531          *  -   the symbol is of type STB_LOCAL.
```



```
1532          *
1533          *  -   the output image is not a relocatable object and the relocation
1534          *      is relative to the .got.
1535          *
1536          *  -   the section being relocated is of type SHT_SUNW_dof.  These
1537          *      sections must be bound to the functions in the containing
1538          *      object and can not be interposed upon.
1539          *
1540          *  -   the symbol has been reduced (scoped to a local or symbolic) and
1541          *      reductions are being processed.
1542          *
1543          *  -   the -Bsymbolic flag is in use when building a shared object,
1544          *      and the symbol hasn't explicitly been defined as nodirect.
1545          *
1546          *  -   an executable (fixed address) )is being created, and the symbol
1546          *  -   an executable (fixed address) is being created, and the symbol
1547          *      is defined in the executable.
1548          *
1549          *  -   the relocation is against a segment which will not be loaded
1550          *      into memory.  In this case, the relocation must be resolved
1551          *      now, as ld.so.1 can not process relocations against unmapped
1552          *      segments.
1553          */
1554         local = FALSE;
1555         if (ELF_ST_BIND(sdp->sd_sym->st_info) == STB_LOCAL) {
1556                 local = TRUE;
1557         } else if (!(reld->rel_flags & FLG_REL_LOAD)) {
1558                 local = TRUE;
1559         } else if (sdp->sd_sym->st_shndx != SHN_UNDEF) {
1560                 if (reld->rel_isdesc &&
1561                     reld->rel_isdesc->is_shdr->sh_type == SHT_SUNW_dof) {
1562                         local = TRUE;
1563                 } else if (!(flags & FLG_OF_RELOBJ) &&
1564                     (IS_LOCALBND(rtype) || IS_SEG_RELATIVE(rtype))) {
1565                         local = TRUE;
1566                 } else if ((sdp->sd_ref == REF_REL_NEED) &&
1567                     ((sdp->sd_flags & FLG_SY_CAP) == 0)) {
1568                         /*
1569                          * Global symbols may have been individually reduced in
1570                          * scope.  If the whole object is to be self contained,
1571                          * such as when generating an executable or a symbolic
1572                          * shared object, make sure all relocation symbol
1573                          * references (sections too) are treated locally.  Note,
1574                          * explicit no-direct symbols should not be bound to
1575                          * locally.
1576                          */
1577                         if ((sdp->sd_flags &
1578                             (FLG_SY_HIDDEN | FLG_SY_PROTECT)))
1579                                 local = TRUE;
1580                         else if ((flags & FLG_OF_EXEC) ||
1581                             ((flags & FLG_OF_SYMBOLIC) &&
1582                             ((sdp->sd_flags & FLG_SY_NDIR) == 0))) {
1583                                 local = TRUE;
1584                         }
1585                 }
1586         }

1588         /*
1589          * If this is a PC_RELATIVE relocation, the relocation could be
1590          * compromised if the relocated address is later used as a copy
1591          * relocated symbol (PSARC 1999/636, bugid 4187211).  Scan the input
1592          * files symbol table to cross reference this relocation offset.
1593          */
1594         if ((ofl->ofl_flags & FLG_OF_SHAROBJ) &&
1595             IS_PC_RELATIVE(rtype) &&
1596             (IS_GOT_PC(rtype) == 0) &&
```

```
1597             (IS_PLT(rtype) == 0)) {
1598                 if (disp_inspect(ofl, reld, local) == S_ERROR)
1599                         return (S_ERROR);
1600         }

1602         /*
1603          * GOT based relocations must bind to the object being built - since
1604          * they are relevant to the current GOT.  If not building a relocatable
1605          * object - give a appropriate error message.
1606          */
1607         if (!local && !(flags & FLG_OF_RELOBJ) &&
1608             IS_GOT_BASED(rtype)) {
1609                 Ifl_desc        *ifl = reld->rel_isdesc->is_file;

1611                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADGOTBASED),
1612                     conv_reloc_type(ifl->ifl_ehdr->e_machine, rtype,
1613                     0, &inv_buf), ifl->ifl_name, demangle(sdp->sd_name));
1614                 return (S_ERROR);
1615         }

1617         /*
1618          * TLS symbols can only have TLS relocations.
1619          */
1620         if ((ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_TLS) &&
1621             (IS_TLS_INS(rtype) == 0)) {
1622                 /*
1623                  * The above test is relaxed if the target section is
1624                  * non-allocable.
1625                  */
1626                 if (RELAUX_GET_OSDESC(reld)->os_shdr->sh_flags & SHF_ALLOC) {
1627                         Ifl_desc        *ifl = reld->rel_isdesc->is_file;

1629                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLS),
1630                             conv_reloc_type(ifl->ifl_ehdr->e_machine,
1631                             rtype, 0, &inv_buf), ifl->ifl_name,
1632                             demangle(sdp->sd_name));
1633                         return (S_ERROR);
1634                 }
1635         }

1637         /*
1638          * Select the relocation to perform.
1639          */
1640         if (IS_REGISTER(rtype)) {
1641                 if (ld_targ.t_mr.mr_reloc_register == NULL) {
1642                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_NOREG));
1643                         return (S_ERROR);
1644                 }
1645                 return ((*ld_targ.t_mr.mr_reloc_register)(reld, isp, ofl));
1646         }

1648         if (flags & FLG_OF_RELOBJ)
1649                 return (reloc_relobj(local, reld, ofl));

1651         if (IS_TLS_INS(rtype))
1652                 return (reloc_TLS(local, reld, ofl));

1654         if (IS_GOT_OPINS(rtype)) {
1655                 if (ld_targ.t_mr.mr_reloc_GOTOP == NULL) {
1656                         assert(0);
1657                         return (S_ERROR);
1658                 }
1659                 return ((*ld_targ.t_mr.mr_reloc_GOTOP)(local, reld, ofl));
1660         }

1662         if (IS_GOT_RELATIVE(rtype))
```



```
1663                 return (ld_reloc_GOT_relative(local, reld, ofl));

1665         if (local)
1666                 return ((*ld_targ.t_mr.mr_reloc_local)(reld, ofl));

1668         if ((IS_PLT(rtype) || ((sdp->sd_flags & FLG_SY_CAP) &&
1669             (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_FUNC))) &&
1670             ((flags & FLG_OF_BFLAG) == 0))
1671                 return (ld_reloc_plt(reld, ofl));

1673         if ((sdp->sd_ref == REF_REL_NEED) ||
1674             (flags & FLG_OF_BFLAG) || (flags & FLG_OF_SHAROBJ) ||
1675             (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_NOTYPE))
1676                 return ((*ld_targ.t_mr.mr_add_outrel)(NULL, reld, ofl));

1678         if (sdp->sd_ref == REF_DYN_NEED)
1679                 return (reloc_exec(reld, ofl));

1681         /*
1682          * IS_NOT_REL(rtype)
1683          */
1684         return (reloc_generic(reld, ofl));
1685 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   96493 Sun Feb 24 19:19:13 2019
new/usr/src/cmd/sgs/libld/common/sections.c
ld: implement -ztype and rework option parsing
**********************************************************
______unchanged_portion_omitted_

 927 /*
 928  * Make the dynamic section.  Calculate the size of any strings referenced
 929  * within this structure, they will be added to the global string table
 930  * (.dynstr).  This routine should be called before make_dynstr().
 931  *
 932  * This routine must be maintained in parallel with update_odynamic()
 933  * in update.c
 934  */
 935 static uintptr_t
 936 make_dynamic(Ofl_desc *ofl)
 937 {
 938         Shdr            *shdr;
 939         Os_desc         *osp;
 940         Elf_Data        *data;
 941         Is_desc         *isec;
 942         size_t          cnt = 0;
 943         Aliste          idx;
 944         Ifl_desc        *ifl;
 945         Sym_desc        *sdp;
 946         size_t          size;
 947         Str_tbl         *strtbl;
 948         ofl_flag_t      flags = ofl->ofl_flags;
 949         int             not_relobj = !(flags & FLG_OF_RELOBJ);
 950         int             unused = 0;

 952         /*
 953          * Select the required string table.
 954          */
 955         if (OFL_IS_STATIC_OBJ(ofl))
 956                 strtbl = ofl->ofl_strtab;
 957         else
 958                 strtbl = ofl->ofl_dynstrtab;

 960         /*
 961          * Only a limited subset of DT_ entries apply to relocatable
 962          * objects. See the comment at the head of update_odynamic() in
 963          * update.c for details.
 964          */
 965         if (new_section(ofl, SHT_DYNAMIC, MSG_ORIG(MSG_SCN_DYNAMIC), 0,
 966             &isec, &shdr, &data) == S_ERROR)
 967                 return (S_ERROR);

 969         /*
 970          * new_section() does not set SHF_ALLOC.  If we're building anything
 971          * besides a relocatable object, then the .dynamic section should
 972          * reside in allocatable memory.
 973          */
 974         if (not_relobj)
 975                 shdr->sh_flags |= SHF_ALLOC;

 977         /*
 978          * new_section() does not set SHF_WRITE.  If we're building an object
 979          * that specifies an interpretor, then a DT_DEBUG entry is created,
 980          * which is initialized to the applications link-map list at runtime.
 981          */
 982         if (ofl->ofl_osinterp)
 983                 shdr->sh_flags |= SHF_WRITE;

 985         osp = ofl->ofl_osdynamic =
```



```
 986             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_dynamic, NULL);

 988         /*
 989          * Reserve entries for any needed dependencies.
 990          */
 991         for (APLIST_TRAVERSE(ofl->ofl_sos, idx, ifl)) {
 992                 if (!(ifl->ifl_flags & (FLG_IF_NEEDED | FLG_IF_NEEDSTR)))
 993                         continue;

 995                 /*
 996                  * If this dependency didn't satisfy any symbol references,
 997                  * generate a debugging diagnostic (ld(1) -Dunused can be used
 998                  * to display these).  If this is a standard needed dependency,
 999                  * and -z ignore is in effect, drop the dependency.  Explicitly
1000                  * defined dependencies (i.e., -N dep) don't get dropped, and
1001                  * are flagged as being required to simplify update_odynamic()
1002                  * processing.
1003                  */
1004                 if ((ifl->ifl_flags & FLG_IF_NEEDSTR) ||
1005                     ((ifl->ifl_flags & FLG_IF_DEPREQD) == 0)) {
1006                         if (unused++ == 0)
1007                                 DBG_CALL(Dbg_util_nl(ofl->ofl_lml, DBG_NL_STD));
1008                         DBG_CALL(Dbg_unused_file(ofl->ofl_lml, ifl->ifl_soname,
1009                             (ifl->ifl_flags & FLG_IF_NEEDSTR), 0));

1011                         /*
1012                          * Guidance: Remove unused dependency.
1013                          *
1014                          * If -z ignore is in effect, this warning is not
1015                          * needed because we will quietly remove the unused
1016                          * dependency.
1017                          */
1018                         if (OFL_GUIDANCE(ofl, FLG_OFG_NO_UNUSED) &&
1019                             ((ifl->ifl_flags & FLG_IF_IGNORE) == 0))
1020                                 ld_eprintf(ofl, ERR_GUIDANCE,
1021                                     MSG_INTL(MSG_GUIDE_UNUSED),
1022                                     ifl->ifl_soname);

1024                         if (ifl->ifl_flags & FLG_IF_NEEDSTR)
1025                                 ifl->ifl_flags |= FLG_IF_DEPREQD;
1026                         else if (ifl->ifl_flags & FLG_IF_IGNORE)
1027                                 continue;
1028                 }

1030                 /*
1031                  * If this object requires a DT_POSFLAG_1 entry, reserve it.
1032                  */
1033                 if ((ifl->ifl_flags & MSK_IF_POSFLAG1) && not_relobj)
1034                         cnt++;

1036                 if (st_insert(strtbl, ifl->ifl_soname) == -1)
1037                         return (S_ERROR);
1038                 cnt++;

1040                 /*
1041                  * If the needed entry contains the $ORIGIN token make sure
1042                  * the associated DT_1_FLAGS entry is created.
1043                  */
1044                 if (strstr(ifl->ifl_soname, MSG_ORIG(MSG_STR_ORIGIN))) {
1045                         ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1046                         ofl->ofl_dtflags |= DF_ORIGIN;
1047                 }
1048         }

1050         if (unused)
1051                 DBG_CALL(Dbg_util_nl(ofl->ofl_lml, DBG_NL_STD));
```

```
1053        if (not_relobj) {
1054                /*
1055                 * Reserve entries for any per-symbol auxiliary/filter strings.
1056                 */
1057                cnt += alist_nitems(ofl->ofl_dtsfltrs);

1059                /*
1060                 * Reserve entries for _init() and _fini() section addresses.
1061                 */
1062                if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_INIT_U),
1063                    SYM_NOHASH, NULL, ofl)) != NULL) &&
1064                    (sdp->sd_ref == REF_REL_NEED) &&
1065                    (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
1066                        sdp->sd_flags |= FLG_SY_UPREQD;
1067                        cnt++;
1068                }
1069                if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_FINI_U),
1070                    SYM_NOHASH, NULL, ofl)) != NULL) &&
1071                    (sdp->sd_ref == REF_REL_NEED) &&
1072                    (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
1073                        sdp->sd_flags |= FLG_SY_UPREQD;
1074                        cnt++;
1075                }

1077                /*
1078                 * Reserve entries for any soname, filter name (shared libs
1079                 * only), run-path pointers, cache names and audit requirements.
1080                 */
1081                if (ofl->ofl_soname) {
1082                        cnt++;
1083                        if (st_insert(strtbl, ofl->ofl_soname) == -1)
1084                                return (S_ERROR);
1085                }
1086                if (ofl->ofl_filtees) {
1087                        cnt++;
1088                        if (st_insert(strtbl, ofl->ofl_filtees) == -1)
1089                                return (S_ERROR);

1091                        /*
1092                         * If the filtees entry contains the $ORIGIN token
1093                         * make sure the associated DT_1_FLAGS entry is created.
1094                         */
1095                        if (strstr(ofl->ofl_filtees,
1096                            MSG_ORIG(MSG_STR_ORIGIN))) {
1097                                ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1098                                ofl->ofl_dtflags |= DF_ORIGIN;
1099                        }
1100                }
1101        }

1103        if (ofl->ofl_rpath) {
1104                cnt += 2;       /* DT_RPATH & DT_RUNPATH */
1105                if (st_insert(strtbl, ofl->ofl_rpath) == -1)
1106                        return (S_ERROR);

1108                /*
1109                 * If the rpath entry contains the $ORIGIN token make sure
1110                 * the associated DT_1_FLAGS entry is created.
1111                 */
1112                if (strstr(ofl->ofl_rpath, MSG_ORIG(MSG_STR_ORIGIN))) {
1113                        ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1114                        ofl->ofl_dtflags |= DF_ORIGIN;
1115                }
1116        }
```



```
1118        if (not_relobj) {
1119                Aliste  idx;
1120                Sg_desc *sgp;

1122                if (ofl->ofl_config) {
1123                        cnt++;
1124                        if (st_insert(strtbl, ofl->ofl_config) == -1)
1125                                return (S_ERROR);

1127                        /*
1128                         * If the config entry contains the $ORIGIN token
1129                         * make sure the associated DT_1_FLAGS entry is created.
1130                         */
1131                        if (strstr(ofl->ofl_config, MSG_ORIG(MSG_STR_ORIGIN))) {
1132                                ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1133                                ofl->ofl_dtflags |= DF_ORIGIN;
1134                        }
1135                }
1136                if (ofl->ofl_depaudit) {
1137                        cnt++;
1138                        if (st_insert(strtbl, ofl->ofl_depaudit) == -1)
1139                                return (S_ERROR);
1140                }
1141                if (ofl->ofl_audit) {
1142                        cnt++;
1143                        if (st_insert(strtbl, ofl->ofl_audit) == -1)
1144                                return (S_ERROR);
1145                }

1147                /*
1148                 * Reserve entries for the DT_HASH, DT_STRTAB, DT_STRSZ,
1149                 * DT_SYMTAB, DT_SYMENT, and DT_CHECKSUM.
1150                 */
1151                cnt += 6;

1153                /*
1154                 * If we are including local functions at the head of
1155                 * the dynsym, then also reserve entries for DT_SUNW_SYMTAB
1156                 * and DT_SUNW_SYMSZ.
1157                 */
1158                if (OFL_ALLOW_LDYNSYM(ofl))
1159                        cnt += 2;

1161                if ((ofl->ofl_dynsymsortcnt > 0) ||
1162                    (ofl->ofl_dyntlssortcnt > 0))
1163                        cnt++;          /* DT_SUNW_SORTENT */

1165                if (ofl->ofl_dynsymsortcnt > 0)
1166                        cnt += 2;       /* DT_SUNW_[SYMSORT|SYMSORTSZ] */

1168                if (ofl->ofl_dyntlssortcnt > 0)
1169                        cnt += 2;       /* DT_SUNW_[TLSSORT|TLSSORTSZ] */

1171                if ((flags & (FLG_OF_VERDEF | FLG_OF_NOVERSEC)) ==
1172                    FLG_OF_VERDEF)
1173                        cnt += 2;               /* DT_VERDEF & DT_VERDEFNUM */

1175                if ((flags & (FLG_OF_VERNEED | FLG_OF_NOVERSEC)) ==
1176                    FLG_OF_VERNEED)
1177                        cnt += 2;               /* DT_VERNEED & DT_VERNEEDNUM */

1179                if ((flags & FLG_OF_COMREL) && ofl->ofl_relocrelcnt)
1180                        cnt++;                  /* DT_RELACOUNT */

1182                if (flags & FLG_OF_TEXTREL)     /* DT_TEXTREL */
1183                        cnt++;
```

```
1185                if (ofl->ofl_osfiniarray)       /* DT_FINI_ARRAY */
1186                        cnt += 2;               /*    DT_FINI_ARRAYSZ */

1188                if (ofl->ofl_osinitarray)       /* DT_INIT_ARRAY */
1189                        cnt += 2;               /*    DT_INIT_ARRAYSZ */

1191                if (ofl->ofl_ospreinitarray)    /* DT_PREINIT_ARRAY & */
1192                        cnt += 2;               /*    DT_PREINIT_ARRAYSZ */

1194                /*
1195                 * If we have plt's reserve a DT_PLTRELSZ, DT_PLTREL and
1196                 * DT_JMPREL.
1197                 */
1198                if (ofl->ofl_pltcnt)
1199                        cnt += 3;

1201                /*
1202                 * If plt padding is needed (Sparcv9).
1203                 */
1204                if (ofl->ofl_pltpad)
1205                        cnt += 2;               /* DT_PLTPAD & DT_PLTPADSZ */

1207                /*
1208                 * If we have any relocations reserve a DT_REL, DT_RELSZ and
1209                 * DT_RELENT entry.
1210                 */
1211                if (ofl->ofl_relocsz)
1212                        cnt += 3;

1214                /*
1215                 * If a syminfo section is required create DT_SYMINFO,
1216                 * DT_SYMINSZ, and DT_SYMINENT entries.
1217                 */
1218                if (flags & FLG_OF_SYMINFO)
1219                        cnt += 3;

1221                /*
1222                 * If there are any partially initialized sections allocate
1223                 * DT_MOVETAB, DT_MOVESZ and DT_MOVEENT.
1224                 */
1225                if (ofl->ofl_osmove)
1226                        cnt += 3;

1228                /*
1229                 * Allocate one DT_REGISTER entry for every register symbol.
1230                 */
1231                cnt += ofl->ofl_regsymcnt;

1233                /*
1234                 * Reserve a entry for each '-zrtldinfo=...' specified
1235                 * on the command line.
1236                 */
1237                for (APLIST_TRAVERSE(ofl->ofl_rtldinfo, idx, sdp))
1238                        cnt++;

1240                /*
1241                 * The following entry should only be placed in a segment that
1242                 * is writable.
1243                 */
1244                if (((sgp = osp->os_sgdesc) != NULL) &&
1245                    (sgp->sg_phdr.p_flags & PF_W) && ofl->ofl_osinterp)
1246                        cnt++;          /* DT_DEBUG */

1248                /*
1249                 * Capabilities require a .dynamic entry for the .SUNW_cap
```



```
1250                 * section.
1251                 */
1252                if (ofl->ofl_oscap)
1253                        cnt++;                  /* DT_SUNW_CAP */

1255                /*
1256                 * Symbol capabilities require a .dynamic entry for the
1257                 * .SUNW_capinfo section.
1258                 */
1259                if (ofl->ofl_oscapinfo)
1260                        cnt++;                  /* DT_SUNW_CAPINFO */

1262                /*
1263                 * Capabilities chain information requires a .SUNW_capchain
1264                 * entry (DT_SUNW_CAPCHAIN), entry size (DT_SUNW_CAPCHAINENT),
1265                 * and total size (DT_SUNW_CAPCHAINSZ).
1266                 */
1267                if (ofl->ofl_oscapchain)
1268                        cnt += 3;

1270                if (flags & FLG_OF_SYMBOLIC)
1271                        cnt++;                  /* DT_SYMBOLIC */

1273                if (ofl->ofl_aslr != 0)         /* DT_SUNW_ASLR */
1274                        cnt++;
1275        }

1277        if (ofl->ofl_flags & FLG_OF_KMOD)
1278                cnt++;

1280 #endif /* ! codereview */
1281        /*
1282         * Account for Architecture dependent .dynamic entries, and defaults.
1283         */
1284        (*ld_targ.t_mr.mr_mach_make_dynamic)(ofl, &cnt);

1286        /*
1287         * DT_FLAGS, DT_FLAGS_1, DT_SUNW_STRPAD, and DT_NULL. Also,
1288         * allow room for the unused extra DT_NULLs. These are included
1289         * to allow an ELF editor room to add items later.
1290         */
1291        cnt += 4 + DYNAMIC_EXTRA_ELTS;

1293        /*
1294         * DT_SUNW_LDMACH. Used to hold the ELF machine code of the
1295         * linker that produced the output object. This information
1296         * allows us to determine whether a given object was linked
1297         * natively, or by a linker running on a different type of
1298         * system. This information can be valuable if one suspects
1299         * that a problem might be due to alignment or byte order issues.
1300         */
1301        cnt++;

1303        /*
1304         * Determine the size of the section from the number of entries.
1305         */
1306        size = cnt * (size_t)shdr->sh_entsize;

1308        shdr->sh_size = (Xword)size;
1309        data->d_size = size;

1311        /*
1312         * There are several tags that are specific to the Solaris osabi
1313         * range which we unconditionally put into any dynamic section
1314         * we create (e.g. DT_SUNW_STRPAD or DT_SUNW_LDMACH). As such,
1315         * any Solaris object with a dynamic section should be tagged as
```

```
1316          * ELFOSABI_SOLARIS.
1317          */
1318         ofl->ofl_flags |= FLG_OF_OSABI;

1320         return ((uintptr_t)ofl->ofl_osdynamic);
1321 }

1323 /*
1324  * Build the GOT section and its associated relocation entries.
1325  */
1326 uintptr_t
1327 ld_make_got(Ofl_desc *ofl)
1328 {
1329         Elf_Data        *data;
1330         Shdr    *shdr;
1331         Is_desc *isec;
1332         size_t  size = (size_t)ofl->ofl_gotcnt * ld_targ.t_m.m_got_entsize;
1333         size_t  rsize = (size_t)ofl->ofl_relocgotsz;

1335         if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_GOT), 0,
1336             &isec, &shdr, &data) == S_ERROR)
1337                 return (S_ERROR);

1339         data->d_size = size;

1341         shdr->sh_flags |= SHF_WRITE;
1342         shdr->sh_size = (Xword)size;
1343         shdr->sh_entsize = ld_targ.t_m.m_got_entsize;

1345         ofl->ofl_osgot = ld_place_section(ofl, isec, NULL,
1346             ld_targ.t_id.id_got, NULL);
1347         if (ofl->ofl_osgot == (Os_desc *)S_ERROR)
1348                 return (S_ERROR);

1350         ofl->ofl_osgot->os_szoutrels = (Xword)rsize;

1352         return (1);
1353 }

1355 /*
1356  * Build an interpreter section.
1357  */
1358 static uintptr_t
1359 make_interp(Ofl_desc *ofl)
1360 {
1361         Shdr            *shdr;
1362         Elf_Data        *data;
1363         Is_desc         *isec;
1364         const char      *iname = ofl->ofl_interp;
1365         size_t          size;

1367         /*
1368          * If -z nointerp is in effect, don't create an interpreter section.
1369          */
1370         if (ofl->ofl_flags1 & FLG_OF1_NOINTRP)
1371                 return (1);

1373         /*
1374          * An .interp section is always created for a dynamic executable.
1375          * A user can define the interpreter to use.  This definition overrides
1376          * the default that would be recorded in an executable, and triggers
1377          * the creation of an .interp section in any other object.  Presumably
1378          * the user knows what they are doing.  Refer to the generic ELF ABI
1379          * section 5-4, and the ld(1) -I option.
1380          */
1381         if (((ofl->ofl_flags & (FLG_OF_DYNAMIC | FLG_OF_EXEC |
```



```
1382             FLG_OF_RELOBJ)) != (FLG_OF_DYNAMIC | FLG_OF_EXEC)) && !iname)
1383                 return (1);

1385         /*
1386          * In the case of a dynamic executable, supply a default interpreter
1387          * if the user has not specified their own.
1388          */
1389         if (iname == NULL)
1390                 iname = ofl->ofl_interp = ld_targ.t_m.m_def_interp;

1392         size = strlen(iname) + 1;

1394         if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_INTERP), 0,
1395             &isec, &shdr, &data) == S_ERROR)
1396                 return (S_ERROR);

1398         data->d_size = size;
1399         shdr->sh_size = (Xword)size;
1400         data->d_align = shdr->sh_addralign = 1;

1402         ofl->ofl_osinterp =
1403             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_interp, NULL);
1404         return ((uintptr_t)ofl->ofl_osinterp);
1405 }

1407 /*
1408  * Common function used to build the SHT_SUNW_versym section, SHT_SUNW_syminfo
1409  * section, and SHT_SUNW_capinfo section.  Each of these sections provide
1410  * additional symbol information, and their size parallels the associated
1411  * symbol table.
1412  */
1413 static Os_desc *
1414 make_sym_sec(Ofl_desc *ofl, const char *sectname, Word stype, int ident)
1415 {
1416         Shdr            *shdr;
1417         Elf_Data        *data;
1418         Is_desc         *isec;

1420         /*
1421          * We don't know the size of this section yet, so set it to 0.  The
1422          * size gets filled in after the associated symbol table is sized.
1423          */
1424         if (new_section(ofl, stype, sectname, 0, &isec, &shdr, &data) ==
1425             S_ERROR)
1426                 return ((Os_desc *)S_ERROR);

1428         return (ld_place_section(ofl, isec, NULL, ident, NULL));
1429 }

1431 /*
1432  * Determine whether a symbol capability is redundant because the object
1433  * capabilities are more restrictive.
1434  */
1435 inline static int
1436 is_cap_redundant(Objcapset *ocapset, Objcapset *scapset)
1437 {
1438         Alist           *oalp, *salp;
1439         elfcap_mask_t   omsk, smsk;

1441         /*
1442          * Inspect any platform capabilities.  If the object defines platform
1443          * capabilities, then the object will only be loaded for those
1444          * platforms.  A symbol capability set that doesn't define the same
1445          * platforms is redundant, and a symbol capability that does not provide
1446          * at least one platform name that matches a platform name in the object
1447          * capabilities will never execute (as the object wouldn't have been
```

```
1448          * loaded).
1449          */
1450         oalp = ocapset->oc_plat.cl_val;
1451         salp = scapset->oc_plat.cl_val;
1452         if (oalp && ((salp == NULL) || cap_names_match(oalp, salp)))
1453                 return (1);

1455         /*
1456          * If the symbol capability set defines platforms, and the object
1457          * doesn't, then the symbol set is more restrictive.
1458          */
1459         if (salp && (oalp == NULL))
1460                 return (0);

1462         /*
1463          * Next, inspect any machine name capabilities.  If the object defines
1464          * machine name capabilities, then the object will only be loaded for
1465          * those machines.  A symbol capability set that doesn't define the same
1466          * machine names is redundant, and a symbol capability that does not
1467          * provide at least one machine name that matches a machine name in the
1468          * object capabilities will never execute (as the object wouldn't have
1469          * been loaded).
1470          */
1471         oalp = ocapset->oc_plat.cl_val;
1472         salp = scapset->oc_plat.cl_val;
1473         if (oalp && ((salp == NULL) || cap_names_match(oalp, salp)))
1474                 return (1);

1476         /*
1477          * If the symbol capability set defines machine names, and the object
1478          * doesn't, then the symbol set is more restrictive.
1479          */
1480         if (salp && (oalp == NULL))
1481                 return (0);

1483         /*
1484          * Next, inspect any hardware capabilities.  If the objects hardware
1485          * capabilities are greater than or equal to that of the symbols
1486          * capabilities, then the symbol capability set is redundant.  If the
1487          * symbols hardware capabilities are greater that the objects, then the
1488          * symbol set is more restrictive.
1489          *
1490          * Note that this is a somewhat arbitrary definition, as each capability
1491          * bit is independent of the others, and some of the higher order bits
1492          * could be considered to be less important than lower ones.  However,
1493          * this is the only reasonable non-subjective definition.
1494          */
1495         omsk = ocapset->oc_hw_2.cm_val;
1496         smsk = scapset->oc_hw_2.cm_val;
1497         if ((omsk > smsk) || (omsk && (omsk == smsk)))
1498                 return (1);
1499         if (omsk < smsk)
1500                 return (0);

1502         /*
1503          * Finally, inspect the remaining hardware capabilities.
1504          */
1505         omsk = ocapset->oc_hw_1.cm_val;
1506         smsk = scapset->oc_hw_1.cm_val;
1507         if ((omsk > smsk) || (omsk && (omsk == smsk)))
1508                 return (1);

1510         return (0);
1511 }

1513 /*
```



```
1514  * Capabilities values might have been assigned excluded values.  These
1515  * excluded values should be removed before calculating any capabilities
1516  * sections size.
1517  */
1518 static void
1519 capmask_value(Lm_list *lml, Word type, Capmask *capmask, int *title)
1520 {
1521         /*
1522          * First determine whether any bits should be excluded.
1523          */
1524         if ((capmask->cm_val & capmask->cm_exc) == 0)
1525                 return;

1527         DBG_CALL(Dbg_cap_post_title(lml, title));

1529         DBG_CALL(Dbg_cap_val_entry(lml, DBG_STATE_CURRENT, type,
1530             capmask->cm_val, ld_targ.t_m.m_mach));
1531         DBG_CALL(Dbg_cap_val_entry(lml, DBG_STATE_EXCLUDE, type,
1532             capmask->cm_exc, ld_targ.t_m.m_mach));

1534         capmask->cm_val &= ~capmask->cm_exc;

1536         DBG_CALL(Dbg_cap_val_entry(lml, DBG_STATE_RESOLVED, type,
1537             capmask->cm_val, ld_targ.t_m.m_mach));
1538 }

1540 static void
1541 capstr_value(Lm_list *lml, Word type, Caplist *caplist, int *title)
1542 {
1543         Aliste  idx1, idx2;
1544         char    *estr;
1545         Capstr  *capstr;
1546         Boolean found = FALSE;

1548         /*
1549          * First determine whether any strings should be excluded.
1550          */
1551         for (APLIST_TRAVERSE(caplist->cl_exc, idx1, estr)) {
1552                 for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1553                         if (strcmp(estr, capstr->cs_str) == 0) {
1554                                 found = TRUE;
1555                                 break;
1556                         }
1557                 }
1558         }

1560         if (found == FALSE)
1561                 return;

1563         /*
1564          * Traverse the current strings, then delete the excluded strings,
1565          * and finally display the resolved strings.
1566          */
1567         if (DBG_ENABLED) {
1568                 Dbg_cap_post_title(lml, title);
1569                 for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1570                         Dbg_cap_ptr_entry(lml, DBG_STATE_CURRENT, type,
1571                             capstr->cs_str);
1572                 }
1573         }
1574         for (APLIST_TRAVERSE(caplist->cl_exc, idx1, estr)) {
1575                 for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1576                         if (strcmp(estr, capstr->cs_str) == 0) {
1577                                 DBG_CALL(Dbg_cap_ptr_entry(lml,
1578                                     DBG_STATE_EXCLUDE, type, capstr->cs_str));
1579                                 alist_delete(caplist->cl_val, &idx2);
```

```
1580                                    break;
1581                            }
1582                    }
1583            }
1584            if (DBG_ENABLED) {
1585                    for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1586                            Dbg_cap_ptr_entry(lml, DBG_STATE_RESOLVED, type,
1587                                capstr->cs_str);
1588                    }
1589            }
1590    }

1592    /*
1593     * Build a capabilities section.
1594     */
1595    #define CAP_UPDATE(cap, capndx, tag, val)       \
1596            cap->c_tag = tag; \
1597            cap->c_un.c_val = val; \
1598            cap++, capndx++;

1600    static uintptr_t
1601    make_cap(Ofl_desc *ofl, Word shtype, const char *shname, int ident)
1602    {
1603            Shdr            *shdr;
1604            Elf_Data        *data;
1605            Is_desc         *isec;
1606            Cap             *cap;
1607            size_t          size = 0;
1608            Word            capndx = 0;
1609            Str_tbl         *strtbl;
1610            Objcapset       *ocapset = &ofl->ofl_ocapset;
1611            Aliste          idx1;
1612            Capstr          *capstr;
1613            int             title = 0;

1615            /*
1616             * Determine which string table to use for any CA_SUNW_MACH,
1617             * CA_SUNW_PLAT, or CA_SUNW_ID strings.
1618             */
1619            if (OFL_IS_STATIC_OBJ(ofl))
1620                    strtbl = ofl->ofl_strtab;
1621            else
1622                    strtbl = ofl->ofl_dynstrtab;

1624            /*
1625             * If symbol capabilities have been requested, but none have been
1626             * created, warn the user.  This scenario can occur if none of the
1627             * input relocatable objects defined any object capabilities.
1628             */
1629            if ((ofl->ofl_flags & FLG_OF_OTOSCAP) && (ofl->ofl_capsymcnt == 0))
1630                    ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_CAP_NOSYMSFOUND));

1632            /*
1633             * If symbol capabilities have been collected, but no symbols are left
1634             * referencing these capabilities, promote the capability groups back
1635             * to an object capability definition.
1636             */
1637            if ((ofl->ofl_flags & FLG_OF_OTOSCAP) && ofl->ofl_capsymcnt &&
1638                (ofl->ofl_capfamilies == NULL)) {
1639                    ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_CAP_NOSYMSFOUND));
1640                    ld_cap_move_symtoobj(ofl);
1641                    ofl->ofl_capsymcnt = 0;
1642                    ofl->ofl_capgroups = NULL;
1643                    ofl->ofl_flags &= ~FLG_OF_OTOSCAP;
1644            }
```



```
1646            /*
1647             * Remove any excluded capabilities.
1648             */
1649            capstr_value(ofl->ofl_lml, CA_SUNW_PLAT, &ocapset->oc_plat, &title);
1650            capstr_value(ofl->ofl_lml, CA_SUNW_MACH, &ocapset->oc_mach, &title);
1651            capmask_value(ofl->ofl_lml, CA_SUNW_HW_2, &ocapset->oc_hw_2, &title);
1652            capmask_value(ofl->ofl_lml, CA_SUNW_HW_1, &ocapset->oc_hw_1, &title);
1653            capmask_value(ofl->ofl_lml, CA_SUNW_SF_1, &ocapset->oc_sf_1, &title);

1655            /*
1656             * Determine how many entries are required for any object capabilities.
1657             */
1658            size += alist_nitems(ocapset->oc_plat.cl_val);
1659            size += alist_nitems(ocapset->oc_mach.cl_val);
1660            if (ocapset->oc_hw_2.cm_val)
1661                    size++;
1662            if (ocapset->oc_hw_1.cm_val)
1663                    size++;
1664            if (ocapset->oc_sf_1.cm_val)
1665                    size++;

1667            /*
1668             * Only identify a capabilities group if the group has content.  If a
1669             * capabilities identifier exists, and no other capabilities have been
1670             * supplied, remove the identifier.  This scenario could exist if a
1671             * user mistakenly defined a lone identifier, or if an identified group
1672             * was overridden so as to clear the existing capabilities and the
1673             * identifier was not also cleared.
1674             */
1675            if (ocapset->oc_id.cs_str) {
1676                    if (size)
1677                            size++;
1678                    else
1679                            ocapset->oc_id.cs_str = NULL;
1680            }
1681            if (size)
1682                    size++;                 /* Add CA_SUNW_NULL */

1684            /*
1685             * Determine how many entries are required for any symbol capabilities.
1686             */
1687            if (ofl->ofl_capsymcnt) {
1688                    /*
1689                     * If there are no object capabilities, a CA_SUNW_NULL entry
1690                     * is required before any symbol capabilities.
1691                     */
1692                    if (size == 0)
1693                            size++;
1694                    size += ofl->ofl_capsymcnt;
1695            }

1697            if (size == 0)
1698                    return (NULL);

1700            if (new_section(ofl, shtype, shname, size, &isec,
1701                &shdr, &data) == S_ERROR)
1702                    return (S_ERROR);

1704            if ((data->d_buf = libld_malloc(shdr->sh_size)) == NULL)
1705                    return (S_ERROR);

1707            cap = (Cap *)data->d_buf;

1709            /*
1710             * Fill in any object capabilities.  If there is an identifier, then the
1711             * identifier comes first.  The remaining items follow in precedence
```

```
1712          * order, although the order isn't important for runtime verification.
1713          */
1714         if (ocapset->oc_id.cs_str) {
1715                 ofl->ofl_flags |= FLG_OF_CAPSTRS;
1716                 if (st_insert(strtbl, ocapset->oc_id.cs_str) == -1)
1717                         return (S_ERROR);
1718                 ocapset->oc_id.cs_ndx = capndx;
1719                 CAP_UPDATE(cap, capndx, CA_SUNW_ID, 0);
1720         }
1721         if (ocapset->oc_plat.cl_val) {
1722                 ofl->ofl_flags |= (FLG_OF_PTCAP | FLG_OF_CAPSTRS);

1724                 /*
1725                  * Insert any platform name strings in the appropriate string
1726                  * table.  The capability value can't be filled in yet, as the
1727                  * final offset of the strings isn't known until later.
1728                  */
1729                 for (ALIST_TRAVERSE(ocapset->oc_plat.cl_val, idx1, capstr)) {
1730                         if (st_insert(strtbl, capstr->cs_str) == -1)
1731                                 return (S_ERROR);
1732                         capstr->cs_ndx = capndx;
1733                         CAP_UPDATE(cap, capndx, CA_SUNW_PLAT, 0);
1734                 }
1735         }
1736         if (ocapset->oc_mach.cl_val) {
1737                 ofl->ofl_flags |= (FLG_OF_PTCAP | FLG_OF_CAPSTRS);

1739                 /*
1740                  * Insert the machine name strings in the appropriate string
1741                  * table.  The capability value can't be filled in yet, as the
1742                  * final offset of the strings isn't known until later.
1743                  */
1744                 for (ALIST_TRAVERSE(ocapset->oc_mach.cl_val, idx1, capstr)) {
1745                         if (st_insert(strtbl, capstr->cs_str) == -1)
1746                                 return (S_ERROR);
1747                         capstr->cs_ndx = capndx;
1748                         CAP_UPDATE(cap, capndx, CA_SUNW_MACH, 0);
1749                 }
1750         }
1751         if (ocapset->oc_hw_2.cm_val) {
1752                 ofl->ofl_flags |= FLG_OF_PTCAP;
1753                 CAP_UPDATE(cap, capndx, CA_SUNW_HW_2, ocapset->oc_hw_2.cm_val);
1754         }
1755         if (ocapset->oc_hw_1.cm_val) {
1756                 ofl->ofl_flags |= FLG_OF_PTCAP;
1757                 CAP_UPDATE(cap, capndx, CA_SUNW_HW_1, ocapset->oc_hw_1.cm_val);
1758         }
1759         if (ocapset->oc_sf_1.cm_val) {
1760                 ofl->ofl_flags |= FLG_OF_PTCAP;
1761                 CAP_UPDATE(cap, capndx, CA_SUNW_SF_1, ocapset->oc_sf_1.cm_val);
1762         }
1763         CAP_UPDATE(cap, capndx, CA_SUNW_NULL, 0);

1765         /*
1766          * Fill in any symbol capabilities.
1767          */
1768         if (ofl->ofl_capgroups) {
1769                 Cap_group       *cgp;

1771                 for (APLIST_TRAVERSE(ofl->ofl_capgroups, idx1, cgp)) {
1772                         Objcapset       *scapset = &cgp->cg_set;
1773                         Aliste          idx2;
1774                         Is_desc         *isp;

1776                         cgp->cg_ndx = capndx;
```



```
1778                         if (scapset->oc_id.cs_str) {
1779                                 ofl->ofl_flags |= FLG_OF_CAPSTRS;
1780                                 /*
1781                                  * Insert the identifier string in the
1782                                  * appropriate string table.  The capability
1783                                  * value can't be filled in yet, as the final
1784                                  * offset of the string isn't known until later.
1785                                  */
1786                                 if (st_insert(strtbl,
1787                                     scapset->oc_id.cs_str) == -1)
1788                                         return (S_ERROR);
1789                                 scapset->oc_id.cs_ndx = capndx;
1790                                 CAP_UPDATE(cap, capndx, CA_SUNW_ID, 0);
1791                         }

1793                         if (scapset->oc_plat.cl_val) {
1794                                 ofl->ofl_flags |= FLG_OF_CAPSTRS;

1796                                 /*
1797                                  * Insert the platform name string in the
1798                                  * appropriate string table.  The capability
1799                                  * value can't be filled in yet, as the final
1800                                  * offset of the string isn't known until later.
1801                                  */
1802                                 for (ALIST_TRAVERSE(scapset->oc_plat.cl_val,
1803                                     idx2, capstr)) {
1804                                         if (st_insert(strtbl,
1805                                             capstr->cs_str) == -1)
1806                                                 return (S_ERROR);
1807                                         capstr->cs_ndx = capndx;
1808                                         CAP_UPDATE(cap, capndx,
1809                                             CA_SUNW_PLAT, 0);
1810                                 }
1811                         }
1812                         if (scapset->oc_mach.cl_val) {
1813                                 ofl->ofl_flags |= FLG_OF_CAPSTRS;

1815                                 /*
1816                                  * Insert the machine name string in the
1817                                  * appropriate string table.  The capability
1818                                  * value can't be filled in yet, as the final
1819                                  * offset of the string isn't known until later.
1820                                  */
1821                                 for (ALIST_TRAVERSE(scapset->oc_mach.cl_val,
1822                                     idx2, capstr)) {
1823                                         if (st_insert(strtbl,
1824                                             capstr->cs_str) == -1)
1825                                                 return (S_ERROR);
1826                                         capstr->cs_ndx = capndx;
1827                                         CAP_UPDATE(cap, capndx,
1828                                             CA_SUNW_MACH, 0);
1829                                 }
1830                         }
1831                         if (scapset->oc_hw_2.cm_val) {
1832                                 CAP_UPDATE(cap, capndx, CA_SUNW_HW_2,
1833                                     scapset->oc_hw_2.cm_val);
1834                         }
1835                         if (scapset->oc_hw_1.cm_val) {
1836                                 CAP_UPDATE(cap, capndx, CA_SUNW_HW_1,
1837                                     scapset->oc_hw_1.cm_val);
1838                         }
1839                         if (scapset->oc_sf_1.cm_val) {
1840                                 CAP_UPDATE(cap, capndx, CA_SUNW_SF_1,
1841                                     scapset->oc_sf_1.cm_val);
1842                         }
1843                         CAP_UPDATE(cap, capndx, CA_SUNW_NULL, 0);
```

```
1845                        /*
1846                         * If any object capabilities are available, determine
1847                         * whether these symbol capabilities are less
1848                         * restrictive, and hence redundant.
1849                         */
1850                        if (((ofl->ofl_flags & FLG_OF_PTCAP) == 0) ||
1851                            (is_cap_redundant(ocapset, scapset) == 0))
1852                                continue;

1854                        /*
1855                         * Indicate any files that provide redundant symbol
1856                         * capabilities.
1857                         */
1858                        for (APLIST_TRAVERSE(cgp->cg_secs, idx2, isp)) {
1859                                ld_eprintf(ofl, ERR_WARNING,
1860                                    MSG_INTL(MSG_CAP_REDUNDANT),
1861                                    isp->is_file->ifl_name,
1862                                    EC_WORD(isp->is_scnndx), isp->is_name);
1863                        }
1864                }
1865        }

1867        /*
1868         * If capabilities strings are required, the sh_info field of the
1869         * section header will be set to the associated string table.
1870         */
1871        if (ofl->ofl_flags & FLG_OF_CAPSTRS)
1872                shdr->sh_flags |= SHF_INFO_LINK;

1874        /*
1875         * Place these capabilities in the output file.
1876         */
1877        if ((ofl->ofl_oscap = ld_place_section(ofl, isec,
1878            NULL, ident, NULL)) == (Os_desc *)S_ERROR)
1879                return (S_ERROR);

1881        /*
1882         * If symbol capabilities are required, then a .SUNW_capinfo section is
1883         * also created.  This table will eventually be sized to match the
1884         * associated symbol table.
1885         */
1886        if (ofl->ofl_capfamilies) {
1887                if ((ofl->ofl_oscapinfo = make_sym_sec(ofl,
1888                    MSG_ORIG(MSG_SCN_SUNWCAPINFO), SHT_SUNW_capinfo,
1889                    ld_targ.t_id.id_capinfo)) == (Os_desc *)S_ERROR)
1890                        return (S_ERROR);

1892                /*
1893                 * If we're generating a dynamic object, capabilities family
1894                 * members are maintained in a .SUNW_capchain section.
1895                 */
1896                if (ofl->ofl_capchaincnt &&
1897                    ((ofl->ofl_flags & FLG_OF_RELOBJ) == 0)) {
1898                        if (new_section(ofl, SHT_SUNW_capchain,
1899                            MSG_ORIG(MSG_SCN_SUNWCAPCHAIN),
1900                            ofl->ofl_capchaincnt, &isec, &shdr,
1901                            &data) == S_ERROR)
1902                                return (S_ERROR);

1904                        ofl->ofl_oscapchain = ld_place_section(ofl, isec,
1905                            NULL, ld_targ.t_id.id_capchain, NULL);
1906                        if (ofl->ofl_oscapchain == (Os_desc *)S_ERROR)
1907                                return (S_ERROR);

1909                }
```



```
1910        }
1911        return (1);
1912 }
1913 #undef  CAP_UPDATE

1915 /*
1916  * Build the PLT section and its associated relocation entries.
1917  */
1918 static uintptr_t
1919 make_plt(Ofl_desc *ofl)
1920 {
1921        Shdr            *shdr;
1922        Elf_Data        *data;
1923        Is_desc         *isec;
1924        size_t          size = ld_targ.t_m.m_plt_reservsz +
1925            (((size_t)ofl->ofl_pltcnt + (size_t)ofl->ofl_pltpad) *
1926            ld_targ.t_m.m_plt_entsize);
1927        size_t          rsize = (size_t)ofl->ofl_relocpltsz;

1929        /*
1930         * On sparc, account for the NOP at the end of the plt.
1931         */
1932        if (ld_targ.t_m.m_mach == LD_TARG_BYCLASS(EM_SPARC, EM_SPARCV9))
1933                size += sizeof (Word);

1935        if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_PLT), 0,
1936            &isec, &shdr, &data) == S_ERROR)
1937                return (S_ERROR);

1939        data->d_size = size;
1940        data->d_align = ld_targ.t_m.m_plt_align;

1942        shdr->sh_flags = ld_targ.t_m.m_plt_shf_flags;
1943        shdr->sh_size = (Xword)size;
1944        shdr->sh_addralign = ld_targ.t_m.m_plt_align;
1945        shdr->sh_entsize = ld_targ.t_m.m_plt_entsize;

1947        ofl->ofl_osplt = ld_place_section(ofl, isec, NULL,
1948            ld_targ.t_id.id_plt, NULL);
1949        if (ofl->ofl_osplt == (Os_desc *)S_ERROR)
1950                return (S_ERROR);

1952        ofl->ofl_osplt->os_szoutrels = (Xword)rsize;

1954        return (1);
1955 }

1957 /*
1958  * Make the hash table.  Only built for dynamic executables and shared
1959  * libraries, and provides hashed lookup into the global symbol table
1960  * (.dynsym) for the run-time linker to resolve symbol lookups.
1961  */
1962 static uintptr_t
1963 make_hash(Ofl_desc *ofl)
1964 {
1965        Shdr            *shdr;
1966        Elf_Data        *data;
1967        Is_desc         *isec;
1968        size_t          size;
1969        Word            nsyms = ofl->ofl_globcnt;
1970        size_t          cnt;

1972        /*
1973         * Allocate section header structures. We set entcnt to 0
1974         * because it's going to change after we place this section.
1975         */
```

```
1976         if (new_section(ofl, SHT_HASH, MSG_ORIG(MSG_SCN_HASH), 0,
1977             &isec, &shdr, &data) == S_ERROR)
1978                 return (S_ERROR);
1979
1980         /*
1981          * Place the section first since it will affect the local symbol
1982          * count.
1983          */
1984         ofl->ofl_oshash =
1985             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_hash, NULL);
1986         if (ofl->ofl_oshash == (Os_desc *)S_ERROR)
1987                 return (S_ERROR);
1988
1989         /*
1990          * Calculate the number of output hash buckets.
1991          */
1992         ofl->ofl_hashbkts = findprime(nsyms);
1993
1994         /*
1995          * The size of the hash table is determined by
1996          *
1997          *      i.      the initial nbucket and nchain entries (2)
1998          *      ii.     the number of buckets (calculated above)
1999          *      iii.    the number of chains (this is based on the number of
2000          *              symbols in the .dynsym array).
2001          */
2002         cnt = 2 + ofl->ofl_hashbkts + DYNSYM_ALL_CNT(ofl);
2003         size = cnt * shdr->sh_entsize;
2004
2005         /*
2006          * Finalize the section header and data buffer initialization.
2007          */
2008         if ((data->d_buf = libld_calloc(size, 1)) == NULL)
2009                 return (S_ERROR);
2010         data->d_size = size;
2011         shdr->sh_size = (Xword)size;
2012
2013         return (1);
2014 }
2015
2016 /*
2017  * Generate the standard symbol table.  Contains all locals and globals,
2018  * and resides in a non-allocatable section (ie. it can be stripped).
2019  */
2020 static uintptr_t
2021 make_symtab(Ofl_desc *ofl)
2022 {
2023         Shdr            *shdr;
2024         Elf_Data        *data;
2025         Is_desc         *isec;
2026         Is_desc         *xisec = 0;
2027         size_t          size;
2028         Word            symcnt;
2029
2030         /*
2031          * Create the section headers. Note that we supply an ent_cnt
2032          * of 0. We won't know the count until the section has been placed.
2033          */
2034         if (new_section(ofl, SHT_SYMTAB, MSG_ORIG(MSG_SCN_SYMTAB), 0,
2035             &isec, &shdr, &data) == S_ERROR)
2036                 return (S_ERROR);
2037
2038         /*
2039          * Place the section first since it will affect the local symbol
2040          * count.
2041          */
```



```
2042         if ((ofl->ofl_ossymtab = ld_place_section(ofl, isec, NULL,
2043             ld_targ.t_id.id_symtab, NULL)) == (Os_desc *)S_ERROR)
2044                 return (S_ERROR);
2045
2046         /*
2047          * At this point we've created all but the 'shstrtab' section.
2048          * Determine if we have to use 'Extended Sections'.  If so - then
2049          * also create a SHT_SYMTAB_SHNDX section.
2050          */
2051         if ((ofl->ofl_shdrcnt + 1) >= SHN_LORESERVE) {
2052                 Shdr            *xshdr;
2053                 Elf_Data        *xdata;
2054
2055                 if (new_section(ofl, SHT_SYMTAB_SHNDX,
2056                     MSG_ORIG(MSG_SCN_SYMTAB_SHNDX), 0, &xisec,
2057                     &xshdr, &xdata) == S_ERROR)
2058                         return (S_ERROR);
2059
2060                 if ((ofl->ofl_ossymshndx = ld_place_section(ofl, xisec, NULL,
2061                     ld_targ.t_id.id_symtab_ndx, NULL)) == (Os_desc *)S_ERROR)
2062                         return (S_ERROR);
2063         }
2064
2065         /*
2066          * Calculated number of symbols, which need to be augmented by
2067          * the (yet to be created) .shstrtab entry.
2068          */
2069         symcnt = (size_t)(1 + SYMTAB_ALL_CNT(ofl));
2070         size = symcnt * shdr->sh_entsize;
2071
2072         /*
2073          * Finalize the section header and data buffer initialization.
2074          */
2075         data->d_size = size;
2076         shdr->sh_size = (Xword)size;
2077
2078         /*
2079          * If we created a SHT_SYMTAB_SHNDX - then set it's sizes too.
2080          */
2081         if (xisec) {
2082                 size_t  xsize = symcnt * sizeof (Word);
2083
2084                 xisec->is_indata->d_size = xsize;
2085                 xisec->is_shdr->sh_size = (Xword)xsize;
2086         }
2087
2088         return (1);
2089 }
2090
2091 /*
2092  * Build a dynamic symbol table. These tables reside in the text
2093  * segment of a dynamic executable or shared library.
2094  *
2095  *      .SUNW_ldynsym contains local function symbols
2096  *      .dynsym contains only globals symbols
2097  *
2098  * The two tables are created adjacent to each other, with .SUNW_ldynsym
2099  * coming first.
2100  */
2101 static uintptr_t
2102 make_dynsym(Ofl_desc *ofl)
2103 {
2104         Shdr            *shdr, *lshdr;
2105         Elf_Data        *data, *ldata;
2106         Is_desc         *isec, *lisec;
2107         size_t          size;
```

```
2108         Xword           cnt;
2109         int             allow_ldynsym;
2110
2111         /*
2112          * Unless explicitly disabled, always produce a .SUNW_ldynsym section
2113          * when it is allowed by the file type, even if the resulting
2114          * table only ends up with a single STT_FILE in it. There are
2115          * two reasons: (1) It causes the generation of the DT_SUNW_SYMTAB
2116          * entry in the .dynamic section, which is something we would
2117          * like to encourage, and (2) Without it, we cannot generate
2118          * the associated .SUNW_dyn[sym|tls]sort sections, which are of
2119          * value to DTrace.
2120          *
2121          * In practice, it is extremely rare for an object not to have
2122          * local symbols for .SUNW_ldynsym, so 99% of the time, we'd be
2123          * doing it anyway.
2124          */
2125         allow_ldynsym = OFL_ALLOW_LDYNSYM(ofl);
2126
2127         /*
2128          * Create the section headers. Note that we supply an ent_cnt
2129          * of 0. We won't know the count until the section has been placed.
2130          */
2131         if (allow_ldynsym && new_section(ofl, SHT_SUNW_LDYNSYM,
2132             MSG_ORIG(MSG_SCN_LDYNSYM), 0, &lisec, &lshdr, &ldata) == S_ERROR)
2133                 return (S_ERROR);
2134
2135         if (new_section(ofl, SHT_DYNSYM, MSG_ORIG(MSG_SCN_DYNSYM), 0,
2136             &isec, &shdr, &data) == S_ERROR)
2137                 return (S_ERROR);
2138
2139         /*
2140          * Place the section(s) first since it will affect the local symbol
2141          * count.
2142          */
2143         if (allow_ldynsym &&
2144             ((ofl->ofl_osldynsym = ld_place_section(ofl, lisec, NULL,
2145             ld_targ.t_id.id_ldynsym, NULL)) == (Os_desc *)S_ERROR))
2146                 return (S_ERROR);
2147         ofl->ofl_osdynsym =
2148             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_dynsym, NULL);
2149         if (ofl->ofl_osdynsym == (Os_desc *)S_ERROR)
2150                 return (S_ERROR);
2151
2152         cnt = DYNSYM_ALL_CNT(ofl);
2153         size = (size_t)cnt * shdr->sh_entsize;
2154
2155         /*
2156          * Finalize the section header and data buffer initialization.
2157          */
2158         data->d_size = size;
2159         shdr->sh_size = (Xword)size;
2160
2161         /*
2162          * An ldynsym contains local function symbols. It is not
2163          * used for linking, but if present, serves to allow better
2164          * stack traces to be generated in contexts where the symtab
2165          * is not available. (dladdr(), or stripped executable/library files).
2166          */
2167         if (allow_ldynsym) {
2168                 cnt = 1 + ofl->ofl_dynlocscnt + ofl->ofl_dynscopecnt;
2169                 size = (size_t)cnt * shdr->sh_entsize;
2170
2171                 ldata->d_size = size;
2172                 lshdr->sh_size = (Xword)size;
2173         }
```



```
2175         return (1);
2176 }
2177
2178 /*
2179  * Build .SUNW_dynsymsort and/or .SUNW_dyntlssort sections. These are
2180  * index sections for the .SUNW_ldynsym/.dynsym pair that present data
2181  * and function symbols sorted by address.
2182  */
2183 static uintptr_t
2184 make_dynsort(Ofl_desc *ofl)
2185 {
2186         Shdr            *shdr;
2187         Elf_Data        *data;
2188         Is_desc         *isec;
2189
2190         /* Only do it if the .SUNW_ldynsym section is present */
2191         if (!OFL_ALLOW_LDYNSYM(ofl))
2192                 return (1);
2193
2194         /* .SUNW_dynsymsort */
2195         if (ofl->ofl_dynsymsortcnt > 0) {
2196                 if (new_section(ofl, SHT_SUNW_symsort,
2197                     MSG_ORIG(MSG_SCN_DYNSYMSORT), ofl->ofl_dynsymsortcnt,
2198                     &isec, &shdr, &data) == S_ERROR)
2199                 return (S_ERROR);
2200
2201                 if ((ofl->ofl_osdynsymsort = ld_place_section(ofl, isec, NULL,
2202                     ld_targ.t_id.id_dynsort, NULL)) == (Os_desc *)S_ERROR)
2203                         return (S_ERROR);
2204         }
2205
2206         /* .SUNW_dyntlssort */
2207         if (ofl->ofl_dyntlssortcnt > 0) {
2208                 if (new_section(ofl, SHT_SUNW_tlssort,
2209                     MSG_ORIG(MSG_SCN_DYNTLSSORT),
2210                     ofl->ofl_dyntlssortcnt, &isec, &shdr, &data) == S_ERROR)
2211                 return (S_ERROR);
2212
2213                 if ((ofl->ofl_osdyntlssort = ld_place_section(ofl, isec, NULL,
2214                     ld_targ.t_id.id_dynsort, NULL)) == (Os_desc *)S_ERROR)
2215                         return (S_ERROR);
2216         }
2217
2218         return (1);
2219 }
2220
2221 /*
2222  * Helper routine for make_dynsym_shndx. Builds a
2223  * a SHT_SYMTAB_SHNDX for .dynsym or .SUNW_ldynsym, without knowing
2224  * which one it is.
2225  */
2226 static uintptr_t
2227 make_dyn_shndx(Ofl_desc *ofl, const char *shname, Os_desc *symtab,
2228     Os_desc **ret_os)
2229 {
2230         Is_desc         *isec;
2231         Is_desc         *dynsymisp;
2232         Shdr            *shdr, *dynshdr;
2233         Elf_Data        *data;
2234
2235         dynsymisp = ld_os_first_isdesc(symtab);
2236         dynshdr = dynsymisp->is_shdr;
2237
2238         if (new_section(ofl, SHT_SYMTAB_SHNDX, shname,
2239             (dynshdr->sh_size / dynshdr->sh_entsize),
```

```
2240             &isec, &shdr, &data) == S_ERROR)
2241                 return (S_ERROR);

2243         if ((*ret_os = ld_place_section(ofl, isec, NULL,
2244             ld_targ.t_id.id_dynsym_ndx, NULL)) == (Os_desc *)S_ERROR)
2245                 return (S_ERROR);

2247         assert(*ret_os);

2249         return (1);
2250 }

2252 /*
2253  * Build a SHT_SYMTAB_SHNDX for the .dynsym, and .SUNW_ldynsym
2254  */
2255 static uintptr_t
2256 make_dynsym_shndx(Ofl_desc *ofl)
2257 {
2258         /*
2259          * If there is a .SUNW_ldynsym, generate a section for its extended
2260          * index section as well.
2261          */
2262         if (OFL_ALLOW_LDYNSYM(ofl)) {
2263                 if (make_dyn_shndx(ofl, MSG_ORIG(MSG_SCN_LDYNSYM_SHNDX),
2264                     ofl->ofl_osldynsym, &ofl->ofl_osldynshndx) == S_ERROR)
2265                         return (S_ERROR);
2266         }

2268         /* The Generate a section for the dynsym */
2269         if (make_dyn_shndx(ofl, MSG_ORIG(MSG_SCN_DYNSYM_SHNDX),
2270             ofl->ofl_osdynsym, &ofl->ofl_osdynshndx) == S_ERROR)
2271                 return (S_ERROR);

2273         return (1);
2274 }


2277 /*
2278  * Build a string table for the section headers.
2279  */
2280 static uintptr_t
2281 make_shstrtab(Ofl_desc *ofl)
2282 {
2283         Shdr            *shdr;
2284         Elf_Data        *data;
2285         Is_desc         *isec;
2286         size_t          size;

2288         if (new_section(ofl, SHT_STRTAB, MSG_ORIG(MSG_SCN_SHSTRTAB),
2289             0, &isec, &shdr, &data) == S_ERROR)
2290                 return (S_ERROR);

2292         /*
2293          * Place the section first, as it may effect the number of section
2294          * headers to account for.
2295          */
2296         ofl->ofl_osshstrtab =
2297             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_note, NULL);
2298         if (ofl->ofl_osshstrtab == (Os_desc *)S_ERROR)
2299                 return (S_ERROR);

2301         size = st_getstrtab_sz(ofl->ofl_shdrsttab);
2302         assert(size > 0);

2304         data->d_size = size;
2305         shdr->sh_size = (Xword)size;
```



```
2307         return (1);
2308 }

2310 /*
2311  * Build a string section for the standard symbol table.
2312  */
2313 static uintptr_t
2314 make_strtab(Ofl_desc *ofl)
2315 {
2316         Shdr            *shdr;
2317         Elf_Data        *data;
2318         Is_desc         *isec;
2319         size_t          size;

2321         /*
2322          * This string table consists of all the global and local symbols.
2323          * Account for null bytes at end of the file name and the beginning
2324          * of section.
2325          */
2326         if (st_insert(ofl->ofl_strtab, ofl->ofl_name) == -1)
2327                 return (S_ERROR);

2329         size = st_getstrtab_sz(ofl->ofl_strtab);
2330         assert(size > 0);

2332         if (new_section(ofl, SHT_STRTAB, MSG_ORIG(MSG_SCN_STRTAB),
2333             0, &isec, &shdr, &data) == S_ERROR)
2334                 return (S_ERROR);

2336         /* Set the size of the data area */
2337         data->d_size = size;
2338         shdr->sh_size = (Xword)size;

2340         ofl->ofl_osstrtab =
2341             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_strtab, NULL);
2342         return ((uintptr_t)ofl->ofl_osstrtab);
2343 }

2345 /*
2346  * Build a string table for the dynamic symbol table.
2347  */
2348 static uintptr_t
2349 make_dynstr(Ofl_desc *ofl)
2350 {
2351         Shdr            *shdr;
2352         Elf_Data        *data;
2353         Is_desc         *isec;
2354         size_t          size;

2356         /*
2357          * If producing a .SUNW_ldynsym, account for the initial STT_FILE
2358          * symbol that precedes the scope reduced global symbols.
2359          */
2360         if (OFL_ALLOW_LDYNSYM(ofl)) {
2361                 if (st_insert(ofl->ofl_dynstrtab, ofl->ofl_name) == -1)
2362                         return (S_ERROR);
2363                 ofl->ofl_dynscopecnt++;
2364         }

2366         /*
2367          * Account for any local, named register symbols.  These locals are
2368          * required for reference from DT_REGISTER .dynamic entries.
2369          */
2370         if (ofl->ofl_regsyms) {
2371                 int     ndx;
```

```
2373                 for (ndx = 0; ndx < ofl->ofl_regsymsno; ndx++) {
2374                         Sym_desc        *sdp;

2376                         if ((sdp = ofl->ofl_regsyms[ndx]) == NULL)
2377                                 continue;

2379                         if (!SYM_IS_HIDDEN(sdp) &&
2380                             (ELF_ST_BIND(sdp->sd_sym->st_info) != STB_LOCAL))
2381                                 continue;

2383                         if (sdp->sd_sym->st_name == NULL)
2384                                 continue;

2386                         if (st_insert(ofl->ofl_dynstrtab, sdp->sd_name) == -1)
2387                                 return (S_ERROR);
2388                 }
2389         }

2391         /*
2392          * Reserve entries for any per-symbol auxiliary/filter strings.
2393          */
2394         if (ofl->ofl_dtsfltrs != NULL) {
2395                 Dfltr_desc      *dftp;
2396                 Aliste          idx;

2398                 for (ALIST_TRAVERSE(ofl->ofl_dtsfltrs, idx, dftp))
2399                         if (st_insert(ofl->ofl_dynstrtab, dftp->dft_str) == -1)
2400                                 return (S_ERROR);
2401         }

2403         size = st_getstrtab_sz(ofl->ofl_dynstrtab);
2404         assert(size > 0);

2406         if (new_section(ofl, SHT_STRTAB, MSG_ORIG(MSG_SCN_DYNSTR),
2407             0, &isec, &shdr, &data) == S_ERROR)
2408                 return (S_ERROR);

2410         /* Make it allocable if necessary */
2411         if (!(ofl->ofl_flags & FLG_OF_RELOBJ))
2412                 shdr->sh_flags |= SHF_ALLOC;

2414         /* Set the size of the data area */
2415         data->d_size = size + DYNSTR_EXTRA_PAD;

2417         shdr->sh_size = (Xword)size;

2419         ofl->ofl_osdynstr =
2420             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_dynstr, NULL);
2421         return ((uintptr_t)ofl->ofl_osdynstr);
2422 }

2424 /*
2425  * Generate an output relocation section which will contain the relocation
2426  * information to be applied to the 'osp' section.
2427  *
2428  * If (osp == NULL) then we are creating the coalesced relocation section
2429  * for an executable and/or a shared object.
2430  */
2431 static uintptr_t
2432 make_reloc(Ofl_desc *ofl, Os_desc *osp)
2433 {
2434         Shdr            *shdr;
2435         Elf_Data        *data;
2436         Is_desc         *isec;
2437         size_t          size;
```



```
2438         Xword           sh_flags;
2439         char            *sectname;
2440         Os_desc         *rosp;
2441         Word            relsize;
2442         const char      *rel_prefix;

2444         /* LINTED */
2445         if (ld_targ.t_m.m_rel_sht_type == SHT_REL) {
2446                 /* REL */
2447                 relsize = sizeof (Rel);
2448                 rel_prefix = MSG_ORIG(MSG_SCN_REL);
2449         } else {
2450                 /* RELA */
2451                 relsize = sizeof (Rela);
2452                 rel_prefix = MSG_ORIG(MSG_SCN_RELA);
2453         }

2455         if (osp) {
2456                 size = osp->os_szoutrels;
2457                 sh_flags = osp->os_shdr->sh_flags;
2458                 if ((sectname = libld_malloc(strlen(rel_prefix) +
2459                     strlen(osp->os_name) + 1)) == 0)
2460                         return (S_ERROR);
2461                 (void) strcpy(sectname, rel_prefix);
2462                 (void) strcat(sectname, osp->os_name);
2463         } else if (ofl->ofl_flags & FLG_OF_COMREL) {
2464                 size = (ofl->ofl_reloccnt - ofl->ofl_reloccntsub) * relsize;
2465                 sh_flags = SHF_ALLOC;
2466                 sectname = (char *)MSG_ORIG(MSG_SCN_SUNWRELOC);
2467         } else {
2468                 size = ofl->ofl_relocrelsz;
2469                 sh_flags = SHF_ALLOC;
2470                 sectname = (char *)rel_prefix;
2471         }

2473         /*
2474          * Keep track of total size of 'output relocations' (to be stored
2475          * in .dynamic)
2476          */
2477         /* LINTED */
2478         ofl->ofl_relocsz += (Xword)size;

2480         if (new_section(ofl, ld_targ.t_m.m_rel_sht_type, sectname, 0, &isec,
2481             &shdr, &data) == S_ERROR)
2482                 return (S_ERROR);

2484         data->d_size = size;

2486         shdr->sh_size = (Xword)size;
2487         if (OFL_ALLOW_DYNSYM(ofl) && (sh_flags & SHF_ALLOC))
2488                 shdr->sh_flags = SHF_ALLOC;

2490         if (osp) {
2491                 /*
2492                  * The sh_info field of the SHT_REL* sections points to the
2493                  * section the relocations are to be applied to.
2494                  */
2495                 shdr->sh_flags |= SHF_INFO_LINK;
2496         }

2498         rosp = ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_rel, NULL);
2499         if (rosp == (Os_desc *)S_ERROR)
2500                 return (S_ERROR);

2502         /*
2503          * Associate this relocation section to the section its going to
```

```
2504          * relocate.
2505          */
2506         if (osp) {
2507                 Aliste  idx;
2508                 Is_desc *risp;

2510                 /*
2511                  * This is used primarily so that we can update
2512                  * SHT_GROUP[sect_no] entries to point to the
2513                  * created output relocation sections.
2514                  */
2515                 for (APLIST_TRAVERSE(osp->os_relisdescs, idx, risp)) {
2516                         risp->is_osdesc = rosp;

2518                         /*
2519                          * If the input relocation section had the SHF_GROUP
2520                          * flag set - propagate it to the output relocation
2521                          * section.
2522                          */
2523                         if (risp->is_shdr->sh_flags & SHF_GROUP) {
2524                                 rosp->os_shdr->sh_flags |= SHF_GROUP;
2525                                 break;
2526                         }
2527                 }
2528                 osp->os_relosdesc = rosp;
2529         } else
2530                 ofl->ofl_osrel = rosp;

2532         /*
2533          * If this is the first relocation section we've encountered save it
2534          * so that the .dynamic entry can be initialized accordingly.
2535          */
2536         if (ofl->ofl_osrelhead == (Os_desc *)0)
2537                 ofl->ofl_osrelhead = rosp;

2539         return (1);
2540 }

2542 /*
2543  * Generate version needed section.
2544  */
2545 static uintptr_t
2546 make_verneed(Ofl_desc *ofl)
2547 {
2548         Shdr            *shdr;
2549         Elf_Data        *data;
2550         Is_desc         *isec;

2552         /*
2553          * verneed sections do not have a constant element size, so the
2554          * value of ent_cnt specified here (0) is meaningless.
2555          */
2556         if (new_section(ofl, SHT_SUNW_verneed, MSG_ORIG(MSG_SCN_SUNWVERSION),
2557             0, &isec, &shdr, &data) == S_ERROR)
2558                 return (S_ERROR);

2560         /* During version processing we calculated the total size. */
2561         data->d_size = ofl->ofl_verneedsz;
2562         shdr->sh_size = (Xword)ofl->ofl_verneedsz;

2564         ofl->ofl_osverneed =
2565             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_version, NULL);
2566         return ((uintptr_t)ofl->ofl_osverneed);
2567 }

2569 /*
```



```
2570  * Generate a version definition section.
2571  *
2572  *  o   the SHT_SUNW_verdef section defines the versions that exist within this
2573  *      image.
2574  */
2575 static uintptr_t
2576 make_verdef(Ofl_desc *ofl)
2577 {
2578         Shdr            *shdr;
2579         Elf_Data        *data;
2580         Is_desc         *isec;
2581         Ver_desc        *vdp;
2582         Str_tbl         *strtab;

2584         /*
2585          * Reserve a string table entry for the base version dependency (other
2586          * dependencies have symbol representations, which will already be
2587          * accounted for during symbol processing).
2588          */
2589         vdp = (Ver_desc *)ofl->ofl_verdesc->apl_data[0];

2591         if (OFL_IS_STATIC_OBJ(ofl))
2592                 strtab = ofl->ofl_strtab;
2593         else
2594                 strtab = ofl->ofl_dynstrtab;

2596         if (st_insert(strtab, vdp->vd_name) == -1)
2597                 return (S_ERROR);

2599         /*
2600          * verdef sections do not have a constant element size, so the
2601          * value of ent_cnt specified here (0) is meaningless.
2602          */
2603         if (new_section(ofl, SHT_SUNW_verdef, MSG_ORIG(MSG_SCN_SUNWVERSION),
2604             0, &isec, &shdr, &data) == S_ERROR)
2605                 return (S_ERROR);

2607         /* During version processing we calculated the total size. */
2608         data->d_size = ofl->ofl_verdefsz;
2609         shdr->sh_size = (Xword)ofl->ofl_verdefsz;

2611         ofl->ofl_osverdef =
2612             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_version, NULL);
2613         return ((uintptr_t)ofl->ofl_osverdef);
2614 }

2616 /*
2617  * This routine is called when -z nopartial is in effect.
2618  */
2619 uintptr_t
2620 ld_make_parexpn_data(Ofl_desc *ofl, size_t size, Xword align)
2621 {
2622         Shdr            *shdr;
2623         Elf_Data        *data;
2624         Is_desc         *isec;
2625         Os_desc         *osp;

2627         if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_DATA), 0,
2628             &isec, &shdr, &data) == S_ERROR)
2629                 return (S_ERROR);

2631         shdr->sh_flags |= SHF_WRITE;
2632         data->d_size = size;
2633         shdr->sh_size = (Xword)size;
2634         if (align != 0) {
2635                 data->d_align = align;
```

```
2636                    shdr->sh_addralign = align;
2637            }

2639            if ((data->d_buf = libld_calloc(size, 1)) == NULL)
2640                    return (S_ERROR);

2642            /*
2643             * Retain handle to this .data input section. Variables using move
2644             * sections (partial initialization) will be redirected here when
2645             * such global references are added and '-z nopartial' is in effect.
2646             */
2647            ofl->ofl_isparexpn = isec;
2648            osp = ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_data, NULL);
2649            if (osp == (Os_desc *)S_ERROR)
2650                    return (S_ERROR);

2652            if (!(osp->os_flags & FLG_OS_OUTREL)) {
2653                    ofl->ofl_dynshdrcnt++;
2654                    osp->os_flags |= FLG_OS_OUTREL;
2655            }
2656            return (1);
2657    }

2659    /*
2660     * Make .sunwmove section
2661     */
2662    uintptr_t
2663    ld_make_sunwmove(Ofl_desc *ofl, int mv_nums)
2664    {
2665            Shdr            *shdr;
2666            Elf_Data        *data;
2667            Is_desc         *isec;
2668            Aliste          idx;
2669            Sym_desc        *sdp;
2670            int             cnt = 1;


2673            if (new_section(ofl, SHT_SUNW_move, MSG_ORIG(MSG_SCN_SUNWMOVE),
2674                mv_nums, &isec, &shdr, &data) == S_ERROR)
2675                    return (S_ERROR);

2677            if ((data->d_buf = libld_calloc(data->d_size, 1)) == NULL)
2678                    return (S_ERROR);

2680            /*
2681             * Copy move entries
2682             */
2683            for (APLIST_TRAVERSE(ofl->ofl_parsyms, idx, sdp)) {
2684                    Aliste          idx2;
2685                    Mv_desc         *mdp;

2687                    if (sdp->sd_flags & FLG_SY_PAREXPN)
2688                            continue;

2690                    for (ALIST_TRAVERSE(sdp->sd_move, idx2, mdp))
2691                            mdp->md_oidx = cnt++;
2692            }

2694            if ((ofl->ofl_osmove = ld_place_section(ofl, isec, NULL, 0, NULL)) ==
2695                (Os_desc *)S_ERROR)
2696                    return (S_ERROR);

2698            return (1);
2699    }

2701    /*
```



```
2702     * Given a relocation descriptor that references a string table
2703     * input section, locate the string referenced and return a pointer
2704     * to it.
2705     */
2706    static const char *
2707    strmerge_get_reloc_str(Ofl_desc *ofl, Rel_desc *rsp)
2708    {
2709            Sym_desc *sdp = rsp->rel_sym;
2710            Xword    str_off;

2712            /*
2713             * In the case of an STT_SECTION symbol, the addend of the
2714             * relocation gives the offset into the string section. For
2715             * other symbol types, the symbol value is the offset.
2716             */

2718            if (ELF_ST_TYPE(sdp->sd_sym->st_info) != STT_SECTION) {
2719                    str_off = sdp->sd_sym->st_value;
2720            } else if ((rsp->rel_flags & FLG_REL_RELA) == FLG_REL_RELA) {
2721                    /*
2722                     * For SHT_RELA, the addend value is found in the
2723                     * rel_raddend field of the relocation.
2724                     */
2725                    str_off = rsp->rel_raddend;
2726            } else {        /* REL and STT_SECTION */
2727                    /*
2728                     * For SHT_REL, the "addend" is not part of the relocation
2729                     * record. Instead, it is found at the relocation target
2730                     * address.
2731                     */
2732                    uchar_t *addr = (uchar_t *)((uintptr_t)rsp->rel_roffset +
2733                        (uintptr_t)rsp->rel_isdesc->is_indata->d_buf);

2735                    if (ld_reloc_targval_get(ofl, rsp, addr, &str_off) == 0)
2736                            return (0);
2737            }

2739            return (str_off + (char *)sdp->sd_isc->is_indata->d_buf);
2740    }

2742    /*
2743     * First pass over the relocation records for string table merging.
2744     * Build lists of relocations and symbols that will need modification,
2745     * and insert the strings they reference into the mstrtab string table.
2746     *
2747     * entry:
2748     *      ofl, osp - As passed to ld_make_strmerge().
2749     *      mstrtab - String table to receive input strings. This table
2750     *              must be in its first (initialization) pass and not
2751     *              yet cooked (st_getstrtab_sz() not yet called).
2752     *      rel_alpp - APlist to receive pointer to any relocation
2753     *              descriptors with STT_SECTION symbols that reference
2754     *              one of the input sections being merged.
2755     *      sym_alpp - APlist to receive pointer to any symbols that reference
2756     *              one of the input sections being merged.
2757     *      rcp - Pointer to cache of relocation descriptors to examine.
2758     *              Either &ofl->ofl_actrels (active relocations)
2759     *              or &ofl->ofl_outrels (output relocations).
2760     *
2761     * exit:
2762     *      On success, rel_alpp and sym_alpp are updated, and
2763     *      any strings in the mergeable input sections referenced by
2764     *      a relocation has been entered into mstrtab. True (1) is returned.
2765     *
2766     *      On failure, False (0) is returned.
2767     */
```

```
2768 static int
2769 strmerge_pass1(Ofl_desc *ofl, Os_desc *osp, Str_tbl *mstrtab,
2770     APlist **rel_alpp, APlist **sym_alpp, Rel_cache *rcp)
2771 {
2772         Aliste          idx;
2773         Rel_cachebuf    *rcbp;
2774         Sym_desc        *sdp;
2775         Sym_desc        *last_sdp = NULL;
2776         Rel_desc        *rsp;
2777         const char      *name;

2779         REL_CACHE_TRAVERSE(rcp, idx, rcbp, rsp) {
2780                 sdp = rsp->rel_sym;
2781                 if ((sdp->sd_isc == NULL) || ((sdp->sd_isc->is_flags &
2782                     (FLG_IS_DISCARD | FLG_IS_INSTRMRG)) != FLG_IS_INSTRMRG) ||
2783                     (sdp->sd_isc->is_osdesc != osp))
2784                         continue;

2786                 /*
2787                  * Remember symbol for use in the third pass. There is no
2788                  * reason to save a given symbol more than once, so we take
2789                  * advantage of the fact that relocations to a given symbol
2790                  * tend to cluster in the list. If this is the same symbol
2791                  * we saved last time, don't bother.
2792                  */
2793                 if (last_sdp != sdp) {
2794                         if (aplist_append(sym_alpp, sdp, AL_CNT_STRMRGSYM) ==
2795                             NULL)
2796                                 return (0);
2797                         last_sdp = sdp;
2798                 }

2800                 /* Enter the string into our new string table */
2801                 name = strmerge_get_reloc_str(ofl, rsp);
2802                 if (st_insert(mstrtab, name) == -1)
2803                         return (0);

2805                 /*
2806                  * If this is an STT_SECTION symbol, then the second pass
2807                  * will need to modify this relocation, so hang on to it.
2808                  */
2809                 if ((ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION) &&
2810                     (aplist_append(rel_alpp, rsp, AL_CNT_STRMRGREL) == NULL))
2811                         return (0);
2812         }

2814         return (1);
2815 }

2817 /*
2818  * If the output section has any SHF_MERGE|SHF_STRINGS input sections,
2819  * replace them with a single merged/compressed input section.
2820  *
2821  * entry:
2822  *      ofl - Output file descriptor
2823  *      osp - Output section descriptor
2824  *      rel_alpp, sym_alpp, - Address of 2 APlists, to be used
2825  *              for internal processing. On the initial call to
2826  *              ld_make_strmerge, these list pointers must be NULL.
2827  *              The caller is encouraged to pass the same lists back for
2828  *              successive calls to this function without freeing
2829  *              them in between calls. This causes a single pair of
2830  *              memory allocations to be reused multiple times.
2831  *
2832  * exit:
2833  *      If section merging is possible, it is done. If no errors are
```



```
2834  *      encountered, True (1) is returned. On error, S_ERROR.
2835  *
2836  *      The contents of rel_alpp and sym_alpp on exit are
2837  *      undefined. The caller can free them, or pass them back to a subsequent
2838  *      call to this routine, but should not examine their contents.
2839  */
2840 static uintptr_t
2841 ld_make_strmerge(Ofl_desc *ofl, Os_desc *osp, APlist **rel_alpp,
2842     APlist **sym_alpp)
2843 {
2844         Str_tbl         *mstrtab;       /* string table for string merge secs */
2845         Is_desc         *mstrsec;       /* Generated string merge section */
2846         Is_desc         *isp;
2847         Shdr            *mstr_shdr;
2848         Elf_Data        *mstr_data;
2849         Sym_desc        *sdp;
2850         Rel_desc        *rsp;
2851         Aliste          idx;
2852         size_t          data_size;
2853         int             st_setstring_status;
2854         size_t          stoff;

2856         /* If string table compression is disabled, there's nothing to do */
2857         if ((ofl->ofl_flags1 & FLG_OF1_NCSTTAB) != 0)
2858                 return (1);

2860         /*
2861          * Pass over the mergeable input sections, and if they haven't
2862          * all been discarded, create a string table.
2863          */
2864         mstrtab = NULL;
2865         for (APLIST_TRAVERSE(osp->os_mstrisdescs, idx, isp)) {
2866                 if (isdesc_discarded(isp))
2867                         continue;

2869                 /*
2870                  * Input sections of 0 size are dubiously valid since they do
2871                  * not even contain the NUL string.  Ignore them.
2872                  */
2873                 if (isp->is_shdr->sh_size == 0)
2874                         continue;

2876                 /*
2877                  * We have at least one non-discarded section.
2878                  * Create a string table descriptor.
2879                  */
2880                 if ((mstrtab = st_new(FLG_STNEW_COMPRESS)) == NULL)
2881                         return (S_ERROR);
2882                 break;
2883         }

2885         /* If no string table was created, we have no mergeable sections */
2886         if (mstrtab == NULL)
2887                 return (1);

2889         /*
2890          * This routine has to make 3 passes:
2891          *
2892          *      1) Examine all relocations, insert strings from relocations
2893          *              to the mergeable input sections into the string table.
2894          *      2) Modify the relocation values to be correct for the
2895          *              new merged section.
2896          *      3) Modify the symbols used by the relocations to reference
2897          *              the new section.
2898          *
2899          * These passes cannot be combined:
```

```
2900          *      - The string table code works in two passes, and all
2901          *              strings have to be loaded in pass one before the
2902          *              offset of any strings can be determined.
2903          *      - Multiple relocations reference a single symbol, so the
2904          *              symbol cannot be modified until all relocations are
2905          *              fixed.
2906          *
2907          * The number of relocations related to section merging is usually
2908          * a mere fraction of the overall active and output relocation lists,
2909          * and the number of symbols is usually a fraction of the number
2910          * of related relocations. We therefore build APlists for the
2911          * relocations and symbols in the first pass, and then use those
2912          * lists to accelerate the operation of pass 2 and 3.
2913          *
2914          * Reinitialize the lists to a completely empty state.
2915          */
2916         aplist_reset(*rel_alpp);
2917         aplist_reset(*sym_alpp);
2918
2919         /*
2920          * Pass 1:
2921          *
2922          * Every relocation related to this output section (and the input
2923          * sections that make it up) is found in either the active, or the
2924          * output relocation list, depending on whether the relocation is to
2925          * be processed by this invocation of the linker, or inserted into the
2926          * output object.
2927          *
2928          * Build lists of relocations and symbols that will need modification,
2929          * and insert the strings they reference into the mstrtab string table.
2930          */
2931         if (strmerge_pass1(ofl, osp, mstrtab, rel_alpp, sym_alpp,
2932             &ofl->ofl_actrels) == 0)
2933                 goto return_s_error;
2934         if (strmerge_pass1(ofl, osp, mstrtab, rel_alpp, sym_alpp,
2935             &ofl->ofl_outrels) == 0)
2936                 goto return_s_error;
2937
2938         /*
2939          * Get the size of the new input section. Requesting the
2940          * string table size "cooks" the table, and finalizes its contents.
2941          */
2942         data_size = st_getstrtab_sz(mstrtab);
2943
2944         /* Create a new input section to hold the merged strings */
2945         if (new_section_from_template(ofl, isp, data_size,
2946             &mstrsec, &mstr_shdr, &mstr_data) == S_ERROR)
2947                 goto return_s_error;
2948         mstrsec->is_flags |= FLG_IS_GNSTRMRG;
2949
2950         /*
2951          * Allocate a data buffer for the new input section.
2952          * Then, associate the buffer with the string table descriptor.
2953          */
2954         if ((mstr_data->d_buf = libld_malloc(data_size)) == NULL)
2955                 goto return_s_error;
2956         if (st_setstrbuf(mstrtab, mstr_data->d_buf, data_size) == -1)
2957                 goto return_s_error;
2958
2959         /* Add the new section to the output image */
2960         if (ld_place_section(ofl, mstrsec, NULL, osp->os_identndx, NULL) ==
2961             (Os_desc *)S_ERROR)
2962                 goto return_s_error;
2963
2964         /*
2965          * Pass 2:
```



```
2966          *
2967          * Revisit the relocation descriptors with STT_SECTION symbols
2968          * that were saved by the first pass. Update each relocation
2969          * record so that the offset it contains is for the new section
2970          * instead of the original.
2971          */
2972         for (APLIST_TRAVERSE(*rel_alpp, idx, rsp)) {
2973                 const char      *name;
2974
2975                 /* Put the string into the merged string table */
2976                 name = strmerge_get_reloc_str(ofl, rsp);
2977                 st_setstring_status = st_setstring(mstrtab, name, &stoff);
2978                 if (st_setstring_status == -1) {
2979                         /*
2980                          * A failure to insert at this point means that
2981                          * something is corrupt. This isn't a resource issue.
2982                          */
2983                         assert(st_setstring_status != -1);
2984                         goto return_s_error;
2985                 }
2986
2987                 /*
2988                  * Alter the relocation to access the string at the
2989                  * new offset in our new string table.
2990                  *
2991                  * For SHT_RELA platforms, it suffices to simply
2992                  * update the rel_raddend field of the relocation.
2993                  *
2994                  * For SHT_REL platforms, the new "addend" value
2995                  * needs to be written at the address being relocated.
2996                  * However, we can't alter the input sections which
2997                  * are mapped readonly, and the output image has not
2998                  * been created yet. So, we defer this operation,
2999                  * using the rel_raddend field of the relocation
3000                  * which is normally 0 on a REL platform, to pass the
3001                  * new "addend" value to ld_perform_outreloc() or
3002                  * ld_do_activerelocs(). The FLG_REL_NADDEND flag
3003                  * tells them that this is the case.
3004                  */
3005                 if ((rsp->rel_flags & FLG_REL_RELA) == 0)   /* REL */
3006                         rsp->rel_flags |= FLG_REL_NADDEND;
3007                 rsp->rel_raddend = (Sxword)stoff;
3008
3009                 /*
3010                  * Generate a symbol name string for STT_SECTION symbols
3011                  * that might reference our merged section. This shows up
3012                  * in debug output and helps show how the relocation has
3013                  * changed from its original input section to our merged one.
3014                  */
3015                 if (ld_stt_section_sym_name(mstrsec) == NULL)
3016                         goto return_s_error;
3017         }
3018
3019         /*
3020          * Pass 3:
3021          *
3022          * Modify the symbols referenced by the relocation descriptors
3023          * so that they reference the new input section containing the
3024          * merged strings instead of the original input sections.
3025          */
3026         for (APLIST_TRAVERSE(*sym_alpp, idx, sdp)) {
3027                 /*
3028                  * If we've already processed this symbol, don't do it
3029                  * twice. strmerge_pass1() uses a heuristic (relocations to
3030                  * the same symbol clump together) to avoid inserting a
3031                  * given symbol more than once, but repeat symbols in
```

```
3032                 * the list can occur.
3033                 */
3034                if ((sdp->sd_isc->is_flags & FLG_IS_INSTRMRG) == 0)
3035                        continue;

3037                if (ELF_ST_TYPE(sdp->sd_sym->st_info) != STT_SECTION) {
3038                        /*
3039                         * This is not an STT_SECTION symbol, so its
3040                         * value is the offset of the string within the
3041                         * input section. Update the address to reflect
3042                         * the address in our new merged section.
3043                         */
3044                        const char *name = sdp->sd_sym->st_value +
3045                            (char *)sdp->sd_isc->is_indata->d_buf;

3047                        st_setstring_status =
3048                            st_setstring(mstrtab, name, &stoff);
3049                        if (st_setstring_status == -1) {
3050                                /*
3051                                 * A failure to insert at this point means
3052                                 * something is corrupt. This isn't a
3053                                 * resource issue.
3054                                 */
3055                                assert(st_setstring_status != -1);
3056                                goto return_s_error;
3057                        }

3059                        if (ld_sym_copy(sdp) == S_ERROR)
3060                                goto return_s_error;
3061                        sdp->sd_sym->st_value = (Word)stoff;
3062                }

3064                /* Redirect the symbol to our new merged section */
3065                sdp->sd_isc = mstrsec;
3066        }

3068        /*
3069         * There are no references left to the original input string sections.
3070         * Mark them as discarded so they don't go into the output image.
3071         * At the same time, add up the sizes of the replaced sections.
3072         */
3073        data_size = 0;
3074        for (APLIST_TRAVERSE(osp->os_mstrisdescs, idx, isp)) {
3075                if (isp->is_flags & (FLG_IS_DISCARD | FLG_IS_GNSTRMRG))
3076                        continue;

3078                data_size += isp->is_indata->d_size;

3080                isp->is_flags |= FLG_IS_DISCARD;
3081                DBG_CALL(Dbg_sec_discarded(ofl->ofl_lml, isp, mstrsec));
3082        }

3084        /* Report how much space we saved in the output section */
3085        DBG_CALL(Dbg_sec_genstr_compress(ofl->ofl_lml, osp->os_name, data_size,
3086            mstr_data->d_size));

3088        st_destroy(mstrtab);
3089        return (1);

3091 return_s_error:
3092        st_destroy(mstrtab);
3093        return (S_ERROR);
3094 }

3096 /*
3097  * Update a data buffers size.  A number of sections have to be created, and
```



```
3098  * the sections header contributes to the size of the eventual section.  Thus,
3099  * a section may be created, and once all associated sections have been created,
3100  * we return to establish the required section size.
3101  */
3102 inline static void
3103 update_data_size(Os_desc *osp, ulong_t cnt)
3104 {
3105        Is_desc         *isec = ld_os_first_isdesc(osp);
3106        Elf_Data        *data = isec->is_indata;
3107        Shdr            *shdr = osp->os_shdr;
3108        size_t          size = cnt * shdr->sh_entsize;

3110        shdr->sh_size = (Xword)size;
3111        data->d_size = size;
3112 }

3114 /*
3115  * The following sections are built after all input file processing and symbol
3116  * validation has been carried out.  The order is important (because the
3117  * addition of a section adds a new symbol there is a chicken and egg problem
3118  * of maintaining the appropriate counts).  By maintaining a known order the
3119  * individual routines can compensate for later, known, additions.
3120  */
3121 uintptr_t
3122 ld_make_sections(Ofl_desc *ofl)
3123 {
3124        ofl_flag_t      flags = ofl->ofl_flags;
3125        Sg_desc         *sgp;

3127        /*
3128         * Generate any special sections.
3129         */
3130        if (flags & FLG_OF_ADDVERS)
3131                if (make_comment(ofl) == S_ERROR)
3132                        return (S_ERROR);

3134        if (make_interp(ofl) == S_ERROR)
3135                return (S_ERROR);

3137        /*
3138         * Create a capabilities section if required.
3139         */
3140        if (make_cap(ofl, SHT_SUNW_cap, MSG_ORIG(MSG_SCN_SUNWCAP),
3141            ld_targ.t_id.id_cap) == S_ERROR)
3142                return (S_ERROR);

3144        /*
3145         * Create any init/fini array sections.
3146         */
3147        if (make_array(ofl, SHT_INIT_ARRAY, MSG_ORIG(MSG_SCN_INITARRAY),
3148            ofl->ofl_initarray) == S_ERROR)
3149                return (S_ERROR);

3151        if (make_array(ofl, SHT_FINI_ARRAY, MSG_ORIG(MSG_SCN_FINIARRAY),
3152            ofl->ofl_finiarray) == S_ERROR)
3153                return (S_ERROR);

3155        if (make_array(ofl, SHT_PREINIT_ARRAY, MSG_ORIG(MSG_SCN_PREINITARRAY),
3156            ofl->ofl_preiarray) == S_ERROR)
3157                return (S_ERROR);

3159        /*
3160         * Make the .plt section.  This occurs after any other relocation
3161         * sections are generated (see reloc_init()) to ensure that the
3162         * associated relocation section is after all the other relocation
3163         * sections.
```

```
3164          */
3165         if ((ofl->ofl_pltcnt) || (ofl->ofl_pltpad))
3166                 if (make_plt(ofl) == S_ERROR)
3167                         return (S_ERROR);
3168
3169         /*
3170          * Determine whether any sections or files are not referenced.  Under
3171          * -Dunused a diagnostic for any unused components is generated, under
3172          * -zignore the component is removed from the final output.
3173          */
3174         if (DBG_ENABLED || (ofl->ofl_flags1 & FLG_OF1_IGNPRC)) {
3175                 if (ignore_section_processing(ofl) == S_ERROR)
3176                         return (S_ERROR);
3177         }
3178
3179         /*
3180          * If we have detected a situation in which previously placed
3181          * output sections may have been discarded, perform the necessary
3182          * readjustment.
3183          */
3184         if (ofl->ofl_flags & FLG_OF_ADJOSCNT)
3185                 adjust_os_count(ofl);
3186
3187         /*
3188          * Do any of the output sections contain input sections that
3189          * are candidates for string table merging? For each such case,
3190          * we create a replacement section, insert it, and discard the
3191          * originals.
3192          *
3193          * rel_alpp and sym_alpp are used by ld_make_strmerge()
3194          * for its internal processing. We are responsible for the
3195          * initialization and cleanup, and ld_make_strmerge() handles the rest.
3196          * This allows us to reuse a single pair of memory buffers, allocated
3197          * for this processing, for all the output sections.
3198          */
3199         if ((ofl->ofl_flags1 & FLG_OF1_NCSTTAB) == 0) {
3200                 int     error_seen = 0;
3201                 APlist  *rel_alpp = NULL;
3202                 APlist  *sym_alpp = NULL;
3203                 Aliste  idx1;
3204
3205                 for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
3206                         Os_desc *osp;
3207                         Aliste  idx2;
3208
3209                         for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp))
3210                                 if ((osp->os_mstrisdescs != NULL) &&
3211                                     (ld_make_strmerge(ofl, osp,
3212                                     &rel_alpp, &sym_alpp) ==
3213                                     S_ERROR)) {
3214                                         error_seen = 1;
3215                                         break;
3216                                 }
3217                 }
3218                 if (rel_alpp != NULL)
3219                         libld_free(rel_alpp);
3220                 if (sym_alpp != NULL)
3221                         libld_free(sym_alpp);
3222                 if (error_seen != 0)
3223                         return (S_ERROR);
3224         }
3225
3226         /*
3227          * Add any necessary versioning information.
3228          */
3229         if (!(flags & FLG_OF_NOVERSEC)) {
```



```
3230                 if ((flags & FLG_OF_VERNEED) &&
3231                     (make_verneed(ofl) == S_ERROR))
3232                         return (S_ERROR);
3233                 if ((flags & FLG_OF_VERDEF) &&
3234                     (make_verdef(ofl) == S_ERROR))
3235                         return (S_ERROR);
3236                 if ((flags & (FLG_OF_VERNEED | FLG_OF_VERDEF)) &&
3237                     ((ofl->ofl_osversym = make_sym_sec(ofl,
3238                     MSG_ORIG(MSG_SCN_SUNWVERSYM), SHT_SUNW_versym,
3239                     ld_targ.t_id.id_version)) == (Os_desc*)S_ERROR))
3240                         return (S_ERROR);
3241         }
3242
3243         /*
3244          * Create a syminfo section if necessary.
3245          */
3246         if (flags & FLG_OF_SYMINFO) {
3247                 if ((ofl->ofl_ossyminfo = make_sym_sec(ofl,
3248                     MSG_ORIG(MSG_SCN_SUNWSYMINFO), SHT_SUNW_syminfo,
3249                     ld_targ.t_id.id_syminfo)) == (Os_desc *)S_ERROR)
3250                         return (S_ERROR);
3251         }
3252
3253         if (flags & FLG_OF_COMREL) {
3254                 /*
3255                  * If -zcombreloc is enabled then all relocations (except for
3256                  * the PLT's) are coalesced into a single relocation section.
3257                  */
3258                 if (ofl->ofl_reloccnt) {
3259                         if (make_reloc(ofl, NULL) == S_ERROR)
3260                                 return (S_ERROR);
3261                 }
3262         } else {
3263                 Aliste  idx1;
3264
3265                 /*
3266                  * Create the required output relocation sections.  Note, new
3267                  * sections may be added to the section list that is being
3268                  * traversed.  These insertions can move the elements of the
3269                  * Alist such that a section descriptor is re-read.  Recursion
3270                  * is prevented by maintaining a previous section pointer and
3271                  * insuring that this pointer isn't re-examined.
3272                  */
3273                 for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
3274                         Os_desc *osp, *posp = 0;
3275                         Aliste  idx2;
3276
3277                         for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
3278                                 if ((osp != posp) && osp->os_szoutrels &&
3279                                     (osp != ofl->ofl_osplt)) {
3280                                         if (make_reloc(ofl, osp) == S_ERROR)
3281                                                 return (S_ERROR);
3282                                 }
3283                                 posp = osp;
3284                         }
3285                 }
3286
3287                 /*
3288                  * If we're not building a combined relocation section, then
3289                  * build a .rel[a] section as required.
3290                  */
3291                 if (ofl->ofl_relocrelsz) {
3292                         if (make_reloc(ofl, NULL) == S_ERROR)
3293                                 return (S_ERROR);
3294                 }
3295         }
```

```
3297         /*
3298          * The PLT relocations are always in their own section, and we try to
3299          * keep them at the end of the PLT table.  We do this to keep the hot
3300          * "data" PLT's at the head of the table nearer the .dynsym & .hash.
3301          */
3302         if (ofl->ofl_osplt && ofl->ofl_relocpltsz) {
3303                 if (make_reloc(ofl, ofl->ofl_osplt) == S_ERROR)
3304                         return (S_ERROR);
3305         }
3306
3307         /*
3308          * Finally build the symbol and section header sections.
3309          */
3310         if (flags & FLG_OF_DYNAMIC) {
3311                 if (make_dynamic(ofl) == S_ERROR)
3312                         return (S_ERROR);
3313
3314                 /*
3315                  * A number of sections aren't necessary within a relocatable
3316                  * object, even if -dy has been used.
3317                  */
3318                 if (!(flags & FLG_OF_RELOBJ)) {
3319                         if (make_hash(ofl) == S_ERROR)
3320                                 return (S_ERROR);
3321                         if (make_dynstr(ofl) == S_ERROR)
3322                                 return (S_ERROR);
3323                         if (make_dynsym(ofl) == S_ERROR)
3324                                 return (S_ERROR);
3325                         if (ld_unwind_make_hdr(ofl) == S_ERROR)
3326                                 return (S_ERROR);
3327                         if (make_dynsort(ofl) == S_ERROR)
3328                                 return (S_ERROR);
3329                 }
3330         }
3331
3332         if (!(flags & FLG_OF_STRIP) || (flags & FLG_OF_RELOBJ) ||
3333             ((flags & FLG_OF_STATIC) && ofl->ofl_osversym)) {
3334                 /*
3335                  * Do we need to make a SHT_SYMTAB_SHNDX section
3336                  * for the dynsym.  If so - do it now.
3337                  */
3338                 if (ofl->ofl_osdynsym &&
3339                     ((ofl->ofl_shdrcnt + 3) >= SHN_LORESERVE)) {
3340                         if (make_dynsym_shndx(ofl) == S_ERROR)
3341                                 return (S_ERROR);
3342                 }
3343
3344                 if (make_strtab(ofl) == S_ERROR)
3345                         return (S_ERROR);
3346                 if (make_symtab(ofl) == S_ERROR)
3347                         return (S_ERROR);
3348         } else {
3349                 /*
3350                  * Do we need to make a SHT_SYMTAB_SHNDX section
3351                  * for the dynsym.  If so - do it now.
3352                  */
3353                 if (ofl->ofl_osdynsym &&
3354                     ((ofl->ofl_shdrcnt + 1) >= SHN_LORESERVE)) {
3355                         if (make_dynsym_shndx(ofl) == S_ERROR)
3356                                 return (S_ERROR);
3357                 }
3358         }
3359
3360         if (make_shstrtab(ofl) == S_ERROR)
3361                 return (S_ERROR);
```



```
3363         /*
3364          * Now that we've created all output sections, adjust the size of the
3365          * SHT_SUNW_versym and SHT_SUNW_syminfo section, which are dependent on
3366          * the associated symbol table sizes.
3367          */
3368         if (ofl->ofl_osversym || ofl->ofl_ossyminfo) {
3369                 ulong_t         cnt;
3370                 Is_desc         *isp;
3371                 Os_desc         *osp;
3372
3373                 if (OFL_IS_STATIC_OBJ(ofl))
3374                         osp = ofl->ofl_ossymtab;
3375                 else
3376                         osp = ofl->ofl_osdynsym;
3377
3378                 isp = ld_os_first_isdesc(osp);
3379                 cnt = (isp->is_shdr->sh_size / isp->is_shdr->sh_entsize);
3380
3381                 if (ofl->ofl_osversym)
3382                         update_data_size(ofl->ofl_osversym, cnt);
3383
3384                 if (ofl->ofl_ossyminfo)
3385                         update_data_size(ofl->ofl_ossyminfo, cnt);
3386         }
3387
3388         /*
3389          * Now that we've created all output sections, adjust the size of the
3390          * SHT_SUNW_capinfo, which is dependent on the associated symbol table
3391          * size.
3392          */
3393         if (ofl->ofl_oscapinfo) {
3394                 ulong_t cnt;
3395
3396                 /*
3397                  * Symbol capabilities symbols are placed directly after the
3398                  * STT_FILE symbol, section symbols, and any register symbols.
3399                  * Effectively these are the first of any series of demoted
3400                  * (scoped) symbols.
3401                  */
3402                 if (OFL_IS_STATIC_OBJ(ofl))
3403                         cnt = SYMTAB_ALL_CNT(ofl);
3404                 else
3405                         cnt = DYNSYM_ALL_CNT(ofl);
3406
3407                 update_data_size(ofl->ofl_oscapinfo, cnt);
3408         }
3409         return (1);
3410 }
3411
3412 /*
3413  * Build an additional data section - used to back OBJT symbol definitions
3414  * added with a mapfile.
3415  */
3416 Is_desc *
3417 ld_make_data(Ofl_desc *ofl, size_t size)
3418 {
3419         Shdr            *shdr;
3420         Elf_Data        *data;
3421         Is_desc         *isec;
3422
3423         if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_DATA), 0,
3424             &isec, &shdr, &data) == S_ERROR)
3425                 return ((Is_desc *)S_ERROR);
3426
3427         data->d_size = size;
```

```
3428         shdr->sh_size = (Xword)size;
3429         shdr->sh_flags |= SHF_WRITE;

3431         if (aplist_append(&ofl->ofl_mapdata, isec, AL_CNT_OFL_MAPSECS) == NULL)
3432                 return ((Is_desc *)S_ERROR);

3434         return (isec);
3435 }

3437 /*
3438  * Build an additional text section - used to back FUNC symbol definitions
3439  * added with a mapfile.
3440  */
3441 Is_desc *
3442 ld_make_text(Ofl_desc *ofl, size_t size)
3443 {
3444         Shdr            *shdr;
3445         Elf_Data        *data;
3446         Is_desc         *isec;

3448         /*
3449          * Insure the size is sufficient to contain the minimum return
3450          * instruction.
3451          */
3452         if (size < ld_targ.t_nf.nf_size)
3453                 size = ld_targ.t_nf.nf_size;

3455         if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_TEXT), 0,
3456             &isec, &shdr, &data) == S_ERROR)
3457                 return ((Is_desc *)S_ERROR);

3459         data->d_size = size;
3460         shdr->sh_size = (Xword)size;
3461         shdr->sh_flags |= SHF_EXECINSTR;

3463         /*
3464          * Fill the buffer with the appropriate return instruction.
3465          * Note that there is no need to swap bytes on a non-native,
3466          * link, as the data being copied is given in bytes.
3467          */
3468         if ((data->d_buf = libld_calloc(size, 1)) == NULL)
3469                 return ((Is_desc *)S_ERROR);
3470         (void) memcpy(data->d_buf, ld_targ.t_nf.nf_template,
3471             ld_targ.t_nf.nf_size);

3473         /*
3474          * If size was larger than required, and the target supplies
3475          * a fill function, use it to fill the balance. If there is no
3476          * fill function, we accept the 0-fill supplied by libld_calloc().
3477          */
3478         if ((ld_targ.t_ff.ff_execfill != NULL) && (size > ld_targ.t_nf.nf_size))
3479                 ld_targ.t_ff.ff_execfill(data->d_buf, ld_targ.t_nf.nf_size,
3480                     size - ld_targ.t_nf.nf_size);

3482         if (aplist_append(&ofl->ofl_maptext, isec, AL_CNT_OFL_MAPSECS) == NULL)
3483                 return ((Is_desc *)S_ERROR);

3485         return (isec);
3486 }

3488 void
3489 ld_comdat_validate(Ofl_desc *ofl, Ifl_desc *ifl)
3490 {
3491         int i;

3493         for (i = 0; i < ifl->ifl_shnum; i++) {
```



```
3494                 Is_desc *isp = ifl->ifl_isdesc[i];
3495                 int types = 0;
3496                 char buf[1024] = "";
3497                 Group_desc *gr = NULL;

3499                 if ((isp == NULL) || (isp->is_flags & FLG_IS_COMDAT) == 0)
3500                         continue;

3502                 if (isp->is_shdr->sh_type == SHT_SUNW_COMDAT) {
3503                         types++;
3504                         (void) strlcpy(buf, MSG_ORIG(MSG_STR_SUNW_COMDAT),
3505                             sizeof (buf));
3506                 }

3508                 if (strncmp(MSG_ORIG(MSG_SCN_GNU_LINKONCE), isp->is_name,
3509                     MSG_SCN_GNU_LINKONCE_SIZE) == 0) {
3510                         types++;
3511                         if (types > 1)
3512                                 (void) strlcat(buf, ", ", sizeof (buf));
3513                         (void) strlcat(buf, MSG_ORIG(MSG_SCN_GNU_LINKONCE),
3514                             sizeof (buf));
3515                 }

3517                 if ((isp->is_shdr->sh_flags & SHF_GROUP) &&
3518                     ((gr = ld_get_group(ofl, isp)) != NULL) &&
3519                     (gr->gd_data[0] & GRP_COMDAT)) {
3520                         types++;
3521                         if (types > 1)
3522                                 (void) strlcat(buf, ", ", sizeof (buf));
3523                         (void) strlcat(buf, MSG_ORIG(MSG_STR_GROUP),
3524                             sizeof (buf));
3525                 }

3527                 if (types > 1)
3528                         ld_eprintf(ofl, ERR_FATAL,
3529                             MSG_INTL(MSG_SCN_MULTICOMDAT), ifl->ifl_name,
3530                             EC_WORD(isp->is_scnndx), isp->is_name, buf);
3531         }
3532 }
```

```
**********************************************************
   97619 Sun Feb 24 19:19:13 2019
new/usr/src/cmd/sgs/libld/common/syms.c
fill out symbol-bound symbols in kmods
linker_set sections shouldn't need leading '.'
code review
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_
```

```
610 /*
611  * Add a special symbol to the symbol table.  Takes special symbol name with
612  * and without underscores.  This routine is called, after all other symbol
613  * resolution has completed, to generate a reserved absolute symbol (the
614  * underscore version).  Special symbols are updated with the appropriate
615  * values in update_osym().  If the user has already defined this symbol
616  * issue a warning and leave the symbol as is.  If the non-underscore symbol
617  * is referenced then turn it into a weak alias of the underscored symbol.
618  *
619  * The bits in sdflags_u are OR'd into the flags field of the symbol for the
620  * underscored symbol.
621  *
622  * If this is a global symbol, and it hasn't explicitly been defined as being
623  * directly bound to, indicate that it can't be directly bound to.
624  * Historically, most special symbols only have meaning to the object in which
625  * they exist, however, they've always been global.  To ensure compatibility
626  * with any unexpected use presently in effect, ensure these symbols don't get
627  * directly bound to.  Note, that establishing this state here isn't sufficient
628  * to create a syminfo table, only if a syminfo table is being created by some
629  * other symbol directives will the nodirect binding be recorded.  This ensures
630  * we don't create syminfo sections for all objects we create, as this might add
631  * unnecessary bloat to users who haven't explicitly requested extra symbol
632  * information.
633  */
634 static uintptr_t
635 sym_add_spec(const char *name, const char *uname, Word sdaux_id,
636     sd_flag_t sdflags_u, sd_flag_t sdflags, Ofl_desc *ofl)
637 {
638         Sym_desc        *sdp;
639         Sym_desc        *usdp;
640         Sym             *sym;
641         Word            hash;
642         avl_index_t     where;

644         /* LINTED */
645         hash = (Word)elf_hash(uname);
646         if (usdp = ld_sym_find(uname, hash, &where, ofl)) {
647                 /*
648                  * If the underscore symbol exists and is undefined, or was
649                  * defined in a shared library, convert it to a local symbol.
650                  * Otherwise leave it as is and warn the user.
651                  */
652                 if ((usdp->sd_shndx == SHN_UNDEF) ||
653                     (usdp->sd_ref != REF_REL_NEED)) {
654                         usdp->sd_ref = REF_REL_NEED;
655                         usdp->sd_shndx = usdp->sd_sym->st_shndx = SHN_ABS;
656                         usdp->sd_flags |= FLG_SY_SPECSEC | sdflags_u;
657                         usdp->sd_sym->st_info =
658                             ELF_ST_INFO(STB_GLOBAL, STT_OBJECT);
659                         usdp->sd_isc = NULL;
660                         usdp->sd_sym->st_size = 0;
661                         usdp->sd_sym->st_value = 0;
662                         /* LINTED */
663                         usdp->sd_aux->sa_symspec = (Half)sdaux_id;
```



```
665                         /*
666                          * If a user hasn't specifically indicated that the
667                          * scope of this symbol be made local, then leave it
668                          * as global (ie. prevent automatic scoping).  The GOT
669                          * should be defined protected, whereas all other
670                          * special symbols are tagged as no-direct.
671                          */
672                         if (!SYM_IS_HIDDEN(usdp) &&
673                             (sdflags & FLG_SY_DEFAULT)) {
674                                 usdp->sd_aux->sa_overndx = VER_NDX_GLOBAL;
675                                 if (sdaux_id == SDAUX_ID_GOT) {
676                                         usdp->sd_flags &= ~FLG_SY_NDIR;
677                                         usdp->sd_flags |= FLG_SY_PROTECT;
678                                         usdp->sd_sym->st_other = STV_PROTECTED;
679                                 } else if (
680                                     ((usdp->sd_flags & FLG_SY_DIR) == 0) &&
681                                     ((ofl->ofl_flags & FLG_OF_SYMBOLIC) == 0)) {
682                                         usdp->sd_flags |= FLG_SY_NDIR;
683                                 }
684                         }
685                         usdp->sd_flags |= sdflags;

687                         /*
688                          * If the reference originated from a mapfile ensure
689                          * we mark the symbol as used.
690                          */
691                         if (usdp->sd_flags & FLG_SY_MAPREF)
692                                 usdp->sd_flags |= FLG_SY_MAPUSED;

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

719                 usdp->sd_aux->sa_overndx = VER_NDX_GLOBAL;

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

```
729         }
730
731         if (name && (sdp = ld_sym_find(name, SYM_NOHASH, NULL, ofl)) &&
732             (sdp->sd_sym->st_shndx == SHN_UNDEF)) {
733                 uchar_t bind;
734
735                 /*
736                  * If the non-underscore symbol exists and is undefined
737                  * convert it to be a local.  If the underscore has
738                  * sa_symspec set (ie. it was created above) then simulate this
739                  * as a weak alias.
740                  */
741                 sdp->sd_ref = REF_REL_NEED;
742                 sdp->sd_shndx = sdp->sd_sym->st_shndx = SHN_ABS;
743                 sdp->sd_flags |= FLG_SY_SPECSEC;
744                 sdp->sd_isc = NULL;
745                 sdp->sd_sym->st_size = 0;
746                 sdp->sd_sym->st_value = 0;
747                 /* LINTED */
748                 sdp->sd_aux->sa_symspec = (Half)sdaux_id;
749                 if (usdp->sd_aux->sa_symspec) {
750                         usdp->sd_aux->sa_linkndx = 0;
751                         sdp->sd_aux->sa_linkndx = 0;
752                         bind = STB_WEAK;
753                 } else
754                         bind = STB_GLOBAL;
755                 sdp->sd_sym->st_info = ELF_ST_INFO(bind, STT_OBJECT);
756
757                 /*
758                  * If a user hasn't specifically indicated the scope of this
759                  * symbol be made local then leave it as global (ie. prevent
760                  * automatic scoping).  The GOT should be defined protected,
761                  * whereas all other special symbols are tagged as no-direct.
762                  */
763                 if (!SYM_IS_HIDDEN(sdp) &&
764                     (sdflags & FLG_SY_DEFAULT)) {
765                         sdp->sd_aux->sa_overndx = VER_NDX_GLOBAL;
766                         if (sdaux_id == SDAUX_ID_GOT) {
767                                 sdp->sd_flags &= ~FLG_SY_NDIR;
768                                 sdp->sd_flags |= FLG_SY_PROTECT;
769                                 sdp->sd_sym->st_other = STV_PROTECTED;
770                         } else if (((sdp->sd_flags & FLG_SY_DIR) == 0) &&
771                             ((ofl->ofl_flags & FLG_OF_SYMBOLIC) == 0)) {
772                                 sdp->sd_flags |= FLG_SY_NDIR;
773                         }
774                 }
775                 sdp->sd_flags |= sdflags;
776
777                 /*
778                  * If the reference originated from a mapfile ensure
779                  * we mark the symbol as used.
780                  */
781                 if (sdp->sd_flags & FLG_SY_MAPREF)
782                         sdp->sd_flags |= FLG_SY_MAPUSED;
783
784                 DBG_CALL(Dbg_syms_updated(ofl, sdp, name));
785         }
786         return (1);
787 }
788
789
790 /*
791  * Undefined symbols can fall into one of four types:
792  *
793  *  -   the symbol is really undefined (SHN_UNDEF).
794  *
```



```
795  *  -   versioning has been enabled, however this symbol has not been assigned
796  *      to one of the defined versions.
797  *
798  *  -   the symbol has been defined by an implicitly supplied library, ie. one
799  *      which was encounted because it was NEEDED by another library, rather
800  *      than from a command line supplied library which would become the only
801  *      dependency of the output file being produced.
802  *
803  *  -   the symbol has been defined by a version of a shared object that is
804  *      not permitted for this link-edit.
805  *
806  * In all cases the file who made the first reference to this symbol will have
807  * been recorded via the 'sa_rfile' pointer.
808  */
809 typedef enum {
810         UNDEF,          NOVERSION,      IMPLICIT,       NOTAVAIL,
811         BNDLOCAL
812 } Type;
813
814 static const Msg format[] = {
815         MSG_SYM_UND_UNDEF,              /* MSG_INTL(MSG_SYM_UND_UNDEF) */
816         MSG_SYM_UND_NOVER,              /* MSG_INTL(MSG_SYM_UND_NOVER) */
817         MSG_SYM_UND_IMPL,               /* MSG_INTL(MSG_SYM_UND_IMPL) */
818         MSG_SYM_UND_NOTA,               /* MSG_INTL(MSG_SYM_UND_NOTA) */
819         MSG_SYM_UND_BNDLOCAL            /* MSG_INTL(MSG_SYM_UND_BNDLOCAL) */
820 };
821
822 /*
823  * Issue an undefined symbol message for the given symbol.
824  *
825  * entry:
826  *      ofl - Output descriptor
827  *      sdp - Undefined symbol to report
828  *      type - Type of undefined symbol
829  *      ofl_flag - One of 0, FLG_OF_FATAL, or FLG_OF_WARN.
830  *      undef_state - Address of variable to be initialized to 0
831  *              before the first call to sym_undef_entry, and passed
832  *              to each subsequent call. A non-zero value for *undef_state
833  *              indicates that this is not the first call in the series.
834  *
835  * exit:
836  *      If *undef_state is 0, a title is issued.
837  *
838  *      A message for the undefined symbol is issued.
839  *
840  *      If ofl_flag is non-zero, its value is OR'd into *undef_state. Otherwise,
841  *      all bits other than FLG_OF_FATAL and FLG_OF_WARN are set, in order to
842  *      provide *undef_state with a non-zero value. These other bits have
843  *      no meaning beyond that, and serve to ensure that *undef_state is
844  *      non-zero if sym_undef_entry() has been called.
845  */
846 static void
847 sym_undef_entry(Ofl_desc *ofl, Sym_desc *sdp, Type type, ofl_flag_t ofl_flag,
848     ofl_flag_t *undef_state)
849 {
850         const char      *name1, *name2, *name3;
851         Ifl_desc        *ifl = sdp->sd_file;
852         Sym_aux         *sap = sdp->sd_aux;
853
854         if (*undef_state == 0)
855                 ld_eprintf(ofl, ERR_NONE, MSG_INTL(MSG_SYM_FMT_UNDEF),
856                     MSG_INTL(MSG_SYM_UNDEF_ITM_11),
857                     MSG_INTL(MSG_SYM_UNDEF_ITM_21),
858                     MSG_INTL(MSG_SYM_UNDEF_ITM_12),
859                     MSG_INTL(MSG_SYM_UNDEF_ITM_22));
```

```
861         ofl->ofl_flags |= ofl_flag;
862         *undef_state |= ofl_flag ? ofl_flag : ~(FLG_OF_FATAL | FLG_OF_WARN);

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

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

889 /*
890  * If an undef symbol exists naming a bound for the output section,
891  * turn it into a defined symbol with the correct value.
892  *
893  * We set an arbitrary 1KB limit on the resulting symbol names.
894  */
895 static void
896 sym_add_bounds(Ofl_desc *ofl, Os_desc *osp, Word bound)
897 {
898         Sym_desc *bsdp;
899         char symn[1024];
900         size_t nsz;

902         switch (bound) {
903         case SDAUX_ID_SECBOUND_START:
904                 nsz = snprintf(symn, sizeof (symn), "%s%s",
905                     MSG_ORIG(MSG_SYM_SECBOUND_START), osp->os_name);
906                 if (nsz >= sizeof (symn))
907                         return;
908                 break;
909         case SDAUX_ID_SECBOUND_STOP:
910                 nsz = snprintf(symn, sizeof (symn), "%s%s",
911                     MSG_ORIG(MSG_SYM_SECBOUND_STOP), osp->os_name);
912                 if (nsz >= sizeof (symn))
913                         return;
914                 break;
915         default:
916                 assert(0);
917         }

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



```
927                 DBG_CALL(Dbg_syms_updated(ofl, bsdp, symn));

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

942 static Boolean
943 is_cname(const char *name)
944 {
945         if (strlen(name) == strspn(name,
946             "abcdefghijklmnopqrstuvwxyz"
947             "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
948             "0123456789"
949             "_"))
950                 return (TRUE);
951         else
952                 return (FALSE);
953 }

955 /*
956 #endif /* ! codereview */
957  * At this point all symbol input processing has been completed, therefore
958  * complete the symbol table entries by generating any necessary internal
959  * symbols.
960  */
961 uintptr_t
962 ld_sym_spec(Ofl_desc *ofl)
963 {
964         Sym_desc        *sdp;
965         Sg_desc         *sgp;
966         Aliste          idx1;

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

970         /*
971          * For each section in the output file, look for symbols named for the
972          * __start/__stop patterns.  If references exist, flesh the symbols to
973          * be defined.
974          *
975          * The symbols are given values at the same time as the other special
976          * symbols.
977          */
978         if (!(ofl->ofl_flags & FLG_OF_RELOBJ) ||
979             (ofl->ofl_flags & FLG_OF_KMOD)) {
980                 for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
981                         Os_desc *osp;
982                         Aliste idx2;

984                         for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
985                                 if (is_cname(osp->os_name)) {
986                                         sym_add_bounds(ofl, osp,
987                                             SDAUX_ID_SECBOUND_START);
988                                         sym_add_bounds(ofl, osp,
989                                             SDAUX_ID_SECBOUND_STOP);
990                                 }
991                         }
992                 }
```

```
 993         }
 994 #endif /* ! codereview */

 996         if (ofl->ofl_flags & FLG_OF_RELOBJ)
 997                 return (1);

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

 999         if (sym_add_spec(MSG_ORIG(MSG_SYM_ETEXT), MSG_ORIG(MSG_SYM_ETEXT_U),
1000             SDAUX_ID_ETEXT, 0, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
1001             ofl) == S_ERROR)
1002                 return (S_ERROR);
1003         if (sym_add_spec(MSG_ORIG(MSG_SYM_EDATA), MSG_ORIG(MSG_SYM_EDATA_U),
1004             SDAUX_ID_EDATA, 0, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
1005             ofl) == S_ERROR)
1006                 return (S_ERROR);
1007         if (sym_add_spec(MSG_ORIG(MSG_SYM_END), MSG_ORIG(MSG_SYM_END_U),
1008             SDAUX_ID_END, FLG_SY_DYNSORT, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
1009             ofl) == S_ERROR)
1010                 return (S_ERROR);
1011         if (sym_add_spec(MSG_ORIG(MSG_SYM_L_END), MSG_ORIG(MSG_SYM_L_END_U),
1012             SDAUX_ID_END, 0, FLG_SY_HIDDEN, ofl) == S_ERROR)
1013                 return (S_ERROR);
1014         if (sym_add_spec(MSG_ORIG(MSG_SYM_L_START), MSG_ORIG(MSG_SYM_L_START_U),
1015             SDAUX_ID_START, 0, FLG_SY_HIDDEN, ofl) == S_ERROR)
1016                 return (S_ERROR);

1018         /*
1019          * Historically we've always produced a _DYNAMIC symbol, even for
1020          * static executables (in which case its value will be 0).
1021          */
1022         if (sym_add_spec(MSG_ORIG(MSG_SYM_DYNAMIC), MSG_ORIG(MSG_SYM_DYNAMIC_U),
1023             SDAUX_ID_DYN, FLG_SY_DYNSORT, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
1024             ofl) == S_ERROR)
1025                 return (S_ERROR);

1027         if (OFL_ALLOW_DYNSYM(ofl))
1028                 if (sym_add_spec(MSG_ORIG(MSG_SYM_PLKTBL),
1029                     MSG_ORIG(MSG_SYM_PLKTBL_U), SDAUX_ID_PLT,
1030                     FLG_SY_DYNSORT, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
1031                     ofl) == S_ERROR)
1032                         return (S_ERROR);

1034         /*
1035          * A GOT reference will be accompanied by the associated GOT symbol.
1036          * Make sure it gets assigned the appropriate special attributes.
1037          */
1038         if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_GOFTBL_U),
1039             SYM_NOHASH, NULL, ofl)) != NULL) && (sdp->sd_ref != REF_DYN_SEEN)) {
1040                 if (sym_add_spec(MSG_ORIG(MSG_SYM_GOFTBL),
1041                     MSG_ORIG(MSG_SYM_GOFTBL_U), SDAUX_ID_GOT, FLG_SY_DYNSORT,
1042                     (FLG_SY_DEFAULT | FLG_SY_EXPDEF), ofl) == S_ERROR)
1043                         return (S_ERROR);
1044         }

1046         return (1);
1047 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  118868 Sun Feb 24 19:19:13 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)
ld: implement -ztype and rework option parsing
**********************************************************
______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;
258
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                 }
276
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         }
292
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;
301
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;
313
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;
322
323         /*
324          * Setup our string tables.
325          */
326         shstrtab = ofl->ofl_shdrsttab;
327         strtab = ofl->ofl_strtab;
328         dynstr = ofl->ofl_dynstrtab;
329
330         DBG_CALL(Dbg_syms_sec_title(ofl->ofl_lml));
331
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;
345
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;
359
360                 /* Scoped symbols get filled in global loop below */
361                 ldynscopesym_ndx = ldynsym_ndx + 1;
362                 ldynsym_ndx += ofl->ofl_dynscopecnt;
363         }
364
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         }
374
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;
384
385                 if (phd->p_type == PT_LOAD) {
```

```
386                         if (sgp->sg_osdescs != NULL) {
387                                 Word    _flags = phd->p_flags & (PF_W | PF_R);
388
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                 }
396
397                 /*
398                  * Generate a section symbol for each output section.
399                  */
400                 for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
401                         Word    sectndx;
402
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);
408
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                         }
419
420                         if (dynsym && (osp->os_flags & FLG_OS_OUTREL))
421                                 dynsym[dynsym_ndx++] = *sym;
422
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                         }
431
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;
437
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                         }
450
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                         }
457
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         }
470
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;
477
478                 for (ndx = 0; ndx < ofl->ofl_regsymsno; ndx++) {
479                         Sym_desc        *rsdp;
480
481                         if ((rsdp = ofl->ofl_regsyms[ndx]) == NULL)
482                                 continue;
483
484                         if (!SYM_IS_HIDDEN(rsdp) &&
485                             (ELF_ST_BIND(rsdp->sd_sym->st_info) != STB_LOCAL))
486                                 continue;
487
488                         dynsym[dynsym_ndx] = *(rsdp->sd_sym);
489                         rsdp->sd_symndx = *symndx;
490
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         }
499
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;
508
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                         }
792
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;
799
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                         }
828
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;
841
842                                 if (_symshndx && ldynshndx)
843                                         ldynshndx[ldynsym_ndx] = *_symshndx;
844
845                                 /* Add it to sort section if it qualifies */
846                                 ADD_TO_DYNSORT(sdp, sym, type, ldynsym_ndx);
```



```
848                                         ldynsym_ndx++;
849                                 }
850                         }
851
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;
862
863                                 for (APLIST_TRAVERSE(cdp->ca_syms, idx2, csp)) {
864                                         Is_desc *isp;
865
866                                         sdp = csp->cs_sdp;
867                                         sym = sdp->sd_sym;
868
869                                         if ((isp = sdp->sd_isc) != NULL) {
870                                                 Os_desc *osp = isp->is_osdesc;
871
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;
881
882                                                 /*
883                                                  * Update the symbols section index.
884                                                  */
885                                                 sdp->sd_shndx = sym->st_shndx =
886                                                     elf_ndxscn(osp->os_scn);
887                                         }
888
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                 }
897
898                 symtab_gbl_bndx = symtab_ndx;   /* .symtab index of 1st global entry */
899
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                }

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                }

1805                /*
1806                 * Finish updating the symbols.
1807                 */

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));

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;

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                }

1832                if (enter_in_symtab) {
1833                        Word    _symndx;

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            }
2236
2237            dyn->d_tag = DT_HASH;
2238            dyn->d_un.d_ptr = ofl->ofl_oshash->os_shdr->sh_addr;
2239            dyn++;
2240
2241            shdr = strosp->os_shdr;
2242            dyn->d_tag = DT_STRTAB;
2243            dyn->d_un.d_ptr = shdr->sh_addr;
2244            dyn++;
2245
2246            dyn->d_tag = DT_STRSZ;
2247            dyn->d_un.d_ptr = shdr->sh_size;
2248            dyn++;
2249
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++;
2258
2259            dyn->d_tag = DT_SYMENT;
2260            dyn->d_un.d_ptr = shdr->sh_entsize;
2261            dyn++;
2262
2263            if (ofl->ofl_osldynsym) {
2264                    Shdr    *lshdr = ofl->ofl_osldynsym->os_shdr;
2265
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++;
2278
2279                    dyn->d_tag = DT_SUNW_SYMSZ;
2280                    dyn->d_un.d_val = lshdr->sh_size + shdr->sh_size;
2281                    dyn++;
2282            }
2283
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            }
2289
2290            if (ofl->ofl_osdynsymsort) {
2291                    shdr = ofl->ofl_osdynsymsort->os_shdr;
2292
2293                    dyn->d_tag = DT_SUNW_SYMSORT;
2294                    dyn->d_un.d_ptr = shdr->sh_addr;
2295                    dyn++;
2296
2297                    dyn->d_tag = DT_SUNW_SYMSORTSZ;
2298                    dyn->d_un.d_val = shdr->sh_size;
2299                    dyn++;
```



```
2300            }
2301
2302            if (ofl->ofl_osdyntlssort) {
2303                    shdr = ofl->ofl_osdyntlssort->os_shdr;
2304
2305                    dyn->d_tag = DT_SUNW_TLSSORT;
2306                    dyn->d_un.d_ptr = shdr->sh_addr;
2307                    dyn++;
2308
2309                    dyn->d_tag = DT_SUNW_TLSSORTSZ;
2310                    dyn->d_un.d_val = shdr->sh_size;
2311                    dyn++;
2312            }
2313
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++;
2321
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;
2358
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    if (ofl->ofl_flags & FLG_OF_KMOD) {
2597            dyn->d_tag = DT_SUNW_KMOD;
2598            dyn->d_un.d_val = 1;
2599            dyn++;
2600    }

2602 #endif /* ! codereview */
2603    (*ld_targ.t_mr.mr_mach_update_odynamic)(ofl, &dyn);

2605    for (cnt = 1 + DYNAMIC_EXTRA_ELTS; cnt--; dyn++) {
2606            dyn->d_tag = DT_NULL;
2607            dyn->d_un.d_val = 0;
2608    }

2610    /*
2611     * Ensure that we wrote the right number of entries. If not, we either
2612     * miscounted in make_dynamic(), or we did something wrong in this
2613     * function.
2614     */
2615    assert((ofl->ofl_osdynamic->os_shdr->sh_size /
2616        ofl->ofl_osdynamic->os_shdr->sh_entsize) ==
2617        ((uintptr_t)dyn - (uintptr_t)_dyn) / sizeof (*dyn));

2619    return (1);
2620 }

2622 /*
2623  * Build the version definition section
2624  */
2625 static int
2626 update_overdef(Ofl_desc *ofl)
2627 {
2628    Aliste          idx1;
2629    Ver_desc        *vdp, *_vdp;
```

```
2630         Verdef          *vdf, *_vdf;
2631         int             num = 0;
2632         Os_desc         *strosp;
2633         Str_tbl         *strtbl;

2635         /*
2636          * Determine which string table to use.
2637          */
2638         if (OFL_IS_STATIC_OBJ(ofl)) {
2639                 strtbl = ofl->ofl_strtab;
2640                 strosp = ofl->ofl_osstrtab;
2641         } else {
2642                 strtbl = ofl->ofl_dynstrtab;
2643                 strosp = ofl->ofl_osdynstr;
2644         }

2646         /*
2647          * Traverse the version descriptors and update the version structures
2648          * to point to the dynstr name in preparation for building the version
2649          * section structure.
2650          */
2651         for (APLIST_TRAVERSE(ofl->ofl_verdesc, idx1, vdp)) {
2652                 Sym_desc        *sdp;

2654                 if (vdp->vd_flags & VER_FLG_BASE) {
2655                         const char      *name = vdp->vd_name;
2656                         size_t          stoff;

2658                         /*
2659                          * Create a new string table entry to represent the base
2660                          * version name (there is no corresponding symbol for
2661                          * this).
2662                          */
2663                         (void) st_setstring(strtbl, name, &stoff);
2664                         /* LINTED */
2665                         vdp->vd_name = (const char *)stoff;
2666                 } else {
2667                         sdp = ld_sym_find(vdp->vd_name, vdp->vd_hash, 0, ofl);
2668                         /* LINTED */
2669                         vdp->vd_name = (const char *)
2670                             (uintptr_t)sdp->sd_sym->st_name;
2671                 }
2672         }

2674         _vdf = vdf = (Verdef *)ofl->ofl_osverdef->os_outdata->d_buf;

2676         /*
2677          * Traverse the version descriptors and update the version section to
2678          * reflect each version and its associated dependencies.
2679          */
2680         for (APLIST_TRAVERSE(ofl->ofl_verdesc, idx1, vdp)) {
2681                 Aliste          idx2;
2682                 Half            cnt = 1;
2683                 Verdaux         *vdap, *_vdap;

2685                 _vdap = vdap = (Verdaux *)(vdf + 1);

2687                 vdf->vd_version = VER_DEF_CURRENT;
2688                 vdf->vd_flags   = vdp->vd_flags & MSK_VER_USER;
2689                 vdf->vd_ndx     = vdp->vd_ndx;
2690                 vdf->vd_hash    = vdp->vd_hash;

2692                 /* LINTED */
2693                 vdap->vda_name = (uintptr_t)vdp->vd_name;
2694                 vdap++;
2695                 /* LINTED */
```



```
2696                 _vdap->vda_next = (Word)((uintptr_t)vdap - (uintptr_t)_vdap);

2698                 /*
2699                  * Traverse this versions dependency list generating the
2700                  * appropriate version dependency entries.
2701                  */
2702                 for (APLIST_TRAVERSE(vdp->vd_deps, idx2, _vdp)) {
2703                         /* LINTED */
2704                         vdap->vda_name = (uintptr_t)_vdp->vd_name;
2705                         _vdap = vdap;
2706                         vdap++, cnt++;
2707                         /* LINTED */
2708                         _vdap->vda_next = (Word)((uintptr_t)vdap -
2709                             (uintptr_t)_vdap);
2710                 }
2711                 _vdap->vda_next = 0;

2713                 /*
2714                  * Record the versions auxiliary array offset and the associated
2715                  * dependency count.
2716                  */
2717                 /* LINTED */
2718                 vdf->vd_aux = (Word)((uintptr_t)(vdf + 1) - (uintptr_t)vdf);
2719                 vdf->vd_cnt = cnt;

2721                 /*
2722                  * Record the next versions offset and update the version
2723                  * pointer.  Remember the previous version offset as the very
2724                  * last structures next pointer should be null.
2725                  */
2726                 _vdf = vdf;
2727                 vdf = (Verdef *)vdap, num++;
2728                 /* LINTED */
2729                 _vdf->vd_next = (Word)((uintptr_t)vdf - (uintptr_t)_vdf);
2730         }
2731         _vdf->vd_next = 0;

2733         /*
2734          * Record the string table association with the version definition
2735          * section, and the symbol table associated with the version symbol
2736          * table (the actual contents of the version symbol table are filled
2737          * in during symbol update).
2738          */
2739         /* LINTED */
2740         ofl->ofl_osverdef->os_shdr->sh_link = (Word)elf_ndxscn(strosp->os_scn);

2742         /*
2743          * The version definition sections ‘info’ field is used to indicate the
2744          * number of entries in this section.
2745          */
2746         ofl->ofl_osverdef->os_shdr->sh_info = num;

2748         return (1);
2749 }

2751 /*
2752  * Finish the version symbol index section
2753  */
2754 static void
2755 update_oversym(Ofl_desc *ofl)
2756 {
2757         Os_desc         *osp;

2759         /*
2760          * Record the symbol table associated with the version symbol table.
2761          * The contents of the version symbol table are filled in during
```

```
2762          * symbol update.
2763          */
2764         if (OFL_IS_STATIC_OBJ(ofl))
2765                 osp = ofl->ofl_ossymtab;
2766         else
2767                 osp = ofl->ofl_osdynsym;

2769         /* LINTED */
2770         ofl->ofl_osversym->os_shdr->sh_link = (Word)elf_ndxscn(osp->os_scn);
2771 }

2773 /*
2774  * Build the version needed section
2775  */
2776 static int
2777 update_overneed(Ofl_desc *ofl)
2778 {
2779         Aliste          idx1;
2780         Ifl_desc        *ifl;
2781         Verneed         *vnd, *_vnd;
2782         Os_desc         *strosp;
2783         Str_tbl         *strtbl;
2784         Word            num = 0;

2786         _vnd = vnd = (Verneed *)ofl->ofl_osverneed->os_outdata->d_buf;

2788         /*
2789          * Determine which string table is appropriate.
2790          */
2791         if (OFL_IS_STATIC_OBJ(ofl)) {
2792                 strosp = ofl->ofl_osstrtab;
2793                 strtbl = ofl->ofl_strtab;
2794         } else {
2795                 strosp = ofl->ofl_osdynstr;
2796                 strtbl = ofl->ofl_dynstrtab;
2797         }

2799         /*
2800          * Traverse the shared object list looking for dependencies that have
2801          * versions defined within them.
2802          */
2803         for (APLIST_TRAVERSE(ofl->ofl_sos, idx1, ifl)) {
2804                 Half            _cnt;
2805                 Word            cnt = 0;
2806                 Vernaux         *_vnap, *vnap;
2807                 size_t          stoff;

2809                 if (!(ifl->ifl_flags & FLG_IF_VERNEED))
2810                         continue;

2812                 vnd->vn_version = VER_NEED_CURRENT;

2814                 (void) st_setstring(strtbl, ifl->ifl_soname, &stoff);
2815                 vnd->vn_file = stoff;

2817                 _vnap = vnap = (Vernaux *)(vnd + 1);

2819                 /*
2820                  * Traverse the version index list recording
2821                  * each version as a needed dependency.
2822                  */
2823                 for (_cnt = 0; _cnt <= ifl->ifl_vercnt; _cnt++) {
2824                         Ver_index       *vip = &ifl->ifl_verndx[_cnt];

2826                         if (vip->vi_flags & FLG_VER_REFER) {
2827                                 (void) st_setstring(strtbl, vip->vi_name,
```



```
2828                                     &stoff);
2829                                 vnap->vna_name = stoff;

2831                                 if (vip->vi_desc) {
2832                                         vnap->vna_hash = vip->vi_desc->vd_hash;
2833                                         vnap->vna_flags =
2834                                             vip->vi_desc->vd_flags;
2835                                 } else {
2836                                         vnap->vna_hash = 0;
2837                                         vnap->vna_flags = 0;
2838                                 }
2839                                 vnap->vna_other = vip->vi_overndx;

2841                                 /*
2842                                  * If version A inherits version B, then
2843                                  * B is implicit in A. It suffices for ld.so.1
2844                                  * to verify A at runtime and skip B. The
2845                                  * version normalization process sets the INFO
2846                                  * flag for the versions we want ld.so.1 to
2847                                  * skip.
2848                                  */
2849                                 if (vip->vi_flags & VER_FLG_INFO)
2850                                         vnap->vna_flags |= VER_FLG_INFO;

2852                                 _vnap = vnap;
2853                                 vnap++, cnt++;
2854                                 _vnap->vna_next =
2855                                     /* LINTED */
2856                                     (Word)((uintptr_t)vnap - (uintptr_t)_vnap);
2857                         }
2858                 }

2860                 _vnap->vna_next = 0;

2862                 /*
2863                  * Record the versions auxiliary array offset and
2864                  * the associated dependency count.
2865                  */
2866                 /* LINTED */
2867                 vnd->vn_aux = (Word)((uintptr_t)(vnd + 1) - (uintptr_t)vnd);
2868                 /* LINTED */
2869                 vnd->vn_cnt = (Half)cnt;

2871                 /*
2872                  * Record the next versions offset and update the version
2873                  * pointer.  Remember the previous version offset as the very
2874                  * last structures next pointer should be null.
2875                  */
2876                 _vnd = vnd;
2877                 vnd = (Verneed *)vnap, num++;
2878                 /* LINTED */
2879                 _vnd->vn_next = (Word)((uintptr_t)vnd - (uintptr_t)_vnd);
2880         }
2881         _vnd->vn_next = 0;

2883         /*
2884          * Use sh_link to record the associated string table section, and
2885          * sh_info to indicate the number of entries contained in the section.
2886          */
2887         /* LINTED */
2888         ofl->ofl_osverneed->os_shdr->sh_link = (Word)elf_ndxscn(strosp->os_scn);
2889         ofl->ofl_osverneed->os_shdr->sh_info = num;

2891         return (1);
2892 }
```

```
2894 /*
2895  * Update syminfo section.
2896  */
2897 static uintptr_t
2898 update_osyminfo(Ofl_desc *ofl)
2899 {
2900         Os_desc         *symosp, *infosp = ofl->ofl_ossyminfo;
2901         Syminfo         *sip = infosp->os_outdata->d_buf;
2902         Shdr            *shdr = infosp->os_shdr;
2903         char            *strtab;
2904         Aliste          idx;
2905         Sym_desc        *sdp;
2906         Sfltr_desc      *sftp;

2908         if (ofl->ofl_flags & FLG_OF_RELOBJ) {
2909                 symosp = ofl->ofl_ossymtab;
2910                 strtab = ofl->ofl_osstrtab->os_outdata->d_buf;
2911         } else {
2912                 symosp = ofl->ofl_osdynsym;
2913                 strtab = ofl->ofl_osdynstr->os_outdata->d_buf;
2914         }

2916         /* LINTED */
2917         infosp->os_shdr->sh_link = (Word)elf_ndxscn(symosp->os_scn);
2918         if (ofl->ofl_osdynamic)
2919                 infosp->os_shdr->sh_info =
2920                     /* LINTED */
2921                     (Word)elf_ndxscn(ofl->ofl_osdynamic->os_scn);

2923         /*
2924          * Update any references with the index into the dynamic table.
2925          */
2926         for (APLIST_TRAVERSE(ofl->ofl_symdtent, idx, sdp))
2927                 sip[sdp->sd_symndx].si_boundto = sdp->sd_file->ifl_neededndx;

2929         /*
2930          * Update any filtee references with the index into the dynamic table.
2931          */
2932         for (ALIST_TRAVERSE(ofl->ofl_symfltrs, idx, sftp)) {
2933                 Dfltr_desc      *dftp;

2935                 dftp = alist_item(ofl->ofl_dtsfltrs, sftp->sft_idx);
2936                 sip[sftp->sft_sdp->sd_symndx].si_boundto = dftp->dft_ndx;
2937         }

2939         /*
2940          * Display debugging information about section.
2941          */
2942         DBG_CALL(Dbg_syminfo_title(ofl->ofl_lml));
2943         if (DBG_ENABLED) {
2944                 Word    _cnt, cnt = shdr->sh_size / shdr->sh_entsize;
2945                 Sym     *symtab = symosp->os_outdata->d_buf;
2946                 Dyn     *dyn;

2948                 if (ofl->ofl_osdynamic)
2949                         dyn = ofl->ofl_osdynamic->os_outdata->d_buf;
2950                 else
2951                         dyn = NULL;

2953                 for (_cnt = 1; _cnt < cnt; _cnt++) {
2954                         if (sip[_cnt].si_flags || sip[_cnt].si_boundto)
2955                                 /* LINTED */
2956                                 DBG_CALL(Dbg_syminfo_entry(ofl->ofl_lml, _cnt,
2957                                     &sip[_cnt], &symtab[_cnt], strtab, dyn));
2958                 }
2959         }
```



```
2960         return (1);
2961 }

2963 /*
2964  * Build the output elf header.
2965  */
2966 static uintptr_t
2967 update_oehdr(Ofl_desc * ofl)
2968 {
2969         Ehdr    *ehdr = ofl->ofl_nehdr;

2971         /*
2972          * If an entry point symbol has already been established (refer
2973          * sym_validate()) simply update the elf header entry point with the
2974          * symbols value.  If no entry point is defined it will have been filled
2975          * with the start address of the first section within the text segment
2976          * (refer update_outfile()).
2977          */
2978         if (ofl->ofl_entry)
2979                 ehdr->e_entry =
2980                     ((Sym_desc *)(ofl->ofl_entry))->sd_sym->st_value;

2982         ehdr->e_ident[EI_DATA] = ld_targ.t_m.m_data;
2983         ehdr->e_version = ofl->ofl_dehdr->e_version;

2985         /*
2986          * When generating a relocatable object under -z symbolcap, set the
2987          * e_machine to be generic, and remove any e_flags.  Input relocatable
2988          * objects may identify alternative e_machine (m.machplus) and e_flags
2989          * values.  However, the functions within the created output object
2990          * are selected at runtime using the capabilities mechanism, which
2991          * supersedes the e-machine and e_flags information.  Therefore,
2992          * e_machine and e_flag values are not propagated to the output object,
2993          * as these values might prevent the kernel from loading the object
2994          * before the runtime linker gets control.
2995          */
2996         if (ofl->ofl_flags & FLG_OF_OTOSCAP) {
2997                 ehdr->e_machine = ld_targ.t_m.m_mach;
2998                 ehdr->e_flags = 0;
2999         } else {
3000                 /*
3001                  * Note. it may be necessary to update the e_flags field in the
3002                  * machine dependent section.
3003                  */
3004                 ehdr->e_machine = ofl->ofl_dehdr->e_machine;
3005                 ehdr->e_flags = ofl->ofl_dehdr->e_flags;

3007                 if (ehdr->e_machine != ld_targ.t_m.m_mach) {
3008                         if (ehdr->e_machine != ld_targ.t_m.m_machplus)
3009                                 return (S_ERROR);
3010                         if ((ehdr->e_flags & ld_targ.t_m.m_flagsplus) == 0)
3011                                 return (S_ERROR);
3012                 }
3013         }

3015         if (ofl->ofl_flags & FLG_OF_SHAROBJ)
3016                 ehdr->e_type = ET_DYN;
3017         else if (ofl->ofl_flags & FLG_OF_RELOBJ)
3018                 ehdr->e_type = ET_REL;
3019         else
3020                 ehdr->e_type = ET_EXEC;

3022         return (1);
3023 }

3025 /*
```

```
 * Perform move table expansion.
 */
static void
expand_move(Ofl_desc *ofl, Sym_desc *sdp, Move *mvp)
{
        Os_desc         *osp;
        uchar_t         *taddr, *taddr0;
        Sxword          offset;
        Half            cnt;
        uint_t          stride;

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

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

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

                /*
                 * Update the target address based upon the move entry size.
                 * This size was validated in ld_process_move().
                 */
                /* LINTED */
                switch (ELF_M_SIZE(mvp->m_info)) {
                case 1:
                        /* LINTED */
                        *taddr = (uchar_t)mvp->m_value;
                        taddr += stride;
                        break;
                case 2:
                        /* LINTED */
                        *((Half *)taddr) = (Half)mvp->m_value;
                        taddr += 2 * stride;
                        break;
                case 4:
                        /* LINTED */
                        *((Word *)taddr) = (Word)mvp->m_value;
                        taddr += 4 * stride;
                        break;
                case 8:
                        /* LINTED */
                        *((u_longlong_t *)taddr) = mvp->m_value;
                        taddr += 8 * stride;
                        break;
                }
        }
}

/*
 * Update Move sections.
 */
static void
update_move(Ofl_desc *ofl)
{
        Word            ndx = 0;
        ofl_flag_t      flags = ofl->ofl_flags;
        Move            *omvp;
        Aliste          idx1;
        Sym_desc        *sdp;
```



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

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

        /*
         * Update symbol entry index
         */
        for (APLIST_TRAVERSE(ofl->ofl_parsyms, idx1, sdp)) {
                Aliste          idx2;
                Mv_desc         *mdp;

                /*
                 * Expand move table
                 */
                if (sdp->sd_flags & FLG_SY_PAREXPN) {
                        const char      *str;

                        if (flags & FLG_OF_STATIC)
                                str = MSG_INTL(MSG_PSYM_EXPREASON1);
                        else if (ofl->ofl_flags1 & FLG_OF1_NOPARTI)
                                str = MSG_INTL(MSG_PSYM_EXPREASON2);
                        else
                                str = MSG_INTL(MSG_PSYM_EXPREASON3);

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

                        for (ALIST_TRAVERSE(sdp->sd_move, idx2, mdp)) {
                                DBG_CALL(Dbg_move_entry1(ofl->ofl_lml, 0,
                                    mdp->md_move, sdp));
                                expand_move(ofl, sdp, mdp->md_move);
                        }
                        continue;
                }

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

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

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

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

                        *omvp = *imvp;
```

```
3158                        if ((flags & FLG_OF_RELOBJ) == 0) {
3159                                if (ELF_ST_BIND(sym->st_info) == STB_LOCAL) {
3160                                        Os_desc *osp = sdp->sd_isc->is_osdesc;
3161                                        Word    ndx = osp->os_identndx;

3163                                        omvp->m_info =
3164                                            /* LINTED */
3165                                            ELF_M_INFO(ndx, imvp->m_info);

3167                                        if (ELF_ST_TYPE(sym->st_info) !=
3168                                            STT_SECTION) {
3169                                                omvp->m_poffset =
3170                                                    sym->st_value -
3171                                                    osp->os_shdr->sh_addr +
3172                                                    imvp->m_poffset;
3173                                        }
3174                                } else {
3175                                        omvp->m_info =
3176                                            /* LINTED */
3177                                            ELF_M_INFO(sdp->sd_symndx,
3178                                            imvp->m_info);
3179                                }
3180                        } else {
3181                                Boolean         isredloc = FALSE;

3183                                if ((ELF_ST_BIND(sym->st_info) == STB_LOCAL) &&
3184                                    (ofl->ofl_flags & FLG_OF_REDLSYM))
3185                                        isredloc = TRUE;

3187                                if (isredloc && !(sdp->sd_move)) {
3188                                        Os_desc *osp = sdp->sd_isc->is_osdesc;
3189                                        Word    ndx = osp->os_identndx;

3191                                        omvp->m_info =
3192                                            /* LINTED */
3193                                            ELF_M_INFO(ndx, imvp->m_info);

3195                                        omvp->m_poffset += sym->st_value;
3196                                } else {
3197                                        if (isredloc)
3198                                                DBG_CALL(Dbg_syms_reduce(ofl,
3199                                                    DBG_SYM_REDUCE_RETAIN,
3200                                                    sdp, idx,
3201                                                    ofl->ofl_osmove->os_name));

3203                                        omvp->m_info =
3204                                            /* LINTED */
3205                                            ELF_M_INFO(sdp->sd_symndx,
3206                                            imvp->m_info);
3207                                }
3208                        }

3210                        DBG_CALL(Dbg_move_entry1(ofl->ofl_lml, 0, omvp, sdp));
3211                        omvp++;
3212                        idx++;
3213                }
3214        }
3215 }

3217 /*
3218  * Scan through the SHT_GROUP output sections.  Update their sh_link/sh_info
3219  * fields as well as the section contents.
3220  */
3221 static uintptr_t
3222 update_ogroup(Ofl_desc *ofl)
3223 {
```



```
3224        Aliste          idx;
3225        Os_desc         *osp;
3226        uintptr_t       error = 0;

3228        for (APLIST_TRAVERSE(ofl->ofl_osgroups, idx, osp)) {
3229                Is_desc         *isp;
3230                Ifl_desc        *ifl;
3231                Shdr            *shdr = osp->os_shdr;
3232                Sym_desc        *sdp;
3233                Xword           i, grpcnt;
3234                Word            *gdata;

3236                /*
3237                 * Since input GROUP sections always create unique
3238                 * output GROUP sections - we know there is only one
3239                 * item on the list.
3240                 */
3241                isp = ld_os_first_isdesc(osp);

3243                ifl = isp->is_file;
3244                sdp = ifl->ifl_oldndx[isp->is_shdr->sh_info];
3245                shdr->sh_link = (Word)elf_ndxscn(ofl->ofl_ossymtab->os_scn);
3246                shdr->sh_info = sdp->sd_symndx;

3248                /*
3249                 * Scan through the group data section and update
3250                 * all of the links to new values.
3251                 */
3252                grpcnt = shdr->sh_size / shdr->sh_entsize;
3253                gdata = (Word *)osp->os_outdata->d_buf;

3255                for (i = 1; i < grpcnt; i++) {
3256                        Os_desc *_osp;
3257                        Is_desc *_isp = ifl->ifl_isdesc[gdata[i]];

3259                        /*
3260                         * If the referenced section didn't make it to the
3261                         * output file - just zero out the entry.
3262                         */
3263                        if ((_osp = _isp->is_osdesc) == NULL)
3264                                gdata[i] = 0;
3265                        else
3266                                gdata[i] = (Word)elf_ndxscn(_osp->os_scn);
3267                }
3268        }
3269        return (error);
3270 }

3272 static void
3273 update_ostrtab(Os_desc *osp, Str_tbl *stp, uint_t extra)
3274 {
3275        Elf_Data        *data;

3277        if (osp == NULL)
3278                return;

3280        data = osp->os_outdata;
3281        assert(data->d_size == (st_getstrtab_sz(stp) + extra));
3282        (void) st_setstrbuf(stp, data->d_buf, data->d_size - extra);
3283        /* If leaving an extra hole at the end, zero it */
3284        if (extra > 0)
3285                (void) memset((char *)data->d_buf + data->d_size - extra,
3286                    0x0, extra);
3287 }

3289 /*
```

```
 * Update capabilities information.
 *
 * If string table capabilities exist, then the associated string must be
 * translated into an offset into the string table.
 */
static void
update_oscap(Ofl_desc *ofl)
{
        Os_desc         *strosp, *cosp;
        Cap             *cap;
        Str_tbl         *strtbl;
        Capstr          *capstr;
        size_t          stoff;
        Aliste          idx1;

        /*
         * Determine which symbol table or string table is appropriate.
         */
        if (OFL_IS_STATIC_OBJ(ofl)) {
                strosp = ofl->ofl_osstrtab;
                strtbl = ofl->ofl_strtab;
        } else {
                strosp = ofl->ofl_osdynstr;
                strtbl = ofl->ofl_dynstrtab;
        }

        /*
         * If symbol capabilities exist, set the sh_link field of the .SUNW_cap
         * section to the .SUNW_capinfo section.
         */
        if (ofl->ofl_oscapinfo) {
                cosp = ofl->ofl_oscap;
                cosp->os_shdr->sh_link =
                    (Word)elf_ndxscn(ofl->ofl_oscapinfo->os_scn);
        }

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

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

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

        /*
         * Determine whether an object capability identifier, or object
         * machine/platform capabilities exists.
         */
        capstr = &ofl->ofl_ocapset.oc_id;
        if (capstr->cs_str) {
                (void) st_setstring(strtbl, capstr->cs_str, &stoff);
                cap[capstr->cs_ndx].c_un.c_ptr = stoff;
        }
        for (ALIST_TRAVERSE(ofl->ofl_ocapset.oc_plat.cl_val, idx1, capstr)) {
                (void) st_setstring(strtbl, capstr->cs_str, &stoff);
                cap[capstr->cs_ndx].c_un.c_ptr = stoff;
        }
        for (ALIST_TRAVERSE(ofl->ofl_ocapset.oc_mach.cl_val, idx1, capstr)) {
                (void) st_setstring(strtbl, capstr->cs_str, &stoff);
                cap[capstr->cs_ndx].c_un.c_ptr = stoff;
        }
```



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

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

                        capstr = &ocapset->oc_id;
                        if (capstr->cs_str) {
                                (void) st_setstring(strtbl, capstr->cs_str,
                                    &stoff);
                                cap[capstr->cs_ndx].c_un.c_ptr = stoff;
                        }
                        for (ALIST_TRAVERSE(ocapset->oc_plat.cl_val, idx2,
                            capstr)) {
                                (void) st_setstring(strtbl, capstr->cs_str,
                                    &stoff);
                                cap[capstr->cs_ndx].c_un.c_ptr = stoff;
                        }
                        for (ALIST_TRAVERSE(ocapset->oc_mach.cl_val, idx2,
                            capstr)) {
                                (void) st_setstring(strtbl, capstr->cs_str,
                                    &stoff);
                                cap[capstr->cs_ndx].c_un.c_ptr = stoff;
                        }
                }
        }
}

/*
 * Update the .SUNW_capinfo, and possibly the .SUNW_capchain sections.
 */
static void
update_oscapinfo(Ofl_desc *ofl)
{
        Os_desc         *symosp, *ciosp, *ccosp = NULL;
        Capinfo         *ocapinfo;
        Capchain        *ocapchain;
        Cap_avlnode     *cav;
        Word            chainndx = 0;

        /*
         * Determine which symbol table is appropriate.
         */
        if (OFL_IS_STATIC_OBJ(ofl))
                symosp = ofl->ofl_ossymtab;
        else
                symosp = ofl->ofl_osdynsym;

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

        if (OFL_IS_STATIC_OBJ(ofl) == 0) {
                ccosp = ofl->ofl_oscapchain;
                ciosp->os_shdr->sh_info = (Word)elf_ndxscn(ccosp->os_scn);
```

```
3422        }
3423
3424        /*
3425         * Establish the data for each section.  The first element of each
3426         * section defines the section's version number.
3427         */
3428        ocapinfo = ciosp->os_outdata->d_buf;
3429        ocapinfo[0] = CAPINFO_CURRENT;
3430        if (ccosp) {
3431                ocapchain = ccosp->os_outdata->d_buf;
3432                ocapchain[chainndx++] = CAPCHAIN_CURRENT;
3433        }
3434
3435        /*
3436         * Traverse all capabilities families.  Each member has a .SUNW_capinfo
3437         * assignment.  The .SUNW_capinfo entry differs for relocatable objects
3438         * and dynamic objects.
3439         *
3440         * Relocatable objects:
3441         *                      ELF_C_GROUP             ELF_C_SYM
3442         *
3443         * Family lead:         CAPINFO_SUNW_GLOB       lead symbol index
3444         * Family lead alias:   CAPINFO_SUNW_GLOB       lead symbol index
3445         * Family member:       .SUNW_cap index         lead symbol index
3446         *
3447         * Dynamic objects:
3448         *                      ELF_C_GROUP             ELF_C_SYM
3449         *
3450         * Family lead:         CAPINFO_SUNW_GLOB       .SUNW_capchain index
3451         * Family lead alias:   CAPINFO_SUNW_GLOB       .SUNW_capchain index
3452         * Family member:       .SUNW_cap index         lead symbol index
3453         *
3454         * The ELF_C_GROUP field identifies a capabilities symbol.  Lead
3455         * capability symbols, and lead capability aliases are identified by
3456         * a CAPINFO_SUNW_GLOB group identifier.  For family members, the
3457         * ELF_C_GROUP provides an index to the associate capabilities group
3458         * (i.e, an index into the SUNW_cap section that defines a group).
3459         *
3460         * For relocatable objects, the ELF_C_SYM field identifies the lead
3461         * capability symbol.  For the lead symbol itself, the .SUNW_capinfo
3462         * index is the same as the ELF_C_SYM value.  For lead alias symbols,
3463         * the .SUNW_capinfo index differs from the ELF_C_SYM value.  This
3464         * differentiation of CAPINFO_SUNW_GLOB symbols allows ld(1) to
3465         * identify, and propagate lead alias symbols.  For example, the lead
3466         * capability symbol memcpy() would have the ELF_C_SYM for memcpy(),
3467         * and the lead alias _memcpy() would also have the ELF_C_SYM for
3468         * memcpy().
3469         *
3470         * For dynamic objects, both a lead capability symbol, and alias symbol
3471         * would have a ELF_C_SYM value that represents the same capability
3472         * chain index.  The capability chain allows ld.so.1 to traverse a
3473         * family chain for a given lead symbol, and select the most appropriate
3474         * family member.  The .SUNW_capchain array contains a series of symbol
3475         * indexes for each family member:
3476         *
3477         *    chaincap[n]  chaincap[n + 1]  chaincap[n + 2]  chaincap[n + x]
3478         *      foo() ndx    foo%x() ndx        foo%y() ndx     0
3479         *
3480         * For family members, the ELF_C_SYM value associates the capability
3481         * members with their family lead symbol.  This association, although
3482         * unused within a dynamic object, allows ld(1) to identify, and
3483         * propagate family members when processing relocatable objects.
3484         */
3485        for (cav = avl_first(ofl->ofl_capfamilies); cav;
3486            cav = AVL_NEXT(ofl->ofl_capfamilies, cav)) {
3487                Cap_sym         *csp;
```



```
3488                Aliste          idx;
3489                Sym_desc        *asdp, *lsdp = cav->cn_symavlnode.sav_sdp;
3490
3491                if (ccosp) {
3492                        /*
3493                         * For a dynamic object, identify this lead symbol, and
3494                         * point it to the head of a capability chain.  Set the
3495                         * head of the capability chain to the same lead symbol.
3496                         */
3497                        ocapinfo[lsdp->sd_symndx] =
3498                            ELF_C_INFO(chainndx, CAPINFO_SUNW_GLOB);
3499                        ocapchain[chainndx] = lsdp->sd_symndx;
3500                } else {
3501                        /*
3502                         * For a relocatable object, identify this lead symbol,
3503                         * and set the lead symbol index to itself.
3504                         */
3505                        ocapinfo[lsdp->sd_symndx] =
3506                            ELF_C_INFO(lsdp->sd_symndx, CAPINFO_SUNW_GLOB);
3507                }
3508
3509                /*
3510                 * Gather any lead symbol aliases.
3511                 */
3512                for (APLIST_TRAVERSE(cav->cn_aliases, idx, asdp)) {
3513                        if (ccosp) {
3514                                /*
3515                                 * For a dynamic object, identify this lead
3516                                 * alias symbol, and point it to the same
3517                                 * capability chain index as the lead symbol.
3518                                 */
3519                                ocapinfo[asdp->sd_symndx] =
3520                                    ELF_C_INFO(chainndx, CAPINFO_SUNW_GLOB);
3521                        } else {
3522                                /*
3523                                 * For a relocatable object, identify this lead
3524                                 * alias symbol, and set the lead symbol index
3525                                 * to the lead symbol.
3526                                 */
3527                                ocapinfo[asdp->sd_symndx] =
3528                                    ELF_C_INFO(lsdp->sd_symndx,
3529                                    CAPINFO_SUNW_GLOB);
3530                        }
3531                }
3532
3533                chainndx++;
3534
3535                /*
3536                 * Gather the family members.
3537                 */
3538                for (APLIST_TRAVERSE(cav->cn_members, idx, csp)) {
3539                        Sym_desc        *msdp = csp->cs_sdp;
3540
3541                        /*
3542                         * Identify the members capability group, and the lead
3543                         * symbol of the family this symbol is a member of.
3544                         */
3545                        ocapinfo[msdp->sd_symndx] =
3546                            ELF_C_INFO(lsdp->sd_symndx, csp->cs_group->cg_ndx);
3547                        if (ccosp) {
3548                                /*
3549                                 * For a dynamic object, set the next capability
3550                                 * chain to point to this family member.
3551                                 */
3552                                ocapchain[chainndx++] = msdp->sd_symndx;
3553                        }
```

```
3554                 }
3555
3556                 /*
3557                  * Any chain of family members is terminated with a 0 element.
3558                  */
3559                 if (ccosp)
3560                         ocapchain[chainndx++] = 0;
3561         }
3562 }
3563
3564 /*
3565  * Translate the shdr->sh_{link, info} from its input section value to that
3566  * of the corresponding shdr->sh_{link, info} output section value.
3567  */
3568 static Word
3569 translate_link(Ofl_desc *ofl, Os_desc *osp, Word link, const char *msg)
3570 {
3571         Is_desc         *isp;
3572         Ifl_desc        *ifl;
3573
3574         /*
3575          * Don't translate the special section numbers.
3576          */
3577         if (link >= SHN_LORESERVE)
3578                 return (link);
3579
3580         /*
3581          * Does this output section translate back to an input file.  If not
3582          * then there is no translation to do.  In this case we will assume that
3583          * if sh_link has a value, it's the right value.
3584          */
3585         isp = ld_os_first_isdesc(osp);
3586         if ((ifl = isp->is_file) == NULL)
3587                 return (link);
3588
3589         /*
3590          * Sanity check to make sure that the sh_{link, info} value
3591          * is within range for the input file.
3592          */
3593         if (link >= ifl->ifl_shnum) {
3594                 ld_eprintf(ofl, ERR_WARNING, msg, ifl->ifl_name,
3595                     EC_WORD(isp->is_scnndx), isp->is_name, EC_XWORD(link));
3596                 return (link);
3597         }
3598
3599         /*
3600          * Follow the link to the input section.
3601          */
3602         if ((isp = ifl->ifl_isdesc[link]) == NULL)
3603                 return (0);
3604         if ((osp = isp->is_osdesc) == NULL)
3605                 return (0);
3606
3607         /* LINTED */
3608         return ((Word)elf_ndxscn(osp->os_scn));
3609 }
3610
3611 /*
3612  * Having created all of the necessary sections, segments, and associated
3613  * headers, fill in the program headers and update any other data in the
3614  * output image.  Some general rules:
3615  *
3616  *  -   If an interpreter is required always generate a PT_PHDR entry as
3617  *      well.  It is this entry that triggers the kernel into passing the
3618  *      interpreter an aux vector instead of just a file descriptor.
3619  *
```



```
3620  *  -   When generating an image that will be interpreted (ie. a dynamic
3621  *      executable, a shared object, or a static executable that has been
3622  *      provided with an interpreter - weird, but possible), make the initial
3623  *      loadable segment include both the ehdr and phdr[].  Both of these
3624  *      tables are used by the interpreter therefore it seems more intuitive
3625  *      to explicitly defined them as part of the mapped image rather than
3626  *      relying on page rounding by the interpreter to allow their access.
3627  *
3628  *  -   When generating a static image that does not require an interpreter
3629  *      have the first loadable segment indicate the address of the first
3630  *      .section as the start address (things like /kernel/unix and ufsboot
3631  *      expect this behavior).
3632  */
3633 uintptr_t
3634 ld_update_outfile(Ofl_desc *ofl)
3635 {
3636         Addr            size, etext, vaddr;
3637         Sg_desc         *sgp;
3638         Sg_desc         *dtracesgp = NULL, *capsgp = NULL, *intpsgp = NULL;
3639         Os_desc         *osp;
3640         int             phdrndx = 0, segndx = -1, secndx, intppndx, intpsndx;
3641         int             dtracepndx, dtracesndx, cappndx, capsndx;
3642         Ehdr            *ehdr = ofl->ofl_nehdr;
3643         Shdr            *hshdr;
3644         Phdr            *_phdr = NULL;
3645         Word            phdrsz = (ehdr->e_phnum * ehdr->e_phentsize), shscnndx;
3646         ofl_flag_t      flags = ofl->ofl_flags;
3647         Word            ehdrsz = ehdr->e_ehsize;
3648         Boolean         nobits;
3649         Off             offset;
3650         Aliste          idx1;
3651
3652         /*
3653          * Initialize the starting address for the first segment.  Executables
3654          * have different starting addresses depending upon the target ABI,
3655          * where as shared objects have a starting address of 0.  If this is
3656          * a 64-bit executable that is being constructed to run in a restricted
3657          * address space, use an alternative origin that will provide more free
3658          * address space for the the eventual process.
3659          */
3660         if (ofl->ofl_flags & FLG_OF_EXEC) {
3661 #if     defined(_ELF64)
3662                 if (ofl->ofl_ocapset.oc_sf_1.cm_val & SF1_SUNW_ADDR32)
3663                         vaddr = ld_targ.t_m.m_segm_aorigin;
3664                 else
3665 #endif
3666                         vaddr = ld_targ.t_m.m_segm_origin;
3667         } else
3668                 vaddr = 0;
3669
3670         /*
3671          * Loop through the segment descriptors and pick out what we need.
3672          */
3673         DBG_CALL(Dbg_seg_title(ofl->ofl_lml));
3674         for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
3675                 Phdr            *phdr = &(sgp->sg_phdr);
3676                 Xword           p_align;
3677                 Aliste          idx2;
3678                 Sym_desc        *sdp;
3679
3680                 segndx++;
3681
3682                 /*
3683                  * If an interpreter is required generate a PT_INTERP and
3684                  * PT_PHDR program header entry.  The PT_PHDR entry describes
3685                  * the program header table itself.  This information will be
```

```
3686                 * passed via the aux vector to the interpreter (ld.so.1).
3687                 * The program header array is actually part of the first
3688                 * loadable segment (and the PT_PHDR entry is the first entry),
3689                 * therefore its virtual address isn't known until the first
3690                 * loadable segment is processed.
3691                 */
3692                if (phdr->p_type == PT_PHDR) {
3693                        if (ofl->ofl_osinterp) {
3694                                phdr->p_offset = ehdr->e_phoff;
3695                                phdr->p_filesz = phdr->p_memsz = phdrsz;

3697                                DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3698                                ofl->ofl_phdr[phdrndx++] = *phdr;
3699                        }
3700                        continue;
3701                }
3702                if (phdr->p_type == PT_INTERP) {
3703                        if (ofl->ofl_osinterp) {
3704                                intpsgp = sgp;
3705                                intpsndx = segndx;
3706                                intppndx = phdrndx++;
3707                        }
3708                        continue;
3709                }

3711                /*
3712                 * If we are creating a PT_SUNWDTRACE segment, remember where
3713                 * the program header is.  The header values are assigned after
3714                 * update_osym() has completed and the symbol table addresses
3715                 * have been updated.
3716                 */
3717                if (phdr->p_type == PT_SUNWDTRACE) {
3718                        if (ofl->ofl_dtracesym &&
3719                            ((flags & FLG_OF_RELOBJ) == 0)) {
3720                                dtracesgp = sgp;
3721                                dtracesndx = segndx;
3722                                dtracepndx = phdrndx++;
3723                        }
3724                        continue;
3725                }

3727                /*
3728                 * If a hardware/software capabilities section is required,
3729                 * generate the PT_SUNWCAP header.  Note, as this comes before
3730                 * the first loadable segment, we don't yet know its real
3731                 * virtual address.  This is updated later.
3732                 */
3733                if (phdr->p_type == PT_SUNWCAP) {
3734                        if (ofl->ofl_oscap && (ofl->ofl_flags & FLG_OF_PTCAP) &&
3735                            ((flags & FLG_OF_RELOBJ) == 0)) {
3736                                capsgp = sgp;
3737                                capsndx = segndx;
3738                                cappndx = phdrndx++;
3739                        }
3740                        continue;
3741                }

3743                /*
3744                 * As the dynamic program header occurs after the loadable
3745                 * headers in the segment descriptor table, all the address
3746                 * information for the .dynamic output section will have been
3747                 * figured out by now.
3748                 */
3749                if (phdr->p_type == PT_DYNAMIC) {
3750                        if (OFL_ALLOW_DYNSYM(ofl)) {
3751                                Shdr    *shdr = ofl->ofl_osdynamic->os_shdr;
```



```
3753                                phdr->p_vaddr = shdr->sh_addr;
3754                                phdr->p_offset = shdr->sh_offset;
3755                                phdr->p_filesz = shdr->sh_size;
3756                                phdr->p_flags = ld_targ.t_m.m_dataseg_perm;

3758                                DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3759                                ofl->ofl_phdr[phdrndx++] = *phdr;
3760                        }
3761                        continue;
3762                }

3764                /*
3765                 * As the unwind (.eh_frame_hdr) program header occurs after
3766                 * the loadable headers in the segment descriptor table, all
3767                 * the address information for the .eh_frame output section
3768                 * will have been figured out by now.
3769                 */
3770                if (phdr->p_type == PT_SUNW_UNWIND) {
3771                        Shdr        *shdr;

3773                        if (ofl->ofl_unwindhdr == NULL)
3774                                continue;

3776                        shdr = ofl->ofl_unwindhdr->os_shdr;

3778                        phdr->p_flags = PF_R;
3779                        phdr->p_vaddr = shdr->sh_addr;
3780                        phdr->p_memsz = shdr->sh_size;
3781                        phdr->p_filesz = shdr->sh_size;
3782                        phdr->p_offset = shdr->sh_offset;
3783                        phdr->p_align = shdr->sh_addralign;
3784                        phdr->p_paddr = 0;
3785                        ofl->ofl_phdr[phdrndx++] = *phdr;
3786                        continue;
3787                }

3789                /*
3790                 * The sunwstack program is used to convey non-default
3791                 * flags for the process stack. Only emit it if it would
3792                 * change the default.
3793                 */
3794                if (phdr->p_type == PT_SUNWSTACK) {
3795                        if (((flags & FLG_OF_RELOBJ) == 0) &&
3796                            ((sgp->sg_flags & FLG_SG_DISABLED) == 0))
3797                                ofl->ofl_phdr[phdrndx++] = *phdr;
3798                        continue;
3799                }

3801                /*
3802                 * As the TLS program header occurs after the loadable
3803                 * headers in the segment descriptor table, all the address
3804                 * information for the .tls output section will have been
3805                 * figured out by now.
3806                 */
3807                if (phdr->p_type == PT_TLS) {
3808                        Os_desc         *tlsosp;
3809                        Shdr            *lastfileshdr = NULL;
3810                        Shdr            *firstshdr = NULL, *lastshdr;
3811                        Aliste          idx;

3813                        if (ofl->ofl_ostlsseg == NULL)
3814                                continue;

3816                        /*
3817                         * Scan the output sections that have contributed TLS.
```

```
3818                         * Remember the first and last so as to determine the
3819                         * TLS memory size requirement.  Remember the last
3820                         * progbits section to determine the TLS data
3821                         * contribution, which determines the TLS program
3822                         * header filesz.
3823                         */
3824                        for (APLIST_TRAVERSE(ofl->ofl_ostlsseg, idx, tlsosp)) {
3825                                Shdr    *tlsshdr = tlsosp->os_shdr;

3827                                if (firstshdr == NULL)
3828                                        firstshdr = tlsshdr;
3829                                if (tlsshdr->sh_type != SHT_NOBITS)
3830                                        lastfileshdr = tlsshdr;
3831                                lastshdr = tlsshdr;
3832                        }

3834                        phdr->p_flags = PF_R | PF_W;
3835                        phdr->p_vaddr = firstshdr->sh_addr;
3836                        phdr->p_offset = firstshdr->sh_offset;
3837                        phdr->p_align = firstshdr->sh_addralign;

3839                        /*
3840                         * Determine the initialized TLS data size.  This
3841                         * address range is from the start of the TLS segment
3842                         * to the end of the last piece of initialized data.
3843                         */
3844                        if (lastfileshdr)
3845                                phdr->p_filesz = lastfileshdr->sh_offset +
3846                                    lastfileshdr->sh_size - phdr->p_offset;
3847                        else
3848                                phdr->p_filesz = 0;

3850                        /*
3851                         * Determine the total TLS memory size.  This includes
3852                         * all TLS data and TLS uninitialized data.  This
3853                         * address range is from the start of the TLS segment
3854                         * to the memory address of the last piece of
3855                         * uninitialized data.
3856                         */
3857                        phdr->p_memsz = lastshdr->sh_addr +
3858                            lastshdr->sh_size - phdr->p_vaddr;

3860                        DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3861                        ofl->ofl_phdr[phdrndx] = *phdr;
3862                        ofl->ofl_tlsphdr = &ofl->ofl_phdr[phdrndx++];
3863                        continue;
3864                }

3866                /*
3867                 * If this is an empty segment declaration, it will occur after
3868                 * all other loadable segments.  As empty segments can be
3869                 * defined with fixed addresses, make sure that no loadable
3870                 * segments overlap.  This might occur as the object evolves
3871                 * and the loadable segments grow, thus encroaching upon an
3872                 * existing segment reservation.
3873                 *
3874                 * Segments are only created for dynamic objects, thus this
3875                 * checking can be skipped when building a relocatable object.
3876                 */
3877                if (!(flags & FLG_OF_RELOBJ) &&
3878                    (sgp->sg_flags & FLG_SG_EMPTY)) {
3879                        int     i;
3880                        Addr    v_e;

3882                        vaddr = phdr->p_vaddr;
3883                        phdr->p_memsz = sgp->sg_length;
```



```
3884                        DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3885                        ofl->ofl_phdr[phdrndx++] = *phdr;

3887                        if (phdr->p_type != PT_LOAD)
3888                                continue;

3890                        v_e = vaddr + phdr->p_memsz;

3892                        /*
3893                         * Check overlaps
3894                         */
3895                        for (i = 0; i < phdrndx - 1; i++) {
3896                                Addr    p_s = (ofl->ofl_phdr[i]).p_vaddr;
3897                                Addr    p_e;

3899                                if ((ofl->ofl_phdr[i]).p_type != PT_LOAD)
3900                                        continue;

3902                                p_e = p_s + (ofl->ofl_phdr[i]).p_memsz;
3903                                if (((p_s <= vaddr) && (p_e > vaddr)) ||
3904                                    ((vaddr <= p_s) && (v_e > p_s)))
3905                                        ld_eprintf(ofl, ERR_WARNING,
3906                                            MSG_INTL(MSG_UPD_SEGOVERLAP),
3907                                            ofl->ofl_name, EC_ADDR(p_e),
3908                                            sgp->sg_name, EC_ADDR(vaddr));
3909                        }
3910                        continue;
3911                }

3913                /*
3914                 * Having processed any of the special program headers any
3915                 * remaining headers will be built to express individual
3916                 * segments.  Segments are only built if they have output
3917                 * section descriptors associated with them (ie. some form of
3918                 * input section has been matched to this segment).
3919                 */
3920                if (sgp->sg_osdescs == NULL)
3921                        continue;

3923                /*
3924                 * Determine the segments offset and size from the section
3925                 * information provided from elf_update().
3926                 * Allow for multiple NOBITS sections.
3927                 */
3928                osp = sgp->sg_osdescs->apl_data[0];
3929                hshdr = osp->os_shdr;

3931                phdr->p_filesz = 0;
3932                phdr->p_memsz = 0;
3933                phdr->p_offset = offset = hshdr->sh_offset;

3935                nobits = ((hshdr->sh_type == SHT_NOBITS) &&
3936                    ((sgp->sg_flags & FLG_SG_PHREQ) == 0));

3938                for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
3939                        Shdr    *shdr = osp->os_shdr;

3941                        p_align = 0;
3942                        if (shdr->sh_addralign > p_align)
3943                                p_align = shdr->sh_addralign;

3945                        offset = (Off)S_ROUND(offset, shdr->sh_addralign);
3946                        offset += shdr->sh_size;

3948                        if (shdr->sh_type != SHT_NOBITS) {
3949                                if (nobits) {
```

```
3950                                    ld_eprintf(ofl, ERR_FATAL,
3951                                        MSG_INTL(MSG_UPD_NOBITS));
3952                                    return (S_ERROR);
3953                            }
3954                            phdr->p_filesz = offset - phdr->p_offset;
3955                    } else if ((sgp->sg_flags & FLG_SG_PHREQ) == 0)
3956                            nobits = TRUE;
3957            }
3958            phdr->p_memsz = offset - hshdr->sh_offset;

3960            /*
3961             * If this is the first loadable segment of a dynamic object,
3962             * or an interpreter has been specified (a static object built
3963             * with an interpreter will still be given a PT_HDR entry), then
3964             * compensate for the elf header and program header array.  Both
3965             * of these are actually part of the loadable segment as they
3966             * may be inspected by the interpreter.  Adjust the segments
3967             * size and offset accordingly.
3968             */
3969            if ((_phdr == NULL) && (phdr->p_type == PT_LOAD) &&
3970                ((ofl->ofl_osinterp) || (flags & FLG_OF_DYNAMIC)) &&
3971                (!(ofl->ofl_dtflags_1 & DF_1_NOHDR))) {
3972                    size = (Addr)S_ROUND((phdrsz + ehdrsz),
3973                        hshdr->sh_addralign);
3974                    phdr->p_offset -= size;
3975                    phdr->p_filesz += size;
3976                    phdr->p_memsz += size;
3977            }

3979            /*
3980             * If segment size symbols are required (specified via a
3981             * mapfile) update their value.
3982             */
3983            for (APLIST_TRAVERSE(sgp->sg_sizesym, idx2, sdp))
3984                    sdp->sd_sym->st_value = phdr->p_memsz;

3986            /*
3987             * If no file content has been assigned to this segment (it
3988             * only contains no-bits sections), then reset the offset for
3989             * consistency.
3990             */
3991            if (phdr->p_filesz == 0)
3992                    phdr->p_offset = 0;

3994            /*
3995             * If a virtual address has been specified for this segment
3996             * from a mapfile use it and make sure the previous segment
3997             * does not run into this segment.
3998             */
3999            if (phdr->p_type == PT_LOAD) {
4000                    if ((sgp->sg_flags & FLG_SG_P_VADDR)) {
4001                            if (_phdr && (vaddr > phdr->p_vaddr) &&
4002                                (phdr->p_type == PT_LOAD))
4003                                    ld_eprintf(ofl, ERR_WARNING,
4004                                        MSG_INTL(MSG_UPD_SEGOVERLAP),
4005                                        ofl->ofl_name, EC_ADDR(vaddr),
4006                                        sgp->sg_name,
4007                                        EC_ADDR(phdr->p_vaddr));
4008                            vaddr = phdr->p_vaddr;
4009                            phdr->p_align = 0;
4010                    } else {
4011                            vaddr = phdr->p_vaddr =
4012                                (Addr)S_ROUND(vaddr, phdr->p_align);
4013                    }
4014            }
```



```
4016            /*
4017             * Adjust the address offset and p_align if needed.
4018             */
4019            if (((sgp->sg_flags & FLG_SG_P_VADDR) == 0) &&
4020                ((ofl->ofl_dtflags_1 & DF_1_NOHDR) == 0)) {
4021                    if (phdr->p_align != 0)
4022                            vaddr += phdr->p_offset % phdr->p_align;
4023                    else
4024                            vaddr += phdr->p_offset;
4025                    phdr->p_vaddr = vaddr;
4026            }

4028            /*
4029             * If an interpreter is required set the virtual address of the
4030             * PT_PHDR program header now that we know the virtual address
4031             * of the loadable segment that contains it.  Update the
4032             * PT_SUNWCAP header similarly.
4033             */
4034            if ((_phdr == NULL) && (phdr->p_type == PT_LOAD)) {
4035                    _phdr = phdr;

4037                    if ((ofl->ofl_dtflags_1 & DF_1_NOHDR) == 0) {
4038                            if (ofl->ofl_osinterp)
4039                                    ofl->ofl_phdr[0].p_vaddr =
4040                                        vaddr + ehdrsz;

4042                            /*
4043                             * Finally, if we're creating a dynamic object
4044                             * (or a static object in which an interpreter
4045                             * is specified) update the vaddr to reflect
4046                             * the address of the first section within this
4047                             * segment.
4048                             */
4049                            if ((ofl->ofl_osinterp) ||
4050                                (flags & FLG_OF_DYNAMIC))
4051                                    vaddr += size;
4052                    } else {
4053                            /*
4054                             * If the DF_1_NOHDR flag was set, and an
4055                             * interpreter is being generated, the PT_PHDR
4056                             * will not be part of any loadable segment.
4057                             */
4058                            if (ofl->ofl_osinterp) {
4059                                    ofl->ofl_phdr[0].p_vaddr = 0;
4060                                    ofl->ofl_phdr[0].p_memsz = 0;
4061                                    ofl->ofl_phdr[0].p_flags = 0;
4062                            }
4063                    }
4064            }

4066            /*
4067             * Ensure the ELF entry point defaults to zero.  Typically, this
4068             * value is overridden in update_oehdr() to one of the standard
4069             * entry points.  Historically, this default was set to the
4070             * address of first executable section, but this has since been
4071             * found to be more confusing than it is helpful.
4072             */
4073            ehdr->e_entry = 0;

4075            DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));

4077            /*
4078             * Traverse the output section descriptors for this segment so
4079             * that we can update the section headers addresses.  We've
4080             * calculated the virtual address of the initial section within
4081             * this segment, so each successive section can be calculated
```

```
4082                 * based on their offsets from each other.
4083                 */
4084                secndx = 0;
4085                hshdr = 0;
4086                for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
4087                        Shdr    *shdr = osp->os_shdr;

4089                        if (shdr->sh_link)
4090                                shdr->sh_link = translate_link(ofl, osp,
4091                                    shdr->sh_link, MSG_INTL(MSG_FIL_INVSHLINK));

4093                        if (shdr->sh_info && (shdr->sh_flags & SHF_INFO_LINK))
4094                                shdr->sh_info = translate_link(ofl, osp,
4095                                    shdr->sh_info, MSG_INTL(MSG_FIL_INVSHINFO));

4097                        if (!(flags & FLG_OF_RELOBJ) &&
4098                            (phdr->p_type == PT_LOAD)) {
4099                                if (hshdr)
4100                                        vaddr += (shdr->sh_offset -
4101                                            hshdr->sh_offset);

4103                                shdr->sh_addr = vaddr;
4104                                hshdr = shdr;
4105                        }

4107                        DBG_CALL(Dbg_seg_os(ofl, osp, secndx));
4108                        secndx++;
4109                }

4111                /*
4112                 * Establish the virtual address of the end of the last section
4113                 * in this segment so that the next segments offset can be
4114                 * calculated from this.
4115                 */
4116                if (hshdr)
4117                        vaddr += hshdr->sh_size;

4119                /*
4120                 * Output sections for this segment complete.  Adjust the
4121                 * virtual offset for the last sections size, and make sure we
4122                 * haven't exceeded any maximum segment length specification.
4123                 */
4124                if ((sgp->sg_length != 0) && (sgp->sg_length < phdr->p_memsz)) {
4125                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_UPD_LARGSIZE),
4126                            ofl->ofl_name, sgp->sg_name,
4127                            EC_XWORD(phdr->p_memsz), EC_XWORD(sgp->sg_length));
4128                        return (S_ERROR);
4129                }

4131                if (phdr->p_type == PT_NOTE) {
4132                        phdr->p_vaddr = 0;
4133                        phdr->p_paddr = 0;
4134                        phdr->p_align = 0;
4135                        phdr->p_memsz = 0;
4136                }

4138                if ((phdr->p_type != PT_NULL) && !(flags & FLG_OF_RELOBJ))
4139                        ofl->ofl_phdr[phdrndx++] = *phdr;
4140        }

4142        /*
4143         * Update any new output sections.  When building the initial output
4144         * image, a number of sections were created but left uninitialized (eg.
4145         * .dynsym, .dynstr, .symtab, .symtab, etc.).  Here we update these
4146         * sections with the appropriate data.  Other sections may still be
4147         * modified via reloc_process().
```



```
4148         *
4149         * Copy the interpreter name into the .interp section.
4150         */
4151        if (ofl->ofl_interp)
4152                (void) strcpy((char *)ofl->ofl_osinterp->os_outdata->d_buf,
4153                    ofl->ofl_interp);

4155        /*
4156         * Update the .shstrtab, .strtab and .dynstr sections.
4157         */
4158        update_ostrtab(ofl->ofl_osshstrtab, ofl->ofl_shdrsttab, 0);
4159        update_ostrtab(ofl->ofl_osstrtab, ofl->ofl_strtab, 0);
4160        update_ostrtab(ofl->ofl_osdynstr, ofl->ofl_dynstrtab, DYNSTR_EXTRA_PAD);

4162        /*
4163         * Build any output symbol tables, the symbols information is copied
4164         * and updated into the new output image.
4165         */
4166        if ((etext = update_osym(ofl)) == (Addr)S_ERROR)
4167                return (S_ERROR);

4169        /*
4170         * If we have an PT_INTERP phdr, update it now from the associated
4171         * section information.
4172         */
4173        if (intpsgp) {
4174                Phdr    *phdr = &(intpsgp->sg_phdr);
4175                Shdr    *shdr = ofl->ofl_osinterp->os_shdr;

4177                phdr->p_vaddr = shdr->sh_addr;
4178                phdr->p_offset = shdr->sh_offset;
4179                phdr->p_memsz = phdr->p_filesz = shdr->sh_size;
4180                phdr->p_flags = PF_R;

4182                DBG_CALL(Dbg_seg_entry(ofl, intpsndx, intpsgp));
4183                ofl->ofl_phdr[intppndx] = *phdr;
4184        }

4186        /*
4187         * If we have a PT_SUNWDTRACE phdr, update it now with the address of
4188         * the symbol.  It's only now been updated via update_sym().
4189         */
4190        if (dtracesgp) {
4191                Phdr            *aphdr, *phdr = &(dtracesgp->sg_phdr);
4192                Sym_desc        *sdp = ofl->ofl_dtracesym;

4194                phdr->p_vaddr = sdp->sd_sym->st_value;
4195                phdr->p_memsz = sdp->sd_sym->st_size;

4197                /*
4198                 * Take permissions from the segment that the symbol is
4199                 * associated with.
4200                 */
4201                aphdr = &sdp->sd_isc->is_osdesc->os_sgdesc->sg_phdr;
4202                assert(aphdr);
4203                phdr->p_flags = aphdr->p_flags;

4205                DBG_CALL(Dbg_seg_entry(ofl, dtracesndx, dtracesgp));
4206                ofl->ofl_phdr[dtracepndx] = *phdr;
4207        }

4209        /*
4210         * If we have a PT_SUNWCAP phdr, update it now from the associated
4211         * section information.
4212         */
4213        if (capsgp) {
```

```
4214                Phdr    *phdr = &(capsgp->sg_phdr);
4215                Shdr    *shdr = ofl->ofl_oscap->os_shdr;
4216
4217                phdr->p_vaddr = shdr->sh_addr;
4218                phdr->p_offset = shdr->sh_offset;
4219                phdr->p_memsz = phdr->p_filesz = shdr->sh_size;
4220                phdr->p_flags = PF_R;
4221
4222                DBG_CALL(Dbg_seg_entry(ofl, capsndx, capsgp));
4223                ofl->ofl_phdr[cappndx] = *phdr;
4224        }
4225
4226        /*
4227         * Update the GROUP sections.
4228         */
4229        if (update_ogroup(ofl) == S_ERROR)
4230                return (S_ERROR);
4231
4232        /*
4233         * Update Move Table.
4234         */
4235        if (ofl->ofl_osmove || ofl->ofl_isparexpn)
4236                update_move(ofl);
4237
4238        /*
4239         * Build any output headers, version information, dynamic structure and
4240         * syminfo structure.
4241         */
4242        if (update_oehdr(ofl) == S_ERROR)
4243                return (S_ERROR);
4244        if (!(flags & FLG_OF_NOVERSEC)) {
4245                if ((flags & FLG_OF_VERDEF) &&
4246                    (update_overdef(ofl) == S_ERROR))
4247                        return (S_ERROR);
4248                if ((flags & FLG_OF_VERNEED) &&
4249                    (update_overneed(ofl) == S_ERROR))
4250                        return (S_ERROR);
4251                if (flags & (FLG_OF_VERNEED | FLG_OF_VERDEF))
4252                        update_oversym(ofl);
4253        }
4254        if (flags & FLG_OF_DYNAMIC) {
4255                if (update_odynamic(ofl) == S_ERROR)
4256                        return (S_ERROR);
4257        }
4258        if (ofl->ofl_ossyminfo) {
4259                if (update_osyminfo(ofl) == S_ERROR)
4260                        return (S_ERROR);
4261        }
4262
4263        /*
4264         * Update capabilities information if required.
4265         */
4266        if (ofl->ofl_oscap)
4267                update_oscap(ofl);
4268        if (ofl->ofl_oscapinfo)
4269                update_oscapinfo(ofl);
4270
4271        /*
4272         * Sanity test: the first and last data byte of a string table
4273         * must be NULL.
4274         */
4275        assert((ofl->ofl_osshstrtab == NULL) ||
4276            (*((char *)ofl->ofl_osshstrtab->os_outdata->d_buf) == '\0'));
4277        assert((ofl->ofl_osshstrtab == NULL) ||
4278            (*(((char *)ofl->ofl_osshstrtab->os_outdata->d_buf) +
4279            ofl->ofl_osshstrtab->os_outdata->d_size - 1) == '\0'));
```



```
4281        assert((ofl->ofl_osstrtab == NULL) ||
4282            (*((char *)ofl->ofl_osstrtab->os_outdata->d_buf) == '\0'));
4283        assert((ofl->ofl_osstrtab == NULL) ||
4284            (*(((char *)ofl->ofl_osstrtab->os_outdata->d_buf) +
4285            ofl->ofl_osstrtab->os_outdata->d_size - 1) == '\0'));
4286
4287        assert((ofl->ofl_osdynstr == NULL) ||
4288            (*((char *)ofl->ofl_osdynstr->os_outdata->d_buf) == '\0'));
4289        assert((ofl->ofl_osdynstr == NULL) ||
4290            (*(((char *)ofl->ofl_osdynstr->os_outdata->d_buf) +
4291            ofl->ofl_osdynstr->os_outdata->d_size - DYNSTR_EXTRA_PAD - 1) ==
4292            '\0'));
4293
4294        /*
4295         * Emit Strtab diagnostics.
4296         */
4297        DBG_CALL(Dbg_sec_strtab(ofl->ofl_lml, ofl->ofl_osshstrtab,
4298            ofl->ofl_shdrsttab));
4299        DBG_CALL(Dbg_sec_strtab(ofl->ofl_lml, ofl->ofl_osstrtab,
4300            ofl->ofl_strtab));
4301        DBG_CALL(Dbg_sec_strtab(ofl->ofl_lml, ofl->ofl_osdynstr,
4302            ofl->ofl_dynstrtab));
4303
4304        /*
4305         * Initialize the section headers string table index within the elf
4306         * header.
4307         */
4308        /* LINTED */
4309        if ((shscnndx = elf_ndxscn(ofl->ofl_osshstrtab->os_scn)) <
4310            SHN_LORESERVE) {
4311                ofl->ofl_nehdr->e_shstrndx =
4312                    /* LINTED */
4313                    (Half)shscnndx;
4314        } else {
4315                /*
4316                 * If the STRTAB section index doesn't fit into
4317                 * e_shstrndx, then we store it in 'shdr[0].st_link'.
4318                 */
4319                Elf_Scn *scn;
4320                Shdr    *shdr0;
4321
4322                if ((scn = elf_getscn(ofl->ofl_elf, 0)) == NULL) {
4323                        ld_eprintf(ofl, ERR_ELF, MSG_INTL(MSG_ELF_GETSCN),
4324                            ofl->ofl_name);
4325                        return (S_ERROR);
4326                }
4327                if ((shdr0 = elf_getshdr(scn)) == NULL) {
4328                        ld_eprintf(ofl, ERR_ELF, MSG_INTL(MSG_ELF_GETSHDR),
4329                            ofl->ofl_name);
4330                        return (S_ERROR);
4331                }
4332                ofl->ofl_nehdr->e_shstrndx = SHN_XINDEX;
4333                shdr0->sh_link = shscnndx;
4334        }
4335
4336        return ((uintptr_t)etext);
4337 }
```

```
**********************************************************
   88750 Sun Feb 24 19:19:14 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)
ld: implement -ztype and rework option parsing
**********************************************************
```

```
#
# 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
--------------------------------------------------------------------------------
4349563 auxiliary filter error handling regression introduced in 4165487
4355795 ldd -r now gives "displacement relocated" warnings
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-13
        Solaris/SunOS 5.7_x86           patch 106951-13
        Solaris/SunOS 5.6_sparc         patch 107733-09
        Solaris/SunOS 5.6_x86           patch 107734-09
--------------------------------------------------------------------------------
4210412 versioning a static executable causes ld to core dump
4219652 Linker gives misleading error about not finding main (xarch=v9)
4103449 ld command needs a command line flag to force 64-bits
4187211 problem with RDISP32 linking in copy-relocated objects
4287274 dladdr, dlinfo do not provide the full path name of a shared object
4297563 dlclose still does not remove all objects.
4250694 rtld_db needs a new auxvec entry
4235315 new features for rtld_db (DT_CHECKSUM, dynamic linked .o files
4303609 64bit libelf.so.1 does not properly implement elf_hash()
4310901 su.static fails when OSNet build with lazy-loading
4310324 elf_errno() causes Bus Error(coredump) in 64-bit multithreaded programs
4306415 ld core dump
4316531 BCP: possible failure with dlclose/_preexec_exit_handlers
4313765 LD_BREADTH should be shot
4318162 crle uses automatic strings in putenv.
4255943 Description of -t option incomplete.
4322528 sgs message test infrastucture needs improvement
4239213 Want an API to obtain linker's search path
4324134 use of extern mapfile directives can contribute unused symbols
4322581 ELF data structures could be layed out more efficiently...
4040628 Unnecessary section header symbols should be removed from .dynsym
4300018 rtld: bindlock should be freed before calling call_fini()
4336102 dlclose with non-deletable objects can mishandle dependencies
4329785 mixing of SHT_SUNW_COMDAT & SHF_ORDERED causes ld to seg fault
4334617 COPY relocations should be produces for references to .bss symbols
4248250 relcoation of local ABS symbols incorrect
4335801 For complimentary alignments eliminate ld: warning: symbol `ll'
        has differing a
4336980 ld.so.1 relative path processing revisited
4243097 dlerror(3DL) is not affected by setlocale(3C).
4344528 dump should remove -D and -l usage message
xxxxxxx enable LD_ALTEXEC to access alternate link-editor
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-06
        Solaris/SunOS 5.8_x86           patch 109148-06
--------------------------------------------------------------------------------

-----------------------------------
Solaris 8 101 (3rd Q-update - s28u3)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4346144 link-auditing: plt_tracing fails if LA_SYMB_NOPLTENTER given after
        being bound
4346001 The ld should support mapfile syntax to generate PT_SUNWSTACK segment
4349137 rtld_db: A third fallback method for locating the linkmap
4343417 dladdr interface information inadequate
4343801 RFE: crle(1): provide option for updating configuration files
4346615 ld.so.1 attempting to open a directory gives: No such device
4352233 crle should not honor umask
4352330 LD_PRELOAD cannot use absolute path for privileged program
4357805 RFE: man page for ld(1) does not document all -z or -B options in
        Solaris 8 9/00
4358751 ld.so.1: LD_XXX environ variables and LD_FLAGS should be synchronized.
```



```
4358862 link editors should reference "64" symlinks instead of sparcv9 (ia64).
4356879 PLTs could use faster code sequences in some cases
4367118 new fast baplt's fail when traversed twice in threaded application
4366905 Need a way to determine path to a shared library
4351197 nfs performance problem by 103627-13
4367405 LD_LIBRARY_PATH_64 not being used
4354500 SHF_ORDERED ordered scections does not properly sort sections
4369068 ld(1)'s weak symbol processing is inefficient (slow and doesn't scale).
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-07
        Solaris/SunOS 5.8_x86           patch 109148-07
        Solaris/SunOS 5.7_sparc         patch 106950-14
        Solaris/SunOS 5.7_x86           patch 106951-14
--------------------------------------------------------------------------------

-----------------------------------
Solaris 8 701 (5th Q-update - s28u5)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4368846 ld(1) fails to version some interfaces given in a mapfile
4077245 dump core dump on null pointer.
4372554 elfdump should demangle symbols (like nm, dump)
4371114 dlclose may unmap a promiscuous object while it's still in use.
4204447 elfdump should understand SHN_AFTER/SHN_BEGIN macro
4377941 initialization of interposers may not occur
4381116 ldd/ld.so.1 could aid in detecting unused dependencies
4381783 dlopen/dlclose of a libCrun+libthread can dump core
4385402 linker & run-time linker must support gABI ELF updates
4394698 ld.so.1 does not process DF_SYMBOLIC - not gABI conforming
4394212 the link editor quietly ignores missing support libraries
4390308 ld.so.1 should provide more flexibility LD_PRELOAD'ing 32-bit/64-bit
        objects
4401232 crle(1) could provide better flexibility for alternatives
4401815 fix misc nits in debugging output...
4402861 cleanup /usr/demo/link_audit & /usr/tmp/librtld_db demo source code...
4393044 elfdump should allow raw dumping of sections
4413168 SHF_ORDERED bit causes linker to generate a separate section
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-08
        Solaris/SunOS 5.8_x86           patch 109148-08
--------------------------------------------------------------------------------
4452202 Typos in <sys/link.h>
4452220 dump doesn't support RUNPATH
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-09
        Solaris/SunOS 5.8_x86           patch 109148-09
--------------------------------------------------------------------------------

-----------------------------------
Solaris 8 1001 (6th Q-update - s28u6)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4421842 fixups in SHT_GROUP processing required...
4450433 problem with libldddbg output on -Dsection,detail when
        processing SHF_LINK_ORDER
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-10
        Solaris/SunOS 5.8_x86           patch 109148-10
        Solaris/SunOS 5.7_sparc         patch 106950-15
        Solaris/SunOS 5.7_x86           patch 106951-15
```

```
389 ------------------------------------------------------------------------------
390 4463473 pldd showing wrong output
391 ------------------------------------------------------------------------------
392 All the above changes are incorporated in the following patches:
393         Solaris/SunOS 5.8_sparc         patch 109147-11
394         Solaris/SunOS 5.8_x86           patch 109148-11
395 ------------------------------------------------------------------------------

397 -----------------------------------
398 Solaris 8 202 (7th Q-update - s28u7)
399 -----------------------------------
400 Bugid   Risk Synopsis
401 ==============================================================================
402 4488954 ld.so.1 reuses same buffer to send ummapping range to
403         _preexec_exit_handlers()
404 ------------------------------------------------------------------------------
405 All the above changes are incorporated in the following patches:
406         Solaris/SunOS 5.8_sparc         patch 109147-12
407         Solaris/SunOS 5.8_x86           patch 109148-12
408 ------------------------------------------------------------------------------

410 ---------
411 Solaris 9
412 ---------
413 Bugid   Risk Synopsis
414 ==============================================================================
415 4505289 incorrect handling of _START_ and _END_
416 4506164 mcs does not recognize #linkbefore or #linkafter qualifiers
417 4447560 strip is creating unexecutable files...
418 4513842 library names not in ld.so string pool cause corefile bugs
419 ------------------------------------------------------------------------------
420 All the above changes are incorporated in the following patches:
421         Solaris/SunOS 5.8_sparc         patch 109147-13
422         Solaris/SunOS 5.8_x86           patch 109148-13
423         Solaris/SunOS 5.7_sparc         patch 106950-16
424         Solaris/SunOS 5.7_x86           patch 106951-16
425 ------------------------------------------------------------------------------
426 4291384 ld -M with a mapfile does not properly align Fortran REAL*8 data
427 4413322 SunOS 5.9 librtld_db doesn't show dlopened ".o" files anymore?
428 4429371 librtld_db busted on ia32 with SC6.x compilers...
429 4418274 elfdump dumps core on invalid input
430 4432224 libelf xlate routines are out of date
431 4433643 Memory leak using dlopen()/dlclose() in Solaris 8
432 4446564 ldd/lddstub - core dump conditions
433 4446115 translating SUNW_move sections is broken
434 4450225 The rdb command can fall into an infinite loop
435 4448531 Linker Causes Segmentation Fault
436 4453241 Regression in 4291384 can result in empty symbol table.
437 4453398 invalid runpath token can cause ld to spin.
438 4460230 ld (for OS 5.8 and 5.9) loses error message
439 4462245 ld.so.1 core dumps when executed directly...
440 4455802 need more flexibility in establishing a support library for ld
441 4467068 dyn_plt_entsize not properly initialized in ld.so.1
442 4468779 elf_plt_trace_write() broken on i386 (link-auditing)
443 4465871 -zld32 and -zld64 does not work the way it should
444 4461890 bad shared object created with -zredlocsym
445 4469400 ld.so.1: is_so_loaded isn't as efficient as we thought...
446 4469566 lazy loading fallback can reference un-relocated objects
447 4470493 libelf incorectly translates NOTE sections accross architectures...
448 4469684 rtld leaks dl_handles and permits on dlopen/dlclose
449 4475174 ld.so.1 prematurly reports the failure to load a object...
450 4475514 ld.so.1 can core dump in memory allocation fails (no swap)
451 4481851 Setting ld.so.1 environment variables globally would be useful
452 4482035 setting LD_PROFILE & LD_AUDIT causes ping command to issue warnings
453         on 5.8
454 4377735 segment reservations cause sbrk() to fail
```



```
455 4491434 ld.so.1 can leak file-descriptors when loading same named objects
456 4289232 some of warning/error/debugging messages from libld.so can be revised
457 4462748 Linker Portion of TLS Support
458 4496718 run-time linkers mutex_locks not working with ld_libc interface
459 4497270 The -zredlocsym option should not eliminate partially initialized local
460         symbols
461 4496963 dumping an object with crle(1) that uses $ORIGIN can loose its
462         dependencies
463 4499413 Sun linker orders of magnitude slower than gnu linker
464 4461760 lazy loading libXm and libXt can fail.
465 4469031 The partial initialized (local) symbols for intel platform is not
466         working.
467 4492883 Add link-editor option to multi-pass archives to resolve unsatisfied
468         symbols
469 4503731 linker-related commands misspell "argument"
470 4503768 whocalls(1) should output messages to stderr, not stdout
471 4503748 whocalls(1) usage message and manpage could be improved
472 4503625 nm should be taught about TLS symbols - that they aren't allowed that is
473 4300120 segment address validation is too simplistic to handle segment
474         reservations
475 4404547 krtld/reloc.h could have better error message, has typos
476 4270931 R_SPARC_HIX22 relocation is not handled properly
477 4485320 ld needs to support more the 32768 PLTs
478 4516434 sotruss can not watch libc_psr.so.1
479 4213100 sotruss could use more flexible pattern matching
480 4503457 ld seg fault with comdat
481 4510264 sections with SHF_TLS can come in different orders...
482 4518079 link-editor support library unable to modify section header flags
483 4515913 ld.so.1 can incorrectly decrement external reference counts on dlclose()
484 4519569 ld -V does not return a interesting value...
485 4524512 ld.so.1 should allow alternate termination signals
486 4524767 elfdump dies on bogus sh_name fields...
487 4524735 ld getopt processing of '-' changed
488 4521931 subroutine in a shared object as LOCL instead of GLOB
489 ------------------------------------------------------------------------------
490 All the above changes are incorporated in the following patches:
491         Solaris/SunOS 5.8_sparc         patch 109147-14
492         Solaris/SunOS 5.8_x86           patch 109148-14
493         Solaris/SunOS 5.7_sparc         patch 106950-17
494         Solaris/SunOS 5.7_x86           patch 106951-17
495 ------------------------------------------------------------------------------
496 4532729 tentative definition of TLS variable causes linker to dump core
497 4526745 fixup ld error message about duplicate dependencies/needed names
498 4522999 Solaris linker one order of magnitude slower than GNU linker
499 4518966 dldump undoes existing relocations with no thought of alignment or size.
500 4587441 Certain libraries have race conditions when setting error codes
501 4523798 linker option to align bss to large pagesize alignments.
502 4524008 ld can improperly set st_size of symbols named "_init" or "_fini"
503 4619282 ld cannot link a program with the option -sb
504 4620846 Perl Configure probing broken by ld changes
505 4621122 multiple ld '-zinitarray=' on a commandline fails
506 ------------------------------------------------------------------------------
507         Solaris/SunOS 5.8_sparc         patch 109147-15
508         Solaris/SunOS 5.8_x86           patch 109148-15
509         Solaris/SunOS 5.7_sparc         patch 106950-18
510         Solaris/SunOS 5.7_x86           patch 106951-18
511         Solaris/SunOS 5.6_sparc         patch 107733-10
512         Solaris/SunOS 5.6_x86           patch 107734-10
513 ------------------------------------------------------------------------------
514 All the above changes plus:
515         4616944 ar seg faults when order of object file is reversed.
516 are incorporated in the following patches:
517         Solaris/SunOS 5.8_sparc         patch 109147-16
518         Solaris/SunOS 5.8_x86           patch 109148-16
519 ------------------------------------------------------------------------------
520 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
```

```
785 --------------------------------------------------------------------------------

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 liblds tab: 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
 918 6175843 amd64 rdb_demo files not installed
 919 6182293 ld.so.1 can repeatedly relocate object .plts (RTLD_NOW).
 920 6183645 ld core dumps when automounter fails
 921 6178667 ldd list unexpected (file not found) in x86 environment.
 922 6181928 Need new reloc types R_AMD64_GOTOFF64 and R_AMD64_GOTPC32
 923 6182884 AMD64: ld coredumps when building a shared library
 924 6173559 The ld may set incorrect value for sh_addralign under some conditions.
 925 5105601 ld.so.1 gets a little too enthusiastic with interposition
 926 6189384 ld.so.1 should accommodate a files dev/inode change (libc loopback mnt)
 927 6177838 AMD64: linker cannot resolve PLT for 32-bit a.out(s) on amd64-S2 kernel
 928 6190863 sparc disassembly code should be removed from rdb_demo
 929 6191488 unwind eh_frame_hdr needs corrected encoding value
 930 6192490 moe(1) returns /lib/libc.so.1 for optimal expansion of libc HWCAP
 931         libraries
 932 6192164 AMD64: introduce dlamd64getunwind interface
 933         PSARC/2004/747 libc::dlamd64getunwind()
 934 6195030 libdl has bad version name
 935 6195521 64-bit moe(1) missed the train
 936 6198358 AMD64: bad eh_frame_hdr data when C and C++ mixed in a.out
 937 6204123 ld.so.1: symbol lookup fails even after lazy loading fallback
 938 6207495 UNIX98/UNIX03 vsx namespace violation DYNL.hdr/misc/dlfcn/T.dlfcn
 939         14 Failed
 940 6217285 ctfmerge crashed during full onnv build
 941 --------------------------------------------------------------------------------

 943 ------------------------------------
 944 Solaris 10 106 (1st Q-update - s10u1)
 945 ------------------------------------
 946 Bugid   Risk Synopsis
 947 ================================================================================
 948 6209350 Do not include signature section from dynamic dependency library into
 949         relocatable object
 950 6212797 The binary compiled on SunOS4.x doesn't run on Solaris8 with Patch
 951         109147-31
 952 --------------------------------------------------------------------------------
 953 All the above changes are incorporated in the following patches:
 954         Solaris/SunOS 5.9_sparc         patch T112963-18
 955         Solaris/SunOS 5.9_x86           patch T113986-14
 956         Solaris/SunOS 5.8_sparc         patch T109147-33
 957         Solaris/SunOS 5.8_x86           patch T109148-33
 958 --------------------------------------------------------------------------------
 959 6219538 112963-17: linker patch causes binary to dump core
 960 --------------------------------------------------------------------------------
 961 All the above changes are incorporated in the following patches:
 962         Solaris/SunOS 5.10_sparc        patch T117461-01
 963         Solaris/SunOS 5.10_x86          patch T118345-01
 964         Solaris/SunOS 5.9_sparc         patch T112963-19
 965         Solaris/SunOS 5.9_x86           patch T113986-15
 966         Solaris/SunOS 5.8_sparc         patch T109147-34
 967         Solaris/SunOS 5.8_x86           patch T109148-34
 968 --------------------------------------------------------------------------------
 969 6257177 incremental builds of usr/src/cmd/sgs can fail...
 970 6219651 AMD64: Linker does not issue error for out of range R_AMD64_PC32
 971 --------------------------------------------------------------------------------
 972 All the above changes are incorporated in the following patches:
 973         Solaris/SunOS 5.10_sparc        patch T117461-02
 974         Solaris/SunOS 5.10_x86          patch T118345-02
 975         Solaris/SunOS 5.9_sparc         patch T112963-20
 976         Solaris/SunOS 5.9_x86           patch T113986-16
 977         Solaris/SunOS 5.8_sparc         patch T109147-35
 978         Solaris/SunOS 5.8_x86           patch T109148-35
 979 NOTE: The fix for 6219651 is only applicable for 5.10_x86 platform.
 980 --------------------------------------------------------------------------------
 981 5080443 lazy loading failure doesn't clean up after itself (D)
 982 6226206 ld.so.1 failure when processing single segment hwcap filtee
```



```
 983 6228472 ld.so.1: link-map control list stacking can loose objects
 984 6235000 random packages not getting installed in snv_09 and snv_10 -
 985         rtld/common/malloc.c Assertion
 986 6219317 Large page support is needed for mapping executables, libraries and
 987         files (link-editor components only)
 988 6244897 ld.so.1 can't run apps from commandline
 989 6251798 moe(1) returns an internal assertion failure message in some
 990         circumstances
 991 6251722 ld fails silently with exit 1 status when -z ignore passed
 992 6254364 ld won't build libgenunix.so with absolute relocations
 993 6215444 ld.so.1 caches "not there" lazy libraries, foils svc.startd(1M)'s logic
 994 6222525 dlsym(3C) trusts caller(), which may return wrong results with tail call
 995         optimization
 996 6241995 warnings in sgs should be fixed (link-editor components only)
 997 6258834 direct binding availability should be verified at runtime
 998 6260361 lari shouldn't count a.out non-zero undefined entries as interesting
 999 6260780 ldd doesn't recognize LD_NOAUXFLTR
1000 6266261 Add ld(1) -Bnodirect support (D)
1001 6261990 invalid e_flags error could be a little more friendly
1002 6261803 lari(1) should find more events uninteresting (D)
1003 6267352 libld_malloc provides inadequate alignment
1004 6268693 SHN_SUNW_IGNORE symbols should be allowed to be mulitiply defined
1005 6262789 Infosys wants a faster linker
1006 --------------------------------------------------------------------------------
1007 All the above changes are incorporated in the following patches:
1008         Solaris/SunOS 5.10_sparc        patch T117461-03
1009         Solaris/SunOS 5.10_x86          patch T118345-03
1010         Solaris/SunOS 5.9_sparc         patch T112963-21
1011         Solaris/SunOS 5.9_x86           patch T113986-17
1012         Solaris/SunOS 5.8_sparc         patch T109147-36
1013         Solaris/SunOS 5.8_x86           patch T109148-36
1014 --------------------------------------------------------------------------------
1015 6283601 The usr/src/cmd/sgs/packages/common/copyright contains old information
1016         legally problematic
1017 6276905 dlinfo gives inconsistent results (relative vs absolute linkname) (D)
1018         PSARC/2005/357 dlinfo(3c) RTLD_DI_ARGSINFO
1019 6284941 excessive link times with many groups/sections
1020 6280467 dlclose() unmaps shared library before library's _fini() has finished
1021 6291547 ld.so mishandles LD_AUDIT causing security problems.
1022 --------------------------------------------------------------------------------
1023 All the above changes are incorporated in the following patches:
1024         Solaris/SunOS 5.10_sparc        patch T117461-04
1025         Solaris/SunOS 5.10_x86          patch T118345-04
1026         Solaris/SunOS 5.9_sparc         patch T112963-22
1027         Solaris/SunOS 5.9_x86           patch T113986-18
1028         Solaris/SunOS 5.8_sparc         patch T109147-37
1029         Solaris/SunOS 5.8_x86           patch T109148-37
1030 --------------------------------------------------------------------------------
1031 6295971 UNIX98/UNIX03 *vsx* DYNL.hdr/misc/dlfcn/T.dlfcn 14 fails, auxv.h syntax
1032         error
1033 6299525 .init order failure when processing cycles
1034 6273855 gcc and sgs/crle don't get along
1035 6273864 gcc and sgs/libld don't get along
1036 6273875 gcc and sgs/rtld don't get along
1037 6272563 gcc and amd64/krtld/doreloc.c don't get along
1038 6290157 gcc and sgs/librtld_db/rdb_demo don't get along
1039 6301218 Matlab dumps core on startup when running on 112963-22 (D)
1040 --------------------------------------------------------------------------------
1041 All the above changes are incorporated in the following patches:
1042         Solaris/SunOS 5.10_sparc        patch T117461-06
1043         Solaris/SunOS 5.10_x86          patch T118345-08
1044         Solaris/SunOS 5.9_sparc         patch T112963-23
1045         Solaris/SunOS 5.9_x86           patch T113986-19
1046         Solaris/SunOS 5.8_sparc         patch T109147-38
1047         Solaris/SunOS 5.8_x86           patch T109148-38
1048 --------------------------------------------------------------------------------
```

```
6314115 Checkpoint refuses to start, crashes on start, after application of
        linker patch 112963-22
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-24
        Solaris/SunOS 5.9_x86           patch T113986-20
        Solaris/SunOS 5.8_sparc         patch T109147-39
        Solaris/SunOS 5.8_x86           patch T109148-39
--------------------------------------------------------------------------------
6318306 a dlsym() from a filter should be redirected to an associated filtee
6318401 mis-aligned TLS variable
6324019 ld.so.1: malloc alignment is insufficient for new compilers
6324589 psh coredumps on x86 machines on snv_23
6236594 AMD64: Linker needs to handle the new .lbss section (D)
        PSARC 2005/514 AMD64 - large section support
6314743 Linker: incorrect resolution for R_AMD64_GOTPC32
6311865 Linker: x86 medium model; invalid ELF program header
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T117461-07
        Solaris/SunOS 5.10_x86          patch T118345-12
--------------------------------------------------------------------------------
6309061 link_audit should use __asm__ with gcc
6310736 gcc and sgs/libld don't get along on SPARC
6329796 Memory leak with iconv_open/iconv_close with patch 109147-33
6332983 s9 linker patches 112963-24/113986-20 causing cluster machines not
        to boot
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T117461-08
        Solaris/SunOS 5.10_x86          patch T121208-02
        Solaris/SunOS 5.9_sparc         patch T112963-25
        Solaris/SunOS 5.9_x86           patch T113986-21
        Solaris/SunOS 5.8_sparc         patch T109147-40
        Solaris/SunOS 5.8_x86           patch T109148-40
--------------------------------------------------------------------------------
6445311 The sparc S8/S9/S10 linker patches which include the fix for the
        CR6222525 are hit by the CR6439613.
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-26
        Solaris/SunOS 5.8_sparc         patch T109147-41
--------------------------------------------------------------------------------

-------------------------------------
Solaris 10 807 (4th Q-update - s10u4)
-------------------------------------
Bugid   Risk Synopsis
================================================================================
6487273 ld.so.1 may open arbitrary locale files when relative path is built
        from locale environment vars
6487284 ld.so.1: buffer overflow in doprf() function
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T124922-01
        Solaris/SunOS 5.10_x86          patch T124923-01
        Solaris/SunOS 5.9_sparc         patch T112963-27
        Solaris/SunOS 5.9_x86           patch T113986-22
        Solaris/SunOS 5.8_sparc         patch T109147-42
        Solaris/SunOS 5.8_x86           patch T109148-41
--------------------------------------------------------------------------------
6477132 ld.so.1: memory leak when running set*id application
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T124922-02
        Solaris/SunOS 5.10_x86          patch T124923-02
```



```
        Solaris/SunOS 5.9_sparc         patch T112963-30
        Solaris/SunOS 5.9_x86           patch T113986-24
--------------------------------------------------------------------------------
6340814 ld.so.1 core dump with HWCAP relocatable object + updated statistics
6307274 crle bug with LD_LIBRARY_PATH
6317969 elfheader limited to 65535 segments (link-editor components only)
6350027 ld.so.1 aborts with assertion failed on amd64
6362044 ld(1) inconsistencies with LD_DEBUG=-Dunused and -zignore
6362047 ld.so.1 dumps core when combining HWCAP and LD_PROFILE
6304206 runtime linker may respect LANG and LC_MESSAGE more than LC_ALL
6363495 Catchup required with Intel relocations
6326497 ld.so not properly processing LD_LIBRARY_PATH ending in :
6307146 mcs dumps core when appending null string to comment section
6371877 LD_PROFILE_64 with gprof does not produce correct results on amd64
6372082 ld -r erroneously creates .got section on i386
6201866 amd64: linker symbol elimination is broken
6372620 printstack() segfaults when called from static function (D)
6380470 32-bit ld(1) incorrectly builds 64-bit relocatable objects
6391407 Insufficient alignment of 32-bit object in archive makes ld segfault
        (libelf component only) (D)
6316708 LD_DEBUG should provide a means of identifying/isolating individual
        link-map lists (P)
6280209 elfdump cores on memory model 0x3
6197234 elfdump and dump don't handle 64-bit symbols correctly
6398893 Extended section processing needs some work
6397256 ldd dumps core in elf_fix_name
6327926 ld does not set etext symbol correctly for AMD64 medium model (D)
6390410 64-bit LD_PROFILE can fail: relocation error when binding profile plt
6382945 AMD64-GCC: dbx: internal error: dwarf reference attribute out of bounds
6262333 init section of .so dlopened from audit interface not being called
6409613 elf_outsync() should fsync()
6426048 C++ exceptions broken in Nevada for amd64
6429418 ld.so.1: need work-around for Nvidia drivers use of static TLS
6429504 crle(1) shows wrong defaults for non-existent 64-bit config file
6431835 data corruption on x64 in 64-bit mode while LD_PROFILE is in effect
6423051 static TLS support within the link-editors needs a major face lift (D)
6388946 attempting to dlopen a .o file mislabeled as .so fails
6446740 allow mapfile symbol definitions to create backing storage (D)
4986360 linker crash on exec of .so (as opposed to a.out) -- error preferred
        instead
6229145 ld: initarray/finiarray processing occurs after got size is determined
6324924 the linker should warn if there's a .init section but not _init
6424132 elfdump inserts extra whitespace in bitmap value display
6449485 ld(1) creates misaligned TLS in binary compiled with -xpg
6424550 Write to unallocated (wua) errors when libraries are built with
        -z lazyload
6464235 executing the 64-bit ld(1) should be easy (D)
6465623 need a way of building unix without an interpreter
6467925 ld: section deletion (-z ignore) requires improvement
6357230 specfiles should be nuked (link-editor components only)
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T124922-03
        Solaris/SunOS 5.10_x86          patch T124923-03

These patches also include the framework changes for the following bug fixes.
However, the associated feature has not been enabled in Solaris 10 or earlier
releases:

6174390 crle configuration files are inconsistent across platforms (D, P)
6432984 ld(1) output file removal - change default behavior (D)
        PSARC/2006/353 ld(1) output file removal - change default behavior
--------------------------------------------------------------------------------

-------------------------------------
Solaris 10 508 (5th Q-update - s10u5)
```

```
1181 ------------------------------------
1182 Bugid   Risk Synopsis
1183 ================================================================================
1184 6561987 data vac_conflict faults on lipthread libthread libs in s10.
1185 --------------------------------------------------------------------------------
1186 All the above changes are incorporated in the following patches:
1187         Solaris/SunOS 5.10_sparc        patch T127111-01
1188         Solaris/SunOS 5.10_x86          patch T127112-01
1189 --------------------------------------------------------------------------------
1190 6501793 GOTOP relocation transition (optimization) fails with offsets > 2^32
1191 6532924 AMD64: Solaris 5.11 55b: SEGV after whocatches
1192 6551627 OGL: SIGSEGV when trying to use OpenGL pipeline with splash screen,
1193         Solaris/Nvidia only
1194 --------------------------------------------------------------------------------
1195 All the above changes are incorporated in the following patches:
1196         Solaris/SunOS 5.10_sparc        patch T127111-04
1197         Solaris/SunOS 5.10_x86          patch T127112-04
1198 --------------------------------------------------------------------------------
1199 6479848 Enhancements to the linker support interface needed. (D)
1200         PSARC/2006/595 link-editor support library interface - ld_open()
1201 6521608 assertion failure in runtime linker related to auditing
1202 6494228 pclose() error when an audit library calls popen() and the main target
1203         is being run under ldd (D)
1204 6568745 segfault when using LD_DEBUG with bit_audit library when instrumenting
1205         mozilla (D)
1206         PSARC/2007/413 Add -zglobalaudit option to ld
1207 6602294 ps_pbrandname breaks apps linked directly against librtld_db
1208 --------------------------------------------------------------------------------
1209 All the above changes are incorporated in the following patches:
1210         Solaris/SunOS 5.10_sparc        patch T127111-07
1211         Solaris/SunOS 5.10_x86          patch T127112-07
1212 --------------------------------------------------------------------------------

1214 ------------------------------------
1215 Solaris 10 908 (6th Q-update - s10u6)
1216 ------------------------------------
1217 Bugid   Risk Synopsis
1218 ================================================================================
1219 6672544 elf_rtbndr must support non-ABI aligned stacks on amd64
1220 6668050 First trip through PLT does not preserve args in xmm registers
1221 --------------------------------------------------------------------------------
1222 All the above changes are incorporated in the following patch:
1223         Solaris/SunOS 5.10_x86          patch T137138-01
1224 --------------------------------------------------------------------------------

1226 ------------------------------------
1227 Solaris 10 409 (7th Q-update - s10u7)
1228 ------------------------------------
1229 Bugid   Risk Synopsis
1230 ================================================================================
1231 6629404 ld with -z ignore doesn't scale
1232 6606203 link editor ought to allow creation of >2gb sized objects (P)
1233 --------------------------------------------------------------------------------
1234 All the above changes are incorporated in the following patches:
1235         Solaris/SunOS 5.10_sparc        patch T139574-01
1236         Solaris/SunOS 5.10_x86          patch T139575-01
1237 --------------------------------------------------------------------------------
1238 6746674 setuid applications do not find libraries any more because trusted
1239         directories behavior changed (D)
1240 --------------------------------------------------------------------------------
1241 All the above changes are incorporated in the following patches:
1242         Solaris/SunOS 5.10_sparc        patch T139574-02
1243         Solaris/SunOS 5.10_x86          patch T139575-02
1244 --------------------------------------------------------------------------------
1245 6703683 Can't build VirtualBox on Build 88 or 89
1246 6737579 process_req_lib() in libld consumes file descriptors
```



```
1247 6685125 ld/elfdump do not handle ZERO terminator .eh_frame amd64 unwind entry
1248 --------------------------------------------------------------------------------
1249 All the above changes are incorporated in the following patches:
1250         Solaris/SunOS 5.10_sparc        patch T139574-03
1251         Solaris/SunOS 5.10_x86          patch T139575-03
1252 --------------------------------------------------------------------------------

1254 ------------------------------------
1255 Solaris 10 1009 (8th Q-update - s10u8)
1256 ------------------------------------
1257 Bugid   Risk Synopsis
1258 ================================================================================
1259 6782597 32-bit ld.so.1 needs to accept objects with large inode number
1260 6805502 The addition of "inline" keywords to sgs code broke the lint
1261         verification in S10
1262 6807864 ld.so.1 is susceptible to a fatal dlsym()/setlocale() race
1263 --------------------------------------------------------------------------------
1264 All the above changes are incorporated in the following patches:
1265         Solaris/SunOS 5.10_sparc        patch T141692-01
1266         Solaris/SunOS 5.10_x86          patch T141693-01
1267 NOTE: The fix for 6805502 is only applicable to s10.
1268 --------------------------------------------------------------------------------
1269 6826410 ld needs to sort sections using 32-bit sort keys
1270 --------------------------------------------------------------------------------
1271 All the above changes are incorporated in the following patches:
1272         Solaris/SunOS 5.10_sparc        patch T141771-01
1273         Solaris/SunOS 5.10_x86          patch T141772-01
1274 NOTE: The fix for 6826410 is also available for s9 in the following patches:
1275         Solaris/SunOS 5.9_sparc         patch T112963-33
1276         Solaris/SunOS 5.9_x86           patch T113986-27
1277 --------------------------------------------------------------------------------
1278 6568447 bcp is broken by 6551627
1279 6599700 librtld_db needs better plugin support
1280 6713830 mdb dumped core reading a gcore
1281 6756048 rd_loadobj_iter() should always invoke brand plugin callback
1282 6786744 32-bit dbx failed with unknown rtld_db.so error on snv_104
1283 --------------------------------------------------------------------------------
1284 All the above changes are incorporated in the following patches:
1285         Solaris/SunOS 5.10_sparc        patch T141444-06
1286         Solaris/SunOS 5.10_x86          patch T141445-06
1287 --------------------------------------------------------------------------------

1289 ------------------------------------
1290 Solaris 10 1005 (9th Q-update - s10u9)
1291 ------------------------------------
1292 Bugid   Risk Synopsis
1293 ================================================================================
1294 6850124 dlopen reports "No such file or directory" in spite of ENOMEM
1295         when mmap fails in anon_map()
1296 6826513 ldd gets confused by a crle(1) LD_PRELOAD setting
1297 6684577 ld should propagate SHF_LINK_ORDER flag to ET_REL objects
1298 6524709 executables using /usr/lib/libc.so.1 as the ELF interpreter dump core
1299         (link-editor components only)
1300 --------------------------------------------------------------------------------
1301 All the above changes are incorporated in the following patches:
1302         Solaris/SunOS 5.10_sparc        patch T143895-01
1303         Solaris/SunOS 5.10_x86          patch T143896-01
1304 --------------------------------------------------------------------------------

1306 ------------------------------------
1307 Solaris 10 XXXX (10th Q-update - s10u10)
1308 ------------------------------------
1309 Bugid   Risk Synopsis
1310 ================================================================================
1311 6478684 isainfo/cpuid reports pause instruction not supported on amd64
1312         PSARC/2010/089 Removal of AV_386_PAUSE and AV_386_MON
```

```
1313 --------------------------------------------------------------------------------
1314 All the above changes are incorporated in the following patches:
1315         Solaris/SunOS 5.10_sparc        patch TXXXXXX-XX
1316         Solaris/SunOS 5.10_x86          patch TXXXXXX-XX
1317 --------------------------------------------------------------------------------

1319 ------------------------------------------
1320 Solaris Nevada (OpenSolaris 2008.05, snv_86)
1321 ------------------------------------------
1322 Bugid   Risk Synopsis
1323 ================================================================================
1324 6409350 BrandZ project integration into Solaris (link-editor components only)
1325 6459189 UNIX03: *VSC* c99 compiler overwrites non-writable file
1326 6423746 add an option to relax the resolution of COMDAT relocs (D)
1327 4934427 runtime linker should load up static symbol names visible to
1328         dladdr() (D)
1329         PSARC 2006/526 SHT_SUNW_LDYNSYM - default local symbol addition
1330 6448719 sys/elf.h could be updated with additional machine and ABI types
1331 6336605 link-editors need to support R_*_SIZE relocations
1332         PSARC/2006/558 R_*_SIZE relocation support
1333 6475375 symbol search optimization to reduce rescans
1334 6475497 elfdump(1) is misreporting sh_link
1335 6482058 lari(1) could be faster, and handle per-symbol filters better
1336 6482974 defining virtual address of text segment can result in an invalid data
1337         segment
1338 6476734 crle(1m) "-l" as described fails system, crle cores trying to fix
1339         /a/var/ld/ld.config in failsafe
1340 6487499 link_audit "make clobber" creates and populates proto area
1341 6488141 ld(1) should detect attempt to reference 0-length .bss section
1342 6496718 restricted visibility symbol references should trigger archive
1343         extraction
1344 6515970 HWCAP processing doesn't clean up fmap structure - browser fails to
1345         run java applet
1346 6494214 Refinements to symbolic binding, symbol declarations and
1347         interposition (D)
1348         PSARC/2006/714 ld(1) mapfile: symbol interpose definition
1349 6475344 DTrace needs ELF function and data symbols sorted by address (D)
1350         PSARC/2007/026 ELF symbol sort sections
1351 6518480 ld -melf_i386 doesn't complain (D)
1352 6519951 bfu is just another word for exit today (RPATH -> RUNPATH conversion
1353         bites us) (D)
1354 6521504 ld: hardware capabilities processing from relocatables objects needs
1355         hardening.
1356 6518322 Some ELF utilities need updating for .SUNW_ldynsym section (D)
1357         PSARC/2007/074 -L option for nm(1) to display SHT_SUNW_LDYNSYM symbols
1358 6523787 dlopen() handle gets mistakenly orphaned - results in access to freed
1359         memory
1360 6531189 SEGV in dladdr()
1361 6527318 dlopen(name, RTLD_NOLOAD) returns handle for unloaded library
1362 6518359 extern mapfiles references to _init/_fini can create INIT/FINI
1363         addresses of 0
1364 6533587 ld.so.1: init/fini processing needs to compensate for interposer
1365         expectations
1366 6516118 Reserved space needed in ELF dynamic section and string table (D)
1367         PSARC/2007/127 Reserved space for editing ELF dynamic sections
1368 6535688 elfdump could be more robust in the face of Purify (D)
1369 6516665 The link-editors should be more resilient against gcc's symbol
1370         versioning
1371 6541004 hwcap filter processing can leak memory
1372 5108874 elfdump SEGVs on bad object file
1373 6547441 Uninitialized variable causes ld.so.1 to crash on object cleanup
1374 6341667 elfdump should check alignments of ELF header elements
1375 6387860 elfdump cores, when processing linux built ELF file
1376 6198202 mcs -d dumps core
1377 6246083 elfdump should allow section index specification
1378         (numeric -N equivalent) (D)
```



```
1379         PSARC/2007/247 Add -I option to elfdump
1380 6556563 elfdump section overlap checking is too slow for large files
1381 5006034 need ?E mapfile feature extension (D)
1382 6565476 rtld symbol version check prevents GNU ld binary from running
1383 6567670 ld(1) symbol size/section size verification uncovers Haskell
1384         compiler inconsistency
1385 6530249 elfdump should handle ELF files with no section header table (D)
1386         PSARC/2007/395 Add -P option to elfdump
1387 6573641 ld.so.1 does not maintain parent relationship to a dlopen() caller.
1388 6577462 Additional improvements needed to handling of gcc's symbol versioning
1389 6583742 ELF string conversion library needs to lose static writable buffers
1390 6589819 ld generated reference to __tls_get_addr() fails when resolving to a
1391         shared object reference
1392 6595139 various applications should export yy* global variables for libl
1393         PSARC/2007/474 new ldd(1) -w option
1394 6597841 gelf_getdyn() reads one too many dynamic entries
1395 6603313 dlclose() can fail to unload objects after fix for 6573641
1396 6234471 need a way to edit ELF objects (D)
1397         PSARC/2007/509 elfedit
1398 5035454 mixing -Kpic and -KPIC may cause SIGSEGV with -xarch=v9
1399 6473571 strip and mcs get confused and corrupt files when passed
1400         non-ELF arguments
1401 6253589 mcs has problems handling multiple SHT_NOTE sections
1402 6610591 do_reloc() should not require unused arguments
1403 6602451 new symbol visibilities required: EXPORTED, SINGLETON and ELIMINATE (D)
1404         PSARC/2007/559 new symbol visibilities - EXPORTED, SINGLETON, and
1405         ELIMINATE
1406 6570616 elfdump should display incorrectly aligned note section
1407 6614968 elfedit needs string table module (D)
1408 6620533 HWCAP filtering can leave uninitialized data behind - results in
1409         "rejected: Invalid argument"
1410 6617855 nodirect tag can be ignored when other syminfo tags are available
1411         (link-editor components only)
1412 6621066 Reduce need for new elfdump options with every section type (D)
1413         PSARC/2007/620 elfdump -T, and simplified matching
1414 6627765 soffice failure after integration of 6603313 - dangling GROUP pointer.
1415 6319025 SUNWbtool packaging issues in Nevada and S10u1.
1416 6626135 elfedit capabilities str->value mapping should come from
1417         usr/src/common/elfcap
1418 6642769 ld(1) -z combreloc should become default behavior (D)
1419         PSARC/2008/006 make ld(1) -z combreloc become default behavior
1420 6634436 XFFLAG should be updated.  (link-editor components only)
1421 6492726 Merge SHF_MERGE|SHF_STRINGS input sections (D)
1422 4947191 OSNet should use direct bindings  (link-editor components only)
1423 6654381 lazy loading fall-back needs optimizing
1424 6658385 ld core dumps when building Xorg on nv_82
1425 6516808 ld.so.1's token expansion provides no escape for platforms that don't
1426         report HWCAP
1427 6668534 Direct bindings can compromise function address comparisons from
1428         executables
1429 6667661 Direct bindings can compromise executables with insufficient copy
1430         relocation information
1431 6357282 ldd should recognize PARENT and EXTERN symbols (D)
1432         PSARC/2008/148 new ldd(1) -p option
1433 6672394 ldd(1) unused dependency processing is tricked by relocations errors
1434 --------------------------------------------------------------------------------

1436 ------------------------------------------
1437 Solaris Nevada (OpenSolaris 2008.11, snv_101)
1438 ------------------------------------------
1439 Bugid   Risk Synopsis
1440 ================================================================================
1441 6671255 link-editor should support cross linking (D)
1442         PSARC/2008/179 cross link-editor
1443 6674666 elfedit dyn:posflag1 needs option to locate element via NEEDED item
1444 6675591 elfwrap - wrap data in an ELF file (D,P)
```

```
1445         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 XXXXX   support -ztype
1666 10366   ld(1) should support GNU-style linker sets
1667 #endif /* ! codereview */
```

```
**********************************************************
   11088 Sun Feb 24 19:19:15 2019
new/usr/src/cmd/sgs/rtld/common/globals.c
ld should reject kernel modules as input
**********************************************************
______unchanged_portion_omitted_

 200 Dl_argsinfo     argsinfo = { 0 };       /* process argument, environment and */
 201                                         /*      auxv information. */

 203 /*
 204  * Frequently used messages are cached here to reduce _dgettext() overhead and
 205  * also provide for resetting should the locale change (see _ld_libc()).
 206  */
 207 const char      *err_strs[ERR_NUM] = { NULL };
 208 const char      *nosym_str = NULL;


 211 /*
 212  * Rejection error message tables.
 213  */
 214 const Msg
 215 ldd_reject[SGS_REJ_NUM] = {
 216                 MSG_STR_EMPTY,
 217                 MSG_LDD_REJ_MACH,       /* MSG_INTL(MSG_LDD_REJ_MACH) */
 218                 MSG_LDD_REJ_CLASS,      /* MSG_INTL(MSG_LDD_REJ_CLASS) */
 219                 MSG_LDD_REJ_DATA,       /* MSG_INTL(MSG_LDD_REJ_DATA) */
 220                 MSG_LDD_REJ_TYPE,       /* MSG_INTL(MSG_LDD_REJ_TYPE) */
 221                 MSG_LDD_REJ_BADFLAG,    /* MSG_INTL(MSG_LDD_REJ_BADFLAG) */
 222                 MSG_LDD_REJ_MISFLAG,    /* MSG_INTL(MSG_LDD_REJ_MISFLAG) */
 223                 MSG_LDD_REJ_VERSION,    /* MSG_INTL(MSG_LDD_REJ_VERSION) */
 224                 MSG_LDD_REJ_HAL,        /* MSG_INTL(MSG_LDD_REJ_HAL) */
 225                 MSG_LDD_REJ_US3,        /* MSG_INTL(MSG_LDD_REJ_US3) */
 226                 MSG_LDD_REJ_STR,        /* MSG_INTL(MSG_LDD_REJ_STR) */
 227                 MSG_LDD_REJ_UNKFILE,    /* MSG_INTL(MSG_LDD_REJ_UNKFILE) */
 228                 MSG_LDD_REJ_UNKCAP,     /* MSG_INTL(MSG_LDD_REJ_UNKCAP) */
 229                 MSG_LDD_REJ_HWCAP_1,    /* MSG_INTL(MSG_LDD_REJ_HWCAP_1) */
 230                 MSG_LDD_REJ_SFCAP_1,    /* MSG_INTL(MSG_LDD_REJ_SFCAP_1) */
 231                 MSG_LDD_REJ_MACHCAP,    /* MSG_INTL(MSG_LDD_REJ_MACHCAP) */
 232                 MSG_LDD_REJ_PLATCAP,    /* MSG_INTL(MSG_LDD_REJ_PLATCAP) */
 233                 MSG_LDD_REJ_HWCAP_2,    /* MSG_INTL(MSG_LDD_REJ_HWCAP_2) */
 234                 MSG_LDD_REJ_ARCHIVE,    /* MSG_INTL(MSG_LDD_REJ_ARCHIVE) */
 235                 MSG_LDD_REJ_KMOD        /* MSG_INTL(MSG_LDD_REJ_KMOD) */
 234                 MSG_LDD_REJ_ARCHIVE     /* MSG_INTL(MSG_LDD_REJ_ARCHIVE) */
 236         };
 237 #if SGS_REJ_NUM != (SGS_REJ_KMOD + 1)
 236 #if SGS_REJ_NUM != (SGS_REJ_ARCHIVE + 1)
 238 #error SGS_REJ_NUM has changed
 239 #endif

 241 const Msg
 242 err_reject[SGS_REJ_NUM] = {
 243                 MSG_STR_EMPTY,
 244                 MSG_ERR_REJ_MACH,       /* MSG_INTL(MSG_ERR_REJ_MACH) */
 245                 MSG_ERR_REJ_CLASS,      /* MSG_INTL(MSG_ERR_REJ_CLASS) */
 246                 MSG_ERR_REJ_DATA,       /* MSG_INTL(MSG_ERR_REJ_DATA) */
 247                 MSG_ERR_REJ_TYPE,       /* MSG_INTL(MSG_ERR_REJ_TYPE) */
 248                 MSG_ERR_REJ_BADFLAG,    /* MSG_INTL(MSG_ERR_REJ_BADFLAG) */
 249                 MSG_ERR_REJ_MISFLAG,    /* MSG_INTL(MSG_ERR_REJ_MISFLAG) */
 250                 MSG_ERR_REJ_VERSION,    /* MSG_INTL(MSG_ERR_REJ_VERSION) */
 251                 MSG_ERR_REJ_HAL,        /* MSG_INTL(MSG_ERR_REJ_HAL) */
 252                 MSG_ERR_REJ_US3,        /* MSG_INTL(MSG_ERR_REJ_US3) */
 253                 MSG_ERR_REJ_STR,        /* MSG_INTL(MSG_ERR_REJ_STR) */
 254                 MSG_ERR_REJ_UNKFILE,    /* MSG_INTL(MSG_ERR_REJ_UNKFILE) */
 255                 MSG_ERR_REJ_UNKCAP,     /* MSG_INTL(MSG_ERR_REJ_UNKCAP) */
 256                 MSG_ERR_REJ_HWCAP_1,    /* MSG_INTL(MSG_ERR_REJ_HWCAP_1) */
```



```
 257                 MSG_ERR_REJ_SFCAP_1,    /* MSG_INTL(MSG_ERR_REJ_SFCAP_1) */
 258                 MSG_ERR_REJ_MACHCAP,    /* MSG_INTL(MSG_ERR_REJ_MACHCAP) */
 259                 MSG_ERR_REJ_PLATCAP,    /* MSG_INTL(MSG_ERR_REJ_PLATCAP) */
 260                 MSG_ERR_REJ_HWCAP_2,    /* MSG_INTL(MSG_ERR_REJ_HWCAP_2) */
 261                 MSG_ERR_REJ_ARCHIVE,    /* MSG_INTL(MSG_ERR_REJ_ARCHIVE) */
 262                 MSG_ERR_REJ_KMOD,       /* MSG_INTL(MSG_ERR_REJ_KMOD) */
 263 #endif /* ! codereview */
 264         };
 265 #if SGS_REJ_NUM != (SGS_REJ_KMOD + 1)
 261 #if SGS_REJ_NUM != (SGS_REJ_ARCHIVE + 1)
 266 #error SGS_REJ_NUM has changed
 267 #endif

 269 const Msg
 270 ldd_warn[SGS_REJ_NUM] = {
 271                 MSG_STR_EMPTY,
 272                 MSG_STR_EMPTY,
 273                 MSG_STR_EMPTY,
 274                 MSG_STR_EMPTY,
 275                 MSG_STR_EMPTY,
 276                 MSG_STR_EMPTY,
 277                 MSG_STR_EMPTY,
 278                 MSG_STR_EMPTY,
 279                 MSG_STR_EMPTY,
 280                 MSG_STR_EMPTY,
 281                 MSG_STR_EMPTY,
 282                 MSG_STR_EMPTY,
 283                 MSG_LDD_WARN_UNKCAP,    /* MSG_INTL(MSG_LDD_WARN_UNKCAP) */
 284                 MSG_LDD_WARN_HWCAP_1,   /* MSG_INTL(MSG_LDD_WARN_HWCAP_1) */
 285                 MSG_LDD_WARN_SFCAP_1,   /* MSG_INTL(MSG_LDD_WARN_SFCAP_1) */
 286                 MSG_LDD_WARN_MACHCAP,   /* MSG_INTL(MSG_LDD_WARN_MACHCAP) */
 287                 MSG_LDD_WARN_PLATCAP,   /* MSG_INTL(MSG_LDD_WARN_PLATCAP) */
 288                 MSG_LDD_WARN_HWCAP_2,   /* MSG_INTL(MSG_LDD_WARN_HWCAP_2) */
 289                 MSG_STR_EMPTY,
 290                 MSG_STR_EMPTY,
 285                 MSG_STR_EMPTY
 291         };
 292 #if SGS_REJ_NUM != (SGS_REJ_KMOD + 1)
 287 #if SGS_REJ_NUM != (SGS_REJ_ARCHIVE + 1)
 293 #error SGS_REJ_NUM has changed
 294 #endif
```

```
**********************************************************
   15010 Sun Feb 24 19:19:15 2019
new/usr/src/cmd/sgs/rtld/common/rtld.msg
ld should reject kernel modules as input
**********************************************************
```

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

  22 #
  23 # Copyright (c) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
  24 #

  26 @ _START_

  28 # Message file for cmd/sgs/rtld (ld.so.1)

  30 @ MSG_ID_RTLD

  32 # Usage error
  33 @ MSG_USG_BADOPT        "usage: ld.so.1 [-e option,...] \
  34                          dynamic-object [object args,...]"

  36 # Message formatting error.
  37 @ MSG_EMG_BUFOVRFLW     "ld.so.1: internal: message buffer overflow"

  39 # Argument processing errors

  41 @ MSG_ARG_ILLMODE_1     "illegal mode: RTLD_NOW or RTLD_LAZY or RTLD_NOLOAD \
  42                          required"
  43 @ MSG_ARG_ILLMODE_2     "illegal mode: RTLD_NOW cannot be combined with \
  44                          RTLD_LAZY"
  45 @ MSG_ARG_ILLMODE_3     "illegal mode: LM_ID_NEWLM requires non-zero path"
  46 @ MSG_ARG_ILLMODE_4     "illegal mode: LM_ID_NEWLM cannot be combined with \
  47                          RTLD_PARENT"
  48 @ MSG_ARG_ILLMODE_5     "illegal mode: potential multiple path expansion \
  49                          requires RTLD_FIRST"

  51 @ MSG_ARG_ILLPATH       "illegal pathname"
  52 @ MSG_ARG_ILLSYM        "illegal symbol name"
  53 @ MSG_ARG_ILLNAME       "illegal name"
  54 @ MSG_ARG_INVADDR       "address 0x%llx does not fall within any mapped object"
  55 @ MSG_ARG_INVHNDL       "invalid handle: 0x%llx"
  56 @ MSG_ARG_ILLVAL        "illegal request value"
  57 @ MSG_ARG_NOCONFIG      "no configuration file in use"
  58 @ MSG_ARG_NOPROFNAME    "no profile target specified"
  59 @ MSG_ARG_ATEXIT        "purge of atexit() registrations failed: %d"
  60 @ MSG_ARG_SERCNT        "information path count (%d) insufficient"
  61 @ MSG_ARG_SERSIZE       "information buffer size (%lld) insufficient"
```



```
  62 @ MSG_ARG_ILLFLAGS      "illegal flags value: %d"
  63 @ MSG_ARG_ILLINFO       "non-null info field required for flags value: %d"
  64 @ MSG_ARG_INVSIG        "invalid signal supplied: %d"

  66 # General error diagnostics

  68 @ MSG_GEN_NOOPEN        "DF_1_NOOPEN tagged object may not be dlopen()'ed"

  70 @ MSG_GEN_NOFILE        "%s: can't find file"
  71 @ MSG_GEN_ALTER         "%s: alternate file in use"
  72 @ MSG_GEN_NOSYM         "%s: can't find symbol"
  73 @ MSG_GEN_NODUMP        "%s: DF_1_NODUMP tagged object may not be dldump()'ed"

  75 # Move related messages

  77 @ MSG_MOVE_ERR1         "move entry with illegal size; ignored"


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

  84 @ MSG_REL_NOSYM         "relocation error: file %s: symbol %s: \
  85                          referenced symbol not found"
  86 @ MSG_REL_PLTREF        "relocation error: %s: unidentifiable procedure \
  87                          reference: link-map 0x%llx, offset 0x%llx, \
  88                          called from 0x%llx"
  89 @ MSG_REL_UNSUPSZ       "relocation error: %s: file %s: symbol %s: \
  90                          offset size (%d bytes) is not supported"
  91 @ MSG_REL_BADTLS        "relocation error: %s: file %s: symbol %s: \
  92                          file contains insufficient TLS support information"

  94 # System call messages.

  96 @ MSG_SYS_BRK           "%s: brk failed: %s"
  97 @ MSG_SYS_OPEN          "%s: open failed: %s"
  98 @ MSG_SYS_MMAP          "%s: mmap failed: %s"
  99 @ MSG_SYS_MPROT         "%s: mprotect failed: %s"
 100 @ MSG_SYS_MMAPANON      "mmap anon failed: %s"

 102 @ MSG_SEC_OPEN          "%s: open failed: No such file in secure directories"
 103 @ MSG_SEC_ILLEGAL       "%s: open failed: illegal insecure pathname"


 106 # Configuration failures

 108 @ MSG_CONF_APP          "configuration file: %s: is specific to application: %s"
 109 @ MSG_CONF_DSTAT        "configuration file: %s: original directory %s: stat \
 110                          failed: %s"
 111 @ MSG_CONF_FSTAT        "configuration file: %s: original file %s: stat \
 112                          failed: %s"
 113 @ MSG_CONF_FCMP         "configuration file: %s: original file %s: modified \
 114                          since configuration file creation"

 116 # Link Audit diagnostic message formats

 118 @ MSG_AUD_BADVERS       "version mismatch: current %d: required %d"
 119 @ MSG_AUD_DISABLED      "%s: audit initialization failure: disabled"


 122 # Versioning diagnostics.

 124 @ MSG_VER_NFOUND        "%s: version '%s' not found (required by file %s)"


 127 # Diagnostics generated under the control of ldd(1).
```

```
129 @ MSG_LDD_VER_FIND      "   find version=%s\n"
130 @ MSG_LDD_VER_NFOUND    "\t%s (%s) =>\t (version not found)\n"

132 @ MSG_LDD_SYM_NFOUND    "\tsymbol not found: %s\t\t(%s)\n"

134 @ MSG_LDD_PTH_TRYING    "    trying path=%s%s\n"
135 @ MSG_LDD_PTH_LIBPATH   "    search path=%s  (LD_LIBRARY_PATH)\n"
136 @ MSG_LDD_PTH_LIBPATHC  "    search path=%s  (configuration \
137                          LD_LIBRARY_PATH - %s)\n"
138 @ MSG_LDD_PTH_RUNPATH   "    search path=%s  (RUNPATH/RPATH from file %s)\n"
139 @ MSG_LDD_PTH_BGNDFL    "    search path="
140 @ MSG_LDD_PTH_ENDDFL    "  (default)\n"
141 @ MSG_LDD_PTH_ENDDFLC   "  (configuration default - %s)\n"
142 @ MSG_LDD_PTH_IGNORE    "    ignore path=%s  (insecure directory name)\n"

144 @ MSG_LDD_FIL_FILTER    "\n   object=%s; filter for %s\n"
145 @ MSG_LDD_FIL_FIND      "\n   find object=%s; required by %s\n"
146 @ MSG_LDD_FIL_NFOUND    "\t%s =>\t (file not found)\n"
147 @ MSG_LDD_FIL_ILLEGAL   "\t%s =>\t (illegal insecure pathname)\n"
148 @ MSG_LDD_FIL_ALTER     "  (alternate)"

150 @ MSG_LDD_CAP_NFOUND    "\t%s =>\t (no capability objects found)\n"

152 @ MSG_LDD_SEC_NFOUND    "\t%s =>\t (file not found in secure directories)\n"

154 @ MSG_LDD_REL_ERR1      "\trelocation %s offset invalid: %s: offset=0x%llx \
155                          lies outside memory image; relocation discarded\n"
156 @ MSG_LDD_REL_ERR2      "\tloading after relocation has started: interposition \
157                          request (DF_1_INTERPOSE) ignored: %s\n"
158 @ MSG_LDD_MOVE_ERR      "\tmove %lld offset invalid: %s: offset=0x%llx \
159                          lies outside memory image; move discarded\n"
160 @ MSG_LDD_CPY_SIZDIF    "\trelocation %s sizes differ: %s\n\
161                          \t\t(file %s size=0x%llx; file %s size=0x%llx)\n"
162 @ MSG_LDD_CPY_INSDATA   "\t\t%s size used; possible insufficient data copied\n"
163 @ MSG_LDD_CPY_DATRUNC   "\t\t%s size used; possible data truncation\n"
164 @ MSG_LDD_CPY_PROT      "\trelocation %s symbol: %s: file %s: relocation bound \
165                          to a symbol with STV_PROTECTED visibility\n"

167 @ MSG_LDD_INIT_FMT_01   "\n   cyclic dependencies detected, group [%d]:\n"
168 @ MSG_LDD_INIT_FMT_02   "   init object=%s\n"
169 @ MSG_LDD_INIT_FMT_03   "   init object=%s - cyclic group [%d], referenced \
170                          by:\n"

172 @ MSG_LDD_UNUSED_FMT    "   unused object=%s\n"
173 @ MSG_LDD_UNCYC_FMT     "   unused object=%s; member of cyclic group [%d]\n"
174 @ MSG_LDD_UNREF_FMT     "   unreferenced object=%s; unused dependency of %s\n"

176 @ MSG_LDD_REL_CPYDISP   "\tsymbol %s: file %s: copy relocation symbol may \
177                          have been displacement relocated\n"

179 @ MSG_LDD_REJ_MACH      "  - wrong ELF machine type: %s"
180 @ MSG_LDD_REJ_CLASS     "  - wrong ELF class: %s"
181 @ MSG_LDD_REJ_DATA      "  - wrong ELF data format: %s"
182 @ MSG_LDD_REJ_TYPE      "  - bad ELF type: %s"
183 @ MSG_LDD_REJ_BADFLAG   "  - bad ELF flags value: %s"
184 @ MSG_LDD_REJ_MISFLAG   "  - mismatched ELF flags value: %s"
185 @ MSG_LDD_REJ_VERSION   "  - mismatched ELF/lib version: %s"
186 @ MSG_LDD_REJ_HAL       "  - HAL R1 extensions required"
187 @ MSG_LDD_REJ_US3       "  - Sun UltraSPARC III extensions required"
188 @ MSG_LDD_REJ_STR       "  - %s"
189 @ MSG_LDD_REJ_UNKFILE   "  - unknown file type"
190 @ MSG_LDD_REJ_UNKCAP    "  - unknown capability: %d"
191 @ MSG_LDD_REJ_HWCAP_1   "  - hardware capability (CA_SUNW_HW_1) unsupported: %s"
192 @ MSG_LDD_REJ_SFCAP_1   "  - software capability (CA_SUNW_SF_1) unsupported: %s"
193 @ MSG_LDD_REJ_MACHCAP   "  - machine capability (CA_SUNW_MACH) unsupported: %s"
```



```
194 @ MSG_LDD_REJ_PLATCAP   "  - platform capability (CA_SUNW_PLAT) unsupported: %s"
195 @ MSG_LDD_REJ_HWCAP_2   "  - hardware capability (CA_SUNW_HW_2) unsupported: %s"
196 @ MSG_LDD_REJ_ARCHIVE   "  - invalid archive use"
197 @ MSG_LDD_REJ_KMOD      "  - invalid kernel module use"
198 #endif /* ! codereview */

200 @ MSG_LDD_WARN_UNKCAP   "%s: unknown capability: %d"
201 @ MSG_LDD_WARN_HWCAP_1  "%s: warning: hardware capability (CA_SUNW_HW_1) \
202                          unsupported: %s\n"
203 @ MSG_LDD_WARN_SFCAP_1  "%s: warning: software capability (CA_SUNW_SF_1) \
204                          unsupported: %s\n"
205 @ MSG_LDD_WARN_MACHCAP  "%s: warning: machine capability (CA_SUNW_MACH) \
206                          unsupported: %s\n"
207 @ MSG_LDD_WARN_PLATCAP  "%s: warning: platform capability (CA_SUNW_PLAT) \
208                          unsupported: %s\n"
209 @ MSG_LDD_WARN_HWCAP_2  "%s: warning: hardware capability (CA_SUNW_HW_2) \
210                          unsupported: %s\n"

212 # Error rejection messages.

214 @ MSG_ERR_REJ_MACH      "%s: wrong ELF machine type: %s"
215 @ MSG_ERR_REJ_CLASS     "%s: wrong ELF class: %s"
216 @ MSG_ERR_REJ_DATA      "%s: wrong ELF data format: %s"
217 @ MSG_ERR_REJ_TYPE      "%s: bad ELF type: %s"
218 @ MSG_ERR_REJ_BADFLAG   "%s: bad ELF flags value: %s"
219 @ MSG_ERR_REJ_MISFLAG   "%s: mismatched ELF flags value: %s"
220 @ MSG_ERR_REJ_VERSION   "%s: mismatched ELF/lib version: %s"
221 @ MSG_ERR_REJ_HAL       "%s: HAL R1 extensions required"
222 @ MSG_ERR_REJ_US3       "%s: Sun UltraSPARC III extensions required"
223 @ MSG_ERR_REJ_STR       "%s: %s"
224 @ MSG_ERR_REJ_UNKFILE   "%s: unknown file type"
225 @ MSG_ERR_REJ_UNKCAP    "%s: unknown capability: %d"
226 @ MSG_ERR_REJ_HWCAP_1   "%s: hardware capability (CA_SUNW_HW_1) unsupported: %s"
227 @ MSG_ERR_REJ_SFCAP_1   "%s: software capability (CA_SUNW_SF_1) unsupported: %s"
228 @ MSG_ERR_REJ_MACHCAP   "%s: machine capability (CA_SUNW_MACH) unsupported: %s"
229 @ MSG_ERR_REJ_PLATCAP   "%s: platform capability (CA_SUNW_PLAT) unsupported: %s"
230 @ MSG_ERR_REJ_HWCAP_2   "%s: hardware capability (CA_SUNW_HW_2) unsupported: %s"
231 @ MSG_ERR_REJ_ARCHIVE   "%s: invalid archive use"
232 @ MSG_ERR_REJ_KMOD      "%s: invalid kernel module use"
233 #endif /* ! codereview */

235 # Error TLS failures

237 @ MSG_TLS_NOSUPPORT     "%s: TLS requirement failure : TLS support is \
238                          unavailable"
239 @ MSG_TLS_STATBASE      "%s: static TLS failure: object is not part of primary \
240                          link-map list"
241 @ MSG_TLS_STATSIZE      "%s: static TLS failure: object loaded after process \
242                          initialization: size (%#llx) exceeds available backup \
243                          reservation (%#llx)"
244 @ MSG_TLS_STATINIT      "%s: static TLS failure: object loaded after process \
245                          initialization: can not accommodate initialized data"

247 # Error expand()

249 @ MSG_ERR_EXPAND1       "%s: %s: path name too long"
250 @ MSG_ERR_EXPAND2       "%s: %s: token %s could not be expanded"

252 # Specific dlinfo() messages.

254 @ MSG_DEF_NODEPFOUND    "%s: no deferred dependency found"
255 @ MSG_DEF_NOSYMFOUND    "%s: no deferred symbol found"
256 @ MSG_DEF_DEPLOADED     "%s: deferred dependency is already loaded"

258 # Error diagnostic standard prefixes.
```

```
260 @ MSG_ERR_WARNING       "warning: "
261 @ MSG_ERR_GUIDANCE      "guidance: "
262 @ MSG_ERR_FATAL         "fatal: "
263 @ MSG_ERR_ELF           "elf error: "

265 @ MSG_STR_UNKNOWN       "(unknown)"
266 @ MSG_STR_NULL          "(null)"

268 # Unused errors - used by ldd.

270 @ MSG_USD_LDLIBPATH     "   unused search path=%s  (LD_LIBRARY_PATH)\n"
271 @ MSG_DUP_LDLIBPATH     "   unused (duplicate) search path=%s  \
272                          (LD_LIBRARY_PATH)\n"
273 @ MSG_USD_LDLIBPATHC    "   unused search path=%s  (configuration \
274                          LD_LIBRARY_PATH - %s)\n"
275 @ MSG_DUP_LDLIBPATHC    "   unused (duplicate) search path=%s  (configuration \
276                          LD_LIBRARY_PATH - %s)\n"
277 @ MSG_USD_RUNPATH       "   unused search path=%s  (RUNPATH/RPATH from \
278                          file %s)\n"

280 @ MSG_CAP_IGN_UNKCAP    "ignoring unknown capability: %s"

282 @ _END_

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

288 @ MSG_LDD_FIL_PATH      "\t%s%s%s\n"
289 @ MSG_LDD_FIL_EQUIV     "\t%s =>\t %s%s%s\n"
290 @ MSG_LDD_FMT_PATH1     "%s"
291 @ MSG_LDD_FMT_PATHN     ":%s"
292 @ MSG_LDD_INIT_FMT_FILE "\t%s\n"
293 @ MSG_LDD_VER_FOUND     "\t%s (%s) =>\t %s\n"

295 @ MSG_STR_EMPTY         ""
296 @ MSG_STR_NEGATE        "-"
297 @ MSG_STR_ZERO          "0"
298 @ MSG_STR_HEX           "0x"
299 @ MSG_STR_ELF           "ELF"
300 @ MSG_STR_EMSGFOR1      "%s: %s: %s"
301 @ MSG_STR_EMSGFOR2      "%s: %s"
302 @ MSG_STR_HEXNUM        "0123456789abcdef"
303 @ MSG_STR_NL            "\n"
304 @ MSG_STR_SLASH         "/"
305 @ MSG_STR_DELIMIT       ": "
306 @ MSG_STR_ONE           "1"

308 @ MSG_CAP_DELIMIT       ","

310 @ MSG_SUNW_OST_SGS      "SUNW_OST_SGS"
311 @ MSG_SUNW_OST_OSLIB    "SUNW_OST_OSLIB"

313 @ MSG_TKN_CAPABILITY    "CAPABILITY"
314 @ MSG_TKN_MACHINE       "MACHINE"
315 @ MSG_TKN_PLATFORM      "PLATFORM"
316 @ MSG_TKN_ORIGIN        "ORIGIN"
317 @ MSG_TKN_ISALIST       "ISALIST"
318 @ MSG_TKN_OSNAME        "OSNAME"
319 @ MSG_TKN_OSREL         "OSREL"
320 @ MSG_TKN_HWCAP         "HWCAP"
321 @ MSG_TKN_BINDINGS      "bindings"
322 @ MSG_TKN_POSIX         "POSIX"
323 @ MSG_TKN_DOTDOT        ".."

325 @ MSG_FMT_CWD           "."
```



```
326 @ MSG_FMT_MSGFILE       "/usr/lib/locale/%s/LC_MESSAGES/%s.mo"

328 @ MSG_FIL_RTLD          "ld.so.1"
329 @ MSG_FIL_LIBC          "libc.so.1"

331 @ MSG_SYM_ELFERRMSG     "elf_errmsg"
332 @ MSG_SYM_ELFERRNO      "elf_errno"
333 @ MSG_SYM_ELFPLTTRACE   "elf_plt_trace"
334 @ MSG_SYM_ENVIRON       "_environ"

336 @ MSG_SYM_LAPREINIT     "la_preinit"
337 @ MSG_SYM_LAVERSION     "la_version"
338 @ MSG_SYM_LAACTIVITY    "la_activity"
339 @ MSG_SYM_LAOBJSEARCH   "la_objsearch"
340 @ MSG_SYM_LAOBJOPEN     "la_objopen"
341 @ MSG_SYM_LAOBJFILTER   "la_objfilter"
342 @ MSG_SYM_LAOBJCLOSE    "la_objclose"
343 @ MSG_SYM_LADYNDATA     "la_dyndata"

345 @ MSG_SYM_START         "_START_"

347 @ MSG_SPECFIL_DYNPLT    "dyn_plt(ld.so.1)"

349 @ MSG_PTH_LDPROF        "/usr/lib/link_audit/ldprof.so.1"
350 @ MSG_PTH_LDPROFSE      "/usr/lib/secure/ldprof.so.1"
351 @ MSG_PTH_LIBSYS        "/usr/lib/libsys.so.1"
352 @ MSG_PTH_RTLD          "/usr/lib/ld.so.1"
353 @ MSG_PTH_LIB           "/lib"
354 @ MSG_PTH_USRLIB        "/usr/lib"
355 @ MSG_PTH_LIBSE         "/lib/secure"
356 @ MSG_PTH_USRLIBSE      "/usr/lib/secure"
357 @ MSG_PTH_DEVNULL       "/dev/null"
358 @ MSG_PTH_CONFIG        "/var/ld/ld.config"
359 @ MSG_PTH_VARTMP        "/var/tmp"

361 @ MSG_ORG_CONFIG        "$ORIGIN/ld.config.%s"

363 @ MSG_LD_AUDIT          "AUDIT"
364 @ MSG_LD_AUDIT_ARGS     "AUDIT_ARGS"
365 @ MSG_LD_BIND_LAZY      "BIND_LAZY"
366 @ MSG_LD_BIND_NOW       "BIND_NOW"
367 @ MSG_LD_BIND_NOT       "BIND_NOT"
368 @ MSG_LD_BINDINGS       "BINDINGS"
369 @ MSG_LD_CONFGEN        "CONFGEN"
370 @ MSG_LD_CAP_FILES      "CAP_FILES"
371 @ MSG_LD_CONFIG         "CONFIG"
372 @ MSG_LD_DEBUG          "DEBUG"
373 @ MSG_LD_DEBUG_OUTPUT   "DEBUG_OUTPUT"
374 @ MSG_LD_DEMANGLE       "DEMANGLE"
375 @ MSG_LD_FLAGS          "FLAGS"
376 @ MSG_LD_HWCAP          "HWCAP"
377 @ MSG_LD_INIT           "INIT"
378 @ MSG_LD_LIBPATH        "LIBRARY_PATH"
379 @ MSG_LD_LOADAVAIL      "LOADAVAIL"
380 @ MSG_LD_LOADFLTR       "LOADFLTR"
381 @ MSG_LD_MACHCAP        "MACHCAP"
382 @ MSG_LD_NOAUDIT        "NOAUDIT"
383 @ MSG_LD_NOAUXFLTR      "NOAUXFLTR"
384 @ MSG_LD_NOBAPLT        "NOBAPLT"
385 @ MSG_LD_NOCONFIG       "NOCONFIG"
386 @ MSG_LD_NODIRCONFIG    "NODIRCONFIG"
387 @ MSG_LD_NODIRECT       "NODIRECT"
388 @ MSG_LD_NOENVCONFIG    "NOENVCONFIG"
389 @ MSG_LD_NOENVIRON      "NOENVIRON"
390 @ MSG_LD_NOFLTCONFIG    "NOFLTCONFIG"
391 @ MSG_LD_NOLAZY         "NOLAZYLOAD"
```

```
392 @ MSG_LD_NOOBJALTER     "NOOBJALTER"
393 @ MSG_LD_NOPAREXT       "NOPAREXT"
394 @ MSG_LD_NOUNRESWEAK    "NOUNRESWEAK"
395 @ MSG_LD_NOVERSION      "NOVERSION"
396 @ MSG_LD_PLATCAP        "PLATCAP"
397 @ MSG_LD_PRELOAD        "PRELOAD"
398 @ MSG_LD_PROFILE        "PROFILE"
399 @ MSG_LD_PROFILE_OUTPUT "PROFILE_OUTPUT"
400 @ MSG_LD_SFCAP          "SFCAP"
401 @ MSG_LD_SIGNAL         "SIGNAL"
402 @ MSG_LD_TRACE_OBJS     "TRACE_LOADED_OBJECTS"
403 @ MSG_LD_TRACE_OBJS_E   "TRACE_LOADED_OBJECTS_E"
404 @ MSG_LD_TRACE_OBJS_A   "TRACE_LOADED_OBJECTS_A"
405 @ MSG_LD_TRACE_PTHS     "TRACE_SEARCH_PATHS"
406 @ MSG_LD_UNREF          "UNREF"
407 @ MSG_LD_UNUSED         "UNUSED"
408 @ MSG_LD_VERBOSE        "VERBOSE"
409 @ MSG_LD_DEFERRED       "DEFERRED"
410 @ MSG_LD_WARN           "WARN"

412 @ MSG_LD_BRAND_PREFIX   "BRAND_"

414 @ MSG_LC_ALL            "ALL="
415 @ MSG_LC_MESSAGES       "MESSAGES="

417 @ MSG_EMG_ENOMEM        "internal: Not enough space"

419 @ MSG_DBG_PID           "%5.5d: "
420 @ MSG_DBG_RESET         "---------\n"
421 @ MSG_DBG_UNDEF         "debug: "
422 @ MSG_DBG_LMID          "%s: "
423 @ MSG_DBG_THREAD        "%d: "
424 @ MSG_DBG_FILE          "%s.%5.5d"

426 @ MSG_LMID_BASE         "BASE"
427 @ MSG_LMID_LDSO         "LDSO"
428 @ MSG_LMID_ALT          "ALT"

430 @ MSG_LMID_FMT          "%s%d"
431 @ MSG_LMID_MAXED        "ALTMAXEDOUT"
```

```
**********************************************************
   59984 Sun Feb 24 19:19:16 2019
new/usr/src/man/man1/ld.1
ld: implement -ztype and rework option parsing
**********************************************************
```

```
   1 '\" te
   2 .\" Copyright 1989 AT&T
   3 .\" Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved
   4 .\" Copyright (c) 2012, Joyent, Inc. All Rights Reserved
   5 .\" The contents of this file are subject to the terms of the Common Development
   6 .\"  See the License for the specific language governing permissions and limitat
   7 .\" the fields enclosed by brackets "[]" replaced with your own identifying info
   8 .TH LD 1 "May 13, 2017"
   9 .SH NAME
  10 ld \- link-editor for object files
  11 .SH SYNOPSIS
  12 .LP
  13 .nf
  14 \fBld\fR [\fB-32\fR | \fB-64\fR] [\fB-a\fR | \fB-r\fR] [\fB-b\fR] [\fB-B\fRdirec
  15 [\fB-B\fR dynamic | static] [\fB-B\fR eliminate] [\fB-B\fR group] [\fB-B\fR loca
  16 [\fB-B\fR reduce] [\fB-B\fR symbolic] [\fB-c\fR \fIname\fR] [\fB-C\fR] [\fB-d\fR
  17 [\fB-D\fR \fItoken\fR,...] [\fB-e\fR \fIepsym\fR] [\fB-f\fR \fIname\fR | \fB-F\f
  18 [\fB-i\fR] [\fB-I\fR \fIname\fR] [\fB-l\fR \fIx\fR] [\fB-L\fR \fIpath\fR] [\fB-m
  19 [\fB-N\fR \fIstring\fR] [\fB-o\fR \fIoutfile\fR] [\fB-p\fR \fIauditlib\fR] [\fB-
  20 [\fB-Q\fR y | n] [\fB-R\fR \fIpath\fR] [\fB-s\fR] [\fB-S\fR \fIsupportlib\fR] [\
  21 [\fB-u\fR \fIsymname\fR] [\fB-V\fR] [\fB-Y P\fR\fI,dirlist\fR] [\fB-z\fR absexec
  22 [\fB-z\fR allextract | defaultextract | weakextract ] [\fB-z\fR altexec64]
  23 [\fB-z\fR aslr[=\fIstate\fR]] [\fB-z\fR assert-deflib] [ \fB-z\fR assert-deflib=
  24 [\fB-z\fR combreloc | nocombreloc ] [\fB-z\fR defs | nodefs]
  25 [\fB-z\fR direct | nodirect] [\fB-z\fR endfiltee]
  26 [\fB-z\fR fatal-warnings | nofatal-warnings ] [\fB-z\fR finiarray=\fIfunction\fR
  27 [\fB-z\fR globalaudit] [\fB-z\fR groupperm | nogroupperm]
  28 [\fB-z\fR guidance[=\fIid1\fR,\fIid2\fR...] [\fB-z\fR help ]
  29 [\fB-z\fR ignore | record] [\fB-z\fR initarray=\fIfunction\fR] [\fB-z\fR initfir
  30 [\fB-z\fR interpose] [\fB-z\fR lazyload | nolazyload]
  31 [\fB-z\fR ld32=\fIarg1\fR,\fIarg2\fR,...] [\fB-z\fR ld64=\fIarg1\fR,\fIarg2\fR,.
  32 [\fB-z\fR loadfltr] [\fB-z\fR muldefs] [\fB-z\fR nocompstrtab] [\fB-z\fR nodefau
  33 [\fB-z\fR nodelete] [\fB-z\fR nodlopen] [\fB-z\fR nodump] [\fB-z\fR noldynsym]
  34 [\fB-z\fR nopartial] [\fB-z\fR noversion] [\fB-z\fR now] [\fB-z\fR origin]
  35 [\fB-z\fR preinitarray=\fIfunction\fR] [\fB-z\fR redlocsym] [\fB-z\fR relaxreloc
  36 [\fB-z\fR rescan-now] [\fB-z\fR recan] [\fB-z\fR rescan-start \fI\&...\fR \fB-z\
  37 [\fB-z\fR target=sparc|x86] [\fB-z\fR text | textwarn | textoff]
  38 [\fB-z\fR type=\fIexec\fR|\fIkmod\fR|\fIreloc\fR|\fIshared\fR]
  39 #endif /* ! codereview */
  40 [\fB-z\fR verbose] [\fB-z\fR wrap=\fIsymbol\fR] \fIfilename\fR...
  41 .fi

  43 .SH DESCRIPTION
  44 .LP
  45 The link-editor, \fBld\fR, combines relocatable object files by resolving
  46 symbol references to symbol definitions, together with performing relocations.
  47 \fBld\fR operates in two modes, static or dynamic, as governed by the \fB-d\fR
  48 option. In all cases, the output of \fBld\fR is left in the file \fBa.out\fR by
  49 default. See NOTES.
  50 .sp
  51 .LP
  52 In dynamic mode, \fB-dy\fR, the default, relocatable object files that are
  53 provided as arguments are combined to produce an executable object file. This
  54 file is linked at execution with any shared object files that are provided as
  55 arguments. If the \fB-G\fR option is specified, relocatable object files are
  56 combined to produce a shared object. Without the \fB-G\fR option, a dynamic
  57 executable is created.
  58 .sp
  59 .LP
  60 In static mode, \fB-dn\fR, relocatable object files that are provided as
  61 arguments are combined to produce a static executable file. If the \fB-r\fR
```



```
  62 option is specified, relocatable object files are combined to produce one
  63 relocatable object file. See \fBStatic Executables\fR.
  64 .sp
  65 .LP
  66 Dynamic linking is the most common model for combining relocatable objects, and
  67 the eventual creation of processes within Solaris. This environment tightly
  68 couples the work of the link-editor and the runtime linker, \fBld.so.1\fR(1).
  69 Both of these utilities, together with their related technologies and
  70 utilities, are extensively documented in the \fILinker and Libraries Guide\fR.
  71 .sp
  72 .LP
  73 If any argument is a library, \fBld\fR by default searches the library exactly
  74 once at the point the library is encountered on the argument list. The library
  75 can be either a shared object or relocatable archive. See \fBar.h\fR(3HEAD)).
  76 .sp
  77 .LP
  78 A shared object consists of an indivisible, whole unit that has been generated
  79 by a previous link-edit of one or more input files. When the link-editor
  80 processes a shared object, the entire contents of the shared object become a
  81 logical part of the resulting output file image. The shared object is not
  82 physically copied during the link-edit as its actual inclusion is deferred
  83 until process execution. This logical inclusion means that all symbol entries
  84 defined in the shared object are made available to the link-editing process.
  85 See Chapter 4, \fIShared Objects,\fR in \fILinker and Libraries Guide\fR
  86 .sp
  87 .LP
  88 For an archive library, \fBld\fR loads only those routines that define an
  89 unresolved external reference. \fBld\fR searches the symbol table of the
  90 archive library sequentially to resolve external references that can be
  91 satisfied by library members. This search is repeated until no external
  92 references can be resolved by the archive. Thus, the order of members in the
  93 library is functionally unimportant, unless multiple library members exist that
  94 define the same external symbol. Archive libraries that have interdependencies
  95 can require multiple command line definitions, or the use of one of the
  96 \fB-z\fR \fBrescan\fR options. See \fIArchive Processing\fR in \fILinker and
  97 Libraries Guide\fR.
  98 .sp
  99 .LP
 100 \fBld\fR is a cross link-editor, able to link 32-bit objects or 64-bit objects,
 101 for Sparc or x86 targets. \fBld\fR uses the \fBELF\fR class and machine type of
 102 the first relocatable object on the command line to govern the mode in which to
 103 operate. The mixing of 32-bit objects and 64-bit objects is not permitted.
 104 Similarly, only objects of a single machine type are allowed. See the
 105 \fB-32\fR, \fB-64\fR and \fB-z target\fR options, and the \fBLD_NOEXEC_64\fR
 106 environment variable.
 107 .SS "Static Executables"
 108 .LP
 109 The creation of static executables has been discouraged for many releases. In
 110 fact, 64-bit system archive libraries have never been provided. Because a
 111 static executable is built against system archive libraries, the executable
 112 contains system implementation details. This self-containment has a number of
 113 drawbacks.
 114 .RS +4
 115 .TP
 116 .ie t \(bu
 117 .el o
 118 The executable is immune to the benefits of system updates delivered as shared
 119 objects. The executable therefore, must be rebuilt to take advantage of many
 120 system improvements.
 121 .RE
 122 .RS +4
 123 .TP
 124 .ie t \(bu
 125 .el o
 126 The ability of the executable to run on future releases can be compromised.
 127 .RE
```

```
.RS +4
.TP
.ie t \(bu
.el o
The duplication of system implementation details negatively affects system
performance.
.RE
.sp
.LP
With Solaris 10, 32-bit system archive libraries are no longer provided.
Without these libraries, specifically \fBlibc.a\fR, the creation of static
executables is no longer achievable without specialized system knowledge.
However, the capability of \fBld\fR to process static linking options, and the
processing of archive libraries, remains unchanged.
.SH OPTIONS
.LP
The following options are supported.
.sp
.ne 2
.na
\fB\fB-32\fR | \fB-64\fR\fR
.ad
.sp .6
.RS 4n
Creates a 32-bit, or 64-bit object.
.sp
By default, the class of the object being generated is determined from the
first \fBELF\fR object processed from the command line. If no objects are
specified, the class is determined by the first object encountered within the
first archive processed from the command line. If there are no objects or
archives, the link-editor creates a 32-bit object.
.sp
The \fB-64\fR option is required to create a 64-bit object solely from a
mapfile.
.sp
This \fB-32\fR or \fB-64\fR options can also be used in the rare case of
linking entirely from an archive that contains a mixture of 32 and 64-bit
objects. If the first object in the archive is not the class of the object that
is required to be created, then the \fB-32\fR or \fB-64\fR option can be used
to direct the link-editor. See \fIThe 32-bit link-editor and 64-bit
link-editor\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-a\fR\fR
.ad
.sp .6
.RS 4n
In static mode only, produces an executable object file. Undefined references
are not permitted. This option is the default behavior for static mode. The
\fB-a\fR option can not be used with the \fB-r\fR option. See \fBStatic
Executables\fR under DESCRIPTION.
.RE

.sp
.ne 2
.na
\fB\fB-b\fR\fR
.ad
.sp .6
.RS 4n
In dynamic mode only, provides no special processing for dynamic executable
relocations that reference symbols in shared objects. Without the \fB-b\fR
option, the link-editor applies techniques within a dynamic executable so that
```



```
the text segment can remain read-only. One technique is the creation of special
position-independent relocations for references to functions that are defined
in shared objects. Another technique arranges for data objects that are defined
in shared objects to be copied into the memory image of an executable at
runtime.
.sp
The \fB-b\fR option is intended for specialized dynamic objects and is not
recommended for general use. Its use suppresses all specialized processing
required to ensure an object's shareability, and can even prevent the
relocation of 64-bit executables.
.RE

.sp
.ne 2
.na
\fB\fB-B\fR \fBdirect\fR | \fBnodirect\fR\fR
.ad
.sp .6
.RS 4n
These options govern direct binding. \fB-B\fR \fBdirect\fR establishes direct
binding information by recording the relationship between each symbol reference
together with the dependency that provides the definition. In addition, direct
binding information is established between each symbol reference and an
associated definition within the object being created. The runtime linker uses
this information to search directly for a symbol in the associated object
rather than to carry out a default symbol search.
.sp
Direct binding information can only be established to dependencies specified
with the link-edit. Thus, you should use the \fB-z\fR \fBdefs\fR option.
Objects that wish to interpose on symbols in a direct binding environment
should identify themselves as interposers with the \fB-z\fR \fBinterpose\fR
option. The use of \fB-B\fR \fBdirect\fR enables \fB-z\fR \fBlazyload\fR for
all dependencies.
.sp
The \fB-B\fR \fBnodirect\fR option prevents any direct binding to the
interfaces offered by the object being created. The object being created can
continue to directly bind to external interfaces by specifying the \fB-z\fR
\fBdirect\fR option. See Appendix D, \fIDirect Bindings,\fR in \fILinker and
Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-B\fR \fBdynamic\fR | \fBstatic\fR\fR
.ad
.sp .6
.RS 4n
Options governing library inclusion. \fB-B\fR \fBdynamic\fR is valid in dynamic
mode only. These options can be specified any number of times on the command
line as toggles: if the \fB-B\fR \fBstatic\fR option is given, no shared
objects are accepted until \fB-B\fR \fBdynamic\fR is seen. See the \fB-l\fR
option.
.RE

.sp
.ne 2
.na
\fB\fB-B\fR \fBeliminate\fR\fR
.ad
.sp .6
.RS 4n
Causes any global symbols, not assigned to a version definition, to be
eliminated from the symbol table. Version definitions can be supplied by means
of a \fBmapfile\fR to indicate the global symbols that should remain visible in
the generated object. This option achieves the same symbol elimination as the
```

\fIauto-elimination\fR directive that is available as part of a \fBmapfile\fR version definition. This option can be useful when combining versioned and non-versioned relocatable objects. See also the \fB-B\fR \fBlocal\fR option and the \fB-B\fR \fBreduce\fR option. See \fIDefining Additional Symbols with a mapfile\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-B\fR \fBgroup\fR\fR
.ad
.sp .6
.RS 4n
Establishes a shared object and its dependencies as a group. Objects within the group are bound to other members of the group at runtime. This mode is similar to adding the object to the process by using \fBdlopen\fR(3C) with the \fBRTLD_GROUP\fR mode. An object that has an explicit dependency on a object identified as a group, becomes a member of the group.
.sp
As the group must be self contained, use of the \fB-B\fR \fBgroup\fR option also asserts the \fB-z\fR \fBdefs\fR option.
.RE

.sp
.ne 2
.na
\fB\fB-B\fR \fBlocal\fR\fR
.ad
.sp .6
.RS 4n
Causes any global symbols, not assigned to a version definition, to be reduced to local. Version definitions can be supplied by means of a \fBmapfile\fR to indicate the global symbols that should remain visible in the generated object. This option achieves the same symbol reduction as the \fIauto-reduction\fR directive that is available as part of a \fBmapfile\fR version definition. This option can be useful when combining versioned and non-versioned relocatable objects. See also the \fB-B\fR \fBeliminate\fR option and the \fB-B\fR \fBreduce\fR option. See \fIDefining Additional Symbols with a mapfile\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-B\fR \fBreduce\fR\fR
.ad
.sp .6
.RS 4n
When generating a relocatable object, causes the reduction of symbolic information defined by any version definitions. Version definitions can be supplied by means of a \fBmapfile\fR to indicate the global symbols that should remain visible in the generated object. By default, when a relocatable object is generated, version definitions are only recorded in the output image. The actual reduction of symbolic information is carried out when the object is used in the construction of a dynamic executable or shared object. The \fB-B\fR \fBreduce\fR option is applied automatically when a dynamic executable or shared object is created.
.RE

.sp
.ne 2
.na
\fB\fB-B\fR \fBsymbolic\fR\fR
.ad
.sp .6

.RS 4n
In dynamic mode only. When building a shared object, binds references to global symbols to their definitions, if available, within the object. Normally, references to global symbols within shared objects are not bound until runtime, even if definitions are available. This model allows definitions of the same symbol in an executable or other shared object to override the object's own definition. \fBld\fR issues warnings for undefined symbols unless \fB-z\fR \fBdefs\fR overrides.
.sp
The \fB-B\fR \fBsymbolic\fR option is intended for specialized dynamic objects and is not recommended for general use. To reduce the runtime relocation processing that is required an object, the creation of a version definition is recommended.
.RE

.sp
.ne 2
.na
\fB\fB-c\fR \fIname\fR\fR
.ad
.sp .6
.RS 4n
Records the configuration file \fIname\fR for use at runtime. Configuration files can be employed to alter default search paths, provide a directory cache, together with providing alternative object dependencies. See \fBcrle\fR(1).
.RE

.sp
.ne 2
.na
\fB\fB-C\fR\fR
.ad
.sp .6
.RS 4n
Demangles C++ symbol names displayed in diagnostic messages.
.RE

.sp
.ne 2
.na
\fB\fB-d\fR \fBy\fR | \fBn\fR\fR
.ad
.sp .6
.RS 4n
When \fB-d\fR \fBy\fR, the default, is specified, \fBld\fR uses dynamic linking. When \fB-d\fR \fBn\fR is specified, \fBld\fR uses static linking. See \fBStatic Executables\fR under DESCRIPTION, and \fB-B\fR \fBdynamic\fR|\fBstatic\fR.
.RE

.sp
.ne 2
.na
\fB\fB-D\fR \fItoken\fR,...\fR
.ad
.sp .6
.RS 4n
Prints debugging information as specified by each \fItoken\fR, to the standard error. The special token \fBhelp\fR indicates the full list of tokens available. See \fIDebugging Aids\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-e\fR \fIepsym\fR\fR

```
.ad
.br
.na
\fB\fB--entry\fR \fIepsym\fR\fR
.ad
.sp .6
.RS 4n
Sets the entry point address for the output file to be the symbol \fIepsym\fR.
.RE

.sp
.ne 2
.na
\fB\fB-f\fR \fIname\fR\fR
.ad
.br
.na
\fB\fB--auxiliary\fR \fIname\fR\fR
.ad
.sp .6
.RS 4n
Useful only when building a shared object. Specifies that the symbol table of
the shared object is used as an auxiliary filter on the symbol table of the
shared object specified by \fIname\fR. Multiple instances of this option are
allowed. This option can not be combined with the \fB-F\fR option. See
\fIGenerating Auxiliary Filters\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-F\fR \fIname\fR\fR
.ad
.br
.na
\fB\fB--filter\fR \fIname\fR\fR
.ad
.sp .6
.RS 4n
Useful only when building a shared object. Specifies that the symbol table of
the shared object is used as a filter on the symbol table of the shared object
specified by \fIname\fR. Multiple instances of this option are allowed. This
option can not be combined with the \fB-f\fR option. See \fIGenerating Standard
Filters\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-G\fR\fR
.ad
.br
.na
\fB\fB-shared\fR\fR
.ad
.sp .6
.RS 4n
In dynamic mode only, produces a shared object. Undefined symbols are allowed.
See Chapter 4, \fIShared Objects,\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-h\fR \fIname\fR\fR
.ad
```



```
.br
.na
\fB\fB--soname\fR \fIname\fR\fR
.ad
.sp .6
.RS 4n
In dynamic mode only, when building a shared object, records \fIname\fR in the
object's dynamic section. \fIname\fR is recorded in any dynamic objects that
are linked with this object rather than the object's file system name.
Accordingly, \fIname\fR is used by the runtime linker as the name of the shared
object to search for at runtime. See \fIRecording a Shared Object Name\fR in
\fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-i\fR\fR
.ad
.sp .6
.RS 4n
Ignores \fBLD_LIBRARY_PATH\fR. This option is useful when an
\fBLD_LIBRARY_PATH\fR setting is in effect to influence the runtime library
search, which would interfere with the link-editing being performed.
.RE

.sp
.ne 2
.na
\fB\fB-I\fR \fIname\fR\fR
.ad
.br
.na
\fB\fB--dynamic-linker\fR \fIname\fR\fR
.ad
.sp .6
.RS 4n
When building an executable, uses \fIname\fR as the path name of the
interpreter to be written into the program header. The default in static mode
is no interpreter. In dynamic mode, the default is the name of the runtime
linker, \fBld.so.1\fR(1). Either case can be overridden by \fB-I\fR \fIname\fR.
\fBexec\fR(2) loads this interpreter when the \fBa.out\fR is loaded, and passes
control to the interpreter rather than to the \fBa.out\fR directly.
.RE

.sp
.ne 2
.na
\fB\fB-l\fR \fIx\fR\fR
.ad
.br
.na
\fB\fB--library\fR \fIx\fR\fR
.ad
.sp .6
.RS 4n
Searches a library \fBlib\fR\fIx\fR\fB\&.so\fR or \fBlib\fR\fIx\fR\fB\&.a\fR,
the conventional names for shared object and archive libraries, respectively.
In dynamic mode, unless the \fB-B\fR \fBstatic\fR option is in effect, \fBld\fR
searches each directory specified in the library search path for a
\fBlib\fR\fIx\fR\fB\&.so\fR or \fBlib\fR\fIx\fR\fB\&.a\fR file. The directory
search stops at the first directory containing either. \fBld\fR chooses the
file ending in \fB\&.so\fR if \fB-l\fR\fIx\fR expands to two files with names
of the form \fBlib\fR\fIx\fR\fB\&.so\fR and \fBlib\fR\fIx\fR\fB\&.a\fR. If no
\fBlib\fR\fIx\fR\fB\&.so\fR is found, then \fBld\fR accepts
\fBlib\fR\fIx\fR\fB\&.a\fR. In static mode, or when the \fB-B\fR \fBstatic\fR
```

option is in effect, \fBld\fR selects only the file ending in \fB\&.a\fR.
\fBld\fR searches a library when the library is encountered, so the placement
of \fB-l\fR is significant. See \fILinking With Additional Libraries\fR in
\fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-L\fR \fIpath\fR\fR
.ad
.br
.na
\fB\fB--library-path\fR \fIpath\fR\fR
.ad
.sp .6
.RS 4n
Adds \fIpath\fR to the library search directories. \fBld\fR searches for
libraries first in any directories specified by the \fB-L\fR options and then
in the standard directories. This option is useful only if the option precedes
the \fB-l\fR options to which the \fB-L\fR option applies. See \fIDirectories
Searched by the Link-Editor\fR in \fILinker and Libraries Guide\fR.
.sp
The environment variable \fBLD_LIBRARY_PATH\fR can be used to supplement the
library search path, however the \fB-L\fR option is recommended, as the
environment variable is also interpreted by the runtime environment. See
\fBLD_LIBRARY_PATH\fR under ENVIRONMENT VARIABLES.
.RE

.sp
.ne 2
.na
\fB\fB-m\fR\fR
.ad
.sp .6
.RS 4n
Produces a memory map or listing of the input/output sections, together with
any non-fatal multiply-defined symbols, on the standard output.
.RE

.sp
.ne 2
.na
\fB\fB-M\fR \fImapfile\fR\fR
.ad
.sp .6
.RS 4n
Reads \fImapfile\fR as a text file of directives to \fBld\fR. This option can
be specified multiple times. If \fImapfile\fR is a directory, then all regular
files, as defined by \fBstat\fR(2), within the directory are processed. See
Chapter 9, \fIMapfile Option,\fR in \fILinker and Libraries Guide\fR. Example
mapfiles are provided in \fB/usr/lib/ld\fR. See FILES.
.RE

.sp
.ne 2
.na
\fB\fB-N\fR \fIstring\fR\fR
.ad
.sp .6
.RS 4n
This option causes a \fBDT_NEEDED\fR entry to be added to the \fB\&.dynamic\fR
section of the object being built. The value of the \fBDT_NEEDED\fR string is
the \fIstring\fR that is specified on the command line. This option is position
dependent, and the \fBDT_NEEDED\fR \fB\&.dynamic\fR entry is relative to the
other dynamic dependencies discovered on the link-edit line. This option is

useful for specifying dependencies within device driver relocatable objects
when combined with the \fB-dy\fR and \fB-r\fR options.
.RE

.sp
.ne 2
.na
\fB\fB-o\fR \fIoutfile\fR\fR
.ad
.br
.na
\fB\fB--output\fR \fIoutfile\fR\fR
.ad
.sp .6
.RS 4n
Produces an output object file that is named \fIoutfile\fR. The name of the
default object file is \fBa.out\fR.
.RE

.sp
.ne 2
.na
\fB\fB-p\fR \fIauditlib\fR\fR
.ad
.sp .6
.RS 4n
Identifies an audit library, \fIauditlib\fR. This audit library is used to
audit the object being created at runtime. A shared object identified as
requiring auditing with the \fB-p\fR option, has this requirement inherited by
any object that specifies the shared object as a dependency. See the \fB-P\fR
option. See \fIRuntime Linker Auditing Interface\fR in \fILinker and Libraries
Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-P\fR \fIauditlib\fR\fR
.ad
.sp .6
.RS 4n
Identifies an audit library, \fIauditlib\fR. This audit library is used to
audit the dependencies of the object being created at runtime. Dependency
auditing can also be inherited from dependencies that are identified as
requiring auditing. See the \fB-p\fR option, and the \fB-z\fR \fBglobalaudit\fR
option. See \fIRuntime Linker Auditing Interface\fR in \fILinker and Libraries
Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-Q\fR \fBy\fR | \fBn\fR\fR
.ad
.sp .6
.RS 4n
Under \fB-Q\fR \fBy\fR, an \fBident\fR string is added to the \fB\&.comment\fR
section of the output file. This string identifies the version of the \fBld\fR
used to create the file. This results in multiple \fBld\fR \fBidents\fR when
there have been multiple linking steps, such as when using \fBld\fR \fB-r\fR.
This identification is identical with the default action of the \fBcc\fR
command. \fB-Q\fR \fBn\fR suppresses version identification. \fB\&.comment\fR
sections can be manipulated by the \fBmcs\fR(1) utility.
.RE

.sp

```
.ne 2
.na
\fB\fB-r\fR\fR
.ad
.br
.na
\fB\fB--relocatable\fR\fR
.ad
.sp .6
.RS 4n
Combines relocatable object files to produce one relocatable object file.
\fBld\fR does not complain about unresolved references. This option cannot be
used with the \fB-a\fR option.
.RE

.sp
.ne 2
.na
\fB\fB-R\fR \fIpath\fR\fR
.ad
.br
.na
\fB\fB-rpath\fR \fIpath\fR\fR
.ad
.sp .6
.RS 4n
A colon-separated list of directories used to specify library search
directories to the runtime linker. If present and not NULL, the path is
recorded in the output object file and passed to the runtime linker. Multiple
instances of this option are concatenated together with each \fIpath\fR
separated by a colon. See \fIDirectories Searched by the Runtime Linker\fR in
\fILinker and Libraries Guide\fR.
.sp
The use of a runpath within an associated object is preferable to setting
global search paths such as through the \fBLD_LIBRARY_PATH\fR environment
variable. Only the runpaths that are necessary to find the objects dependencies
should be recorded. \fBldd\fR(1) can also be used to discover unused runpaths
in dynamic objects, when used with the \fB-U\fR option.
.sp
Various tokens can also be supplied with a runpath that provide a flexible
means of identifying system capabilities or an objects location. See Appendix
C, \fIEstablishing Dependencies with Dynamic String Tokens,\fR in \fILinker and
Libraries Guide\fR. The \fB$ORIGIN\fR token is especially useful in allowing
dynamic objects to be relocated to different locations in the file system.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR\fR
.ad
.br
.na
\fB\fB--strip-all\fR\fR
.ad
.sp .6
.RS 4n
Strips symbolic information from the output file. Any debugging information,
that is, \fB\&.line\fR, \fB\&.debug*\fR, and \fB\&.stab*\fR sections, and their
associated relocation entries are removed. Except for relocatable files, a
symbol table \fBSHT_SYMTAB\fR and its associated string table section are not
created in the output object file. The elimination of a \fBSHT_SYMTAB\fR symbol
table can reduce the \fB\&.stab*\fR debugging information that is generated
using the compiler drivers \fB-g\fR option. See the \fB-z\fR \fBredlocsym\fR
and \fB-z\fR \fBnoldynsym\fR options.
.RE
```



```
.sp
.ne 2
.na
\fB\fB-S\fR \fIsupportlib\fR\fR
.ad
.sp .6
.RS 4n
The shared object \fIsupportlib\fR is loaded with \fBld\fR and given
information regarding the linking process. Shared objects that are defined by
using the \fB-S\fR option can also be supplied using the \fBSGS_SUPPORT\fR
environment variable. See \fILink-Editor Support Interface\fR in \fILinker and
Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-t\fR\fR
.ad
.sp .6
.RS 4n
Turns off the warning for multiply-defined symbols that have different sizes or
different alignments.
.RE

.sp
.ne 2
.na
\fB\fB-u\fR \fIsymname\fR\fR
.ad
.br
.na
\fB\fB--undefined\fR \fIsymname\fR\fR
.ad
.sp .6
.RS 4n
Enters \fIsymname\fR as an undefined symbol in the symbol table. This option is
useful for loading entirely from an archive library. In this instance, an
unresolved reference is needed to force the loading of the first routine. The
placement of this option on the command line is significant. This option must
be placed before the library that defines the symbol. See \fIDefining
Additional Symbols with the u option\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-V\fR\fR
.ad
.br
.na
\fB\fB--version\fR\fR
.ad
.sp .6
.RS 4n
Outputs a message giving information about the version of \fBld\fR being used.
.RE

.sp
.ne 2
.na
\fB\fB-Y\fR \fBP,\fR\fIdirlist\fR\fR
.ad
.sp .6
.RS 4n
```

```
Changes the default directories used for finding libraries. \fIdirlist\fR is a
colon-separated path list.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBabsexec\fR\fR
.ad
.sp .6
.RS 4n
Useful only when building a dynamic executable. Specifies that references to
external absolute symbols should be resolved immediately instead of being left
for resolution at runtime. In very specialized circumstances, this option
removes text relocations that can result in excessive swap space demands by an
executable.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBallextract\fR | \fBdefaultextract\fR | \fBweakextract\fR\fR
.ad
.br
.na
\fB\fB--whole-archive\fR | \fB--no-whole-archive\fR\fR
.ad
.sp .6
.RS 4n
Alters the extraction criteria of objects from any archives that follow. By
default, archive members are extracted to satisfy undefined references and to
promote tentative definitions with data definitions. Weak symbol references do
not trigger extraction. Under the \fB-z\fR \fBallextract\fR or
\fB--whole-archive\fR options, all archive members are extracted from the
archive. Under \fB-z\fR \fBweakextract\fR, weak references trigger archive
extraction. The \fB-z\fR \fBdefaultextract\fR or \fB--no-whole-archive\fR
options provide a means of returning to the default following use of the former
extract options. See \fIArchive Processing\fR in \fILinker and Libraries
Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBaltexec64\fR\fR
.ad
.sp .6
.RS 4n
Execute the 64-bit \fBld\fR. The creation of very large 32-bit objects can
exhaust the virtual memory that is available to the 32-bit \fBld\fR. The
\fB-z\fR \fBaltexec64\fR option can be used to force the use of the associated
64-bit \fBld\fR. The 64-bit \fBld\fR provides a larger virtual address space
for building 32-bit objects. See \fIThe 32-bit link-editor and 64-bit
link-editor\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB-z\fR \fBaslr[=\fIstate\fR]\fR
.ad
.sp .6
.RS 4n
Specify whether the executable’s address space should be randomized on
execution.  If \fIstate\fR is "enabled" randomization will always occur when
this executable is run (regardless of inherited settings).  If \fIstate\fR is
```



```
"disabled" randomization will never occur when this executable is run.  If
\fIstate\fR is omitted, ASLR is enabled.

An executable that should simply use the settings inherited from its
environment should not use this flag at all.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBcombreloc\fR | \fBnocombreloc\fR\fR
.ad
.sp .6
.RS 4n
By default, \fBld\fR combines multiple relocation sections when building
executables or shared objects. This section combination differs from
relocatable objects, in which relocation sections are maintained in a
one-to-one relationship with the sections to which the relocations must be
applied. The \fB-z\fR \fBnocombreloc\fR option disables this merging of
relocation sections, and preserves the one-to-one relationship found in the
original relocatable objects.
.sp
\fBld\fR sorts the entries of data relocation sections by their symbol
reference. This sorting reduces runtime symbol lookup. When multiple relocation
sections are combined, this sorting produces the least possible relocation
overhead when objects are loaded into memory, and speeds the runtime loading of
dynamic objects.
.sp
Historically, the individual relocation sections were carried over to any
executable or shared object, and the \fB-z\fR \fBcombreloc\fR option was
required to enable the relocation section merging previously described.
Relocation section merging is now the default. The \fB-z\fR \fBcombreloc\fR
option is still accepted for the benefit of old build environments, but the
option is unnecessary, and has no effect.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBassert-deflib\fR\fR
.ad
.br
.na
\fB\fB-z\fR \fBassert-deflib=\fR\fIlibname\fR\fR
.ad
.sp .6
.RS 4n
Enables warnings that check the location of where libraries passed in with
\fB-l\fR are found. If the link-editor finds a library on its default search
path it will emit a warning. This warning can be made fatal in conjunction with
the option \fB-z fatal-warnings\fR. Passing \fIlibname\fR white lists a library
from this check. The library must be the full name of the library, e.g.
\fIlibc.so\fR. To white list multiple libraries, the \fB-z
assert-deflib=\fR\fIlibname\fR option can be repeated multiple times. This
option is useful when trying to build self-contained objects where a referenced
library might exist in the default system library path and in alternate paths
specified by \fB-L\fR, but you only want the alternate paths to be used.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBdefs\fR | \fBnodefs\fR\fR
.ad
.br
.na
```

```
\fB\fB--no-undefined\fR\fR
.ad
.sp .6
.RS 4n
The \fB-z\fR \fBdefs\fR option and the \fB--no-undefined\fR option force a
fatal error if any undefined symbols remain at the end of the link. This mode
is the default when an executable is built. For historic reasons, this mode is
\fBnot\fR the default when building a shared object. Use of the \fB-z\fR
\fBdefs\fR option is recommended, as this mode assures the object being built
is self-contained. A self-contained object has all symbolic references resolved
internally, or to the object's immediate dependencies.
.sp
The \fB-z\fR \fBnodefs\fR option allows undefined symbols. For historic
reasons, this mode is the default when a shared object is built. When used with
executables, the behavior of references to such undefined symbols is
unspecified. Use of the \fB-z\fR \fBnodefs\fR option is not recommended.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBdirect\fR | \fBnodirect\fR\fR
.ad
.sp .6
.RS 4n
Enables or disables direct binding to any dependencies that follow on the
command line. These options allow finer control over direct binding than the
global counterpart \fB-B\fR \fBdirect\fR. The \fB-z\fR \fBdirect\fR option also
differs from the \fB-B\fR \fBdirect\fR option in the following areas. Direct
binding information is not established between a symbol reference and an
associated definition within the object being created. Lazy loading is not
enabled.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBendfiltee\fR\fR
.ad
.sp .6
.RS 4n
Marks a filtee so that when processed by a filter, the filtee terminates any
further filtee searches by the filter. See \fIReducing Filtee Searches\fR in
\fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBfatal-warnings\fR | \fBnofatal-warnings\fR\fR
.ad
.br
.na
\fB\fB--fatal-warnings\fR | \fB--no-fatal-warnings\fR
.ad
.sp .6
.RS 4n
Controls the behavior of warnings emitted from the link-editor. Setting \fB-z
fatal-warnings\fR promotes warnings emitted by the link-editor to fatal errors
that will cause the link-editor to fail before linking. \fB-z
nofatal-warnings\fR instead demotes these warnings such that they will not cause
the link-editor to exit prematurely.
.RE


.sp
```



```
.ne 2
.na
\fB\fB-z\fR \fBfiniarray=\fR\fIfunction\fR\fR
.ad
.sp .6
.RS 4n
Appends an entry to the \fB\&.fini_array\fR section of the object being built.
If no \fB\&.fini_array\fR section is present, a section is created. The new
entry is initialized to point to \fIfunction\fR. See \fIInitialization and
Termination Sections\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBglobalaudit\fR\fR
.ad
.sp .6
.RS 4n
This option supplements an audit library definition that has been recorded with
the \fB-P\fR option. This option is only meaningful when building a dynamic
executable. Audit libraries that are defined within an object with the \fB-P\fR
option typically allow for the auditing of the immediate dependencies of the
object. The \fB-z\fR \fBglobalaudit\fR promotes the auditor to a global
auditor, thus allowing the auditing of all dependencies. See \fIInvoking the
Auditing Interface\fR in \fILinker and Libraries Guide\fR.
.sp
An auditor established with the \fB-P\fR option and the \fB-z\fR
\fBglobalaudit\fR option, is equivalent to the auditor being established with
the \fBLD_AUDIT\fR environment variable. See \fBld.so.1\fR(1).
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBgroupperm\fR | \fBnogroupperm\fR\fR
.ad
.sp .6
.RS 4n
Assigns, or deassigns each dependency that follows to a unique group. The
assignment of a dependency to a group has the same effect as if the dependency
had been built using the \fB-B\fR \fBgroup\fR option.
.RE

.sp
.ne 2
.na
\fB-z\fR \fBguidance\fR[=\fIid1\fR,\fIid2\fR...]
.ad
.sp .6
.RS 4n
Give messages suggesting link-editor features that could improve the resulting
dynamic object.
.LP
Specific classes of suggestion can be silenced by specifying an optional comma s
list of guidance identifiers.
.LP
The current classes of suggestion provided are:

.sp
.ne 2
.na
Enable use of direct binding
.ad
.sp .6
.RS 4n
```

```
Suggests that \fB-z direct\fR or \fB-B direct\fR be present prior to any
specified dependency.  This allows predictable symbol binding at runtime.

Can be disabled with \fB-z guidance=nodirect\fR
.RE

.sp
.ne 2
.na
Enable lazy dependency loading
.ad
.sp .6
.RS 4n
Suggests that \fB-z lazyload\fR be present prior to any specified dependency.
This allows the dynamic object to be loaded more quickly.

Can be disabled with \fB-z guidance=nolazyload\fR.
.RE

.sp
.ne 2
.na
Shared objects should define all their dependencies.
.ad
.sp .6
.RS 4n
Suggests that \fB-z defs\fR be specified on the link-editor command line.
Shared objects that explicitly state all their dependencies behave more
predictably when used.

Can be be disabled with \fB-z guidance=nodefs\fR
.RE

.sp
.ne 2
.na
Version 2 mapfile syntax
.ad
.sp .6
.RS 4n
Suggests that any specified mapfiles use the more readable version 2 syntax.

Can be disabled with \fB-z guidance=nomapfile\fR.
.RE

.sp
.ne 2
.na
Read-only text segment
.ad
.sp .6
.RS 4n
Should any runtime relocations within the text segment exist, suggests that
the object be compiled with position independent code (PIC).  Keeping large
allocatable sections read-only allows them to be shared between processes
using a given shared object.

Can be disabled with \fB-z guidance=notext\fR
.RE

.sp
.ne 2
.na
No unused dependencies
.ad
.sp .6
```



```
.RS 4n
Suggests that any dependency not referenced by the resulting dynamic object be
removed from the link-editor command line.

Can be disabled with \fB-z guidance=nounused\fR.
.RE
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBhelp\fR\fR
.ad
.br
.na
\fB\fB--help\fR\fR
.ad
.sp .6
.RS 4n
Print a summary of the command line options on the standard output and exit.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBignore\fR | \fBrecord\fR\fR
.ad
.sp .6
.RS 4n
Ignores, or records, dynamic dependencies that are not referenced as part of
the link-edit. Ignores, or records, unreferenced \fBELF\fR sections from the
relocatable objects that are read as part of the link-edit. By default,
\fB-z\fR \fBrecord\fR is in effect.
.sp
If an \fBELF\fR section is ignored, the section is eliminated from the output
file being generated. A section is ignored when three conditions are true. The
eliminated section must contribute to an allocatable segment. The eliminated
section must provide no global symbols. No other section from any object that
contributes to the link-edit, must reference an eliminated section.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBinitarray=\fR\fIfunction\fR\fR
.ad
.sp .6
.RS 4n
Appends an entry to the \fB\&.init_array\fR section of the object being built.
If no \fB\&.init_array\fR section is present, a section is created. The new
entry is initialized to point to \fIfunction\fR. See \fIInitialization and
Termination Sections\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBinitfirst\fR\fR
.ad
.sp .6
.RS 4n
Marks the object so that its runtime initialization occurs before the runtime
initialization of any other objects brought into the process at the same time.
In addition, the object runtime finalization occurs after the runtime
finalization of any other objects removed from the process at the same time.
This option is only meaningful when building a shared object.
```

```
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBinterpose\fR\fR
.ad
.sp .6
.RS 4n
Marks the object as an interposer. At runtime, an object is identified as an
explicit interposer if the object has been tagged using the \fB-z interpose\fR
option. An explicit interposer is also established when an object is loaded
using the \fBLD_PRELOAD\fR environment variable. Implicit interposition can
occur because of the load order of objects, however, this implicit
interposition is unknown to the runtime linker. Explicit interposition can
ensure that interposition takes place regardless of the order in which objects
are loaded. Explicit interposition also ensures that the runtime linker
searches for symbols in any explicit interposers when direct bindings are in
effect.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBlazyload\fR | \fBnolazyload\fR\fR
.ad
.sp .6
.RS 4n
Enables or disables the marking of dynamic dependencies to be lazily loaded.
Dynamic dependencies which are marked \fBlazyload\fR are not loaded at initial
process start-up. These dependencies are delayed until the first binding to the
object is made. \fBNote:\fR Lazy loading requires the correct declaration of
dependencies, together with associated runpaths for each dynamic object used
within a process. See \fILazy Loading of Dynamic Dependencies\fR in \fILinker
and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBld32\fR=\fIarg1\fR,\fIarg2\fR,...\fR
.ad
.br
.na
\fB\fB-z\fR \fBld64\fR=\fIarg1\fR,\fIarg2\fR,...\fR
.ad
.sp .6
.RS 4n
The class of the link-editor is affected by the class of the output file being
created and by the capabilities of the underlying operating system. The
\fB-z\fR \fBld\fR[\fB32\fR|\fB64\fR] options provide a means of defining any
link-editor argument. The defined argument is only interpreted, respectively,
by the 32-bit class or 64-bit class of the link-editor.
.sp
For example, support libraries are class specific, so the correct class of
support library can be ensured using:
.sp
.in +2
.nf
\fBld ... -z ld32=-Saudit32.so.1 -z ld64=-Saudit64.so.1 ...\fR
.fi
.in -2
.sp

The class of link-editor that is invoked is determined from the \fBELF\fR class
of the first relocatable file that is seen on the command line. This
```



```
determination is carried out \fBprior\fR to any \fB-z\fR
\fBld\fR[\fB32\fR|\fB64\fR] processing.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBloadfltr\fR\fR
.ad
.sp .6
.RS 4n
Marks a filter to indicate that filtees must be processed immediately at
runtime. Normally, filter processing is delayed until a symbol reference is
bound to the filter. The runtime processing of an object that contains this
flag mimics that which occurs if the \fBLD_LOADFLTR\fR environment variable is
in effect. See the \fBld.so.1\fR(1).
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBmuldefs\fR\fR
.ad
.br
.na
\fB\fB--allow-multiple-definition\fR\fR
.ad
.sp .6
.RS 4n
Allows multiple symbol definitions. By default, multiple symbol definitions
that occur between relocatable objects result in a fatal error condition. This
option, suppresses the error condition, allowing the first symbol definition to
be taken.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBnocompstrtab\fR\fR
.ad
.sp .6
.RS 4n
Disables the compression of \fBELF\fR string tables. By default, string
compression is applied to \fBSHT_STRTAB\fR sections, and to \fBSHT_PROGBITS\fR
sections that have their \fBSHF_MERGE\fR and \fBSHF_STRINGS\fR section flags
set.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBnodefaultlib\fR\fR
.ad
.sp .6
.RS 4n
Marks the object so that the runtime default library search path, used after
any \fBLD_LIBRARY_PATH\fR or runpaths, is ignored. This option implies that all
dependencies of the object can be satisfied from its runpath.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBnodelete\fR\fR
.ad
.sp .6
```

```
.RS 4n
Marks the object as non-deletable at runtime. This mode is similar to adding
the object to the process by using \fBdlopen\fR(3C) with the
\fBRTLD_NODELETE\fR mode.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBnodlopen\fR\fR
.ad
.sp .6
.RS 4n
Marks the object as not available to \fBdlopen\fR(3C), either as the object
specified by the \fBdlopen()\fR, or as any form of dependency required by the
object specified by the \fBdlopen()\fR. This option is only meaningful when
building a shared object.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBnodump\fR\fR
.ad
.sp .6
.RS 4n
Marks the object as not available to \fBdldump\fR(3C).
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBnoldynsym\fR\fR
.ad
.sp .6
.RS 4n
Prevents the inclusion of a \fB\&.SUNW_ldynsym\fR section in dynamic
executables or sharable libraries. The \fB\&.SUNW_ldynsym\fR section augments
the \fB\&.dynsym\fR section by providing symbols for local functions. Local
function symbols allow debuggers to display local function names in stack
traces from stripped programs. Similarly, \fBdladdr\fR(3C) is able to supply
more accurate results.
.sp
The \fB-z\fR \fBnoldynsym\fR option also prevents the inclusion of the two
symbol sort sections that are related to the \fB\&.SUNW_ldynsym\fR section. The
\fB\&.SUNW_dynsymsort\fR section provides sorted access to regular function and
variable symbols. The \fB\&.SUNW_dyntlssort\fR section provides sorted access
to thread local storage (\fBTLS\fR) variable symbols.
.sp
The \fB\&.SUNW_ldynsym\fR, \fB\&.SUNW_dynsymsort\fR, and
\fB\&.SUNW_dyntlssort\fR sections, which becomes part of the allocable text
segment of the resulting file, cannot be removed by \fBstrip\fR(1). Therefore,
the \fB-z\fR \fBnoldynsym\fR option is the only way to prevent their inclusion.
See the \fB-s\fR and \fB-z\fR \fBredlocsym\fR options.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBnopartial\fR\fR
.ad
.sp .6
.RS 4n
Partially initialized symbols, that are defined within relocatable object
files, are expanded in the output file being generated.
.RE
```



```
.sp
.ne 2
.na
\fB\fB-z\fR \fBnoversion\fR\fR
.ad
.sp .6
.RS 4n
Does not record any versioning sections. Any version sections or associated
\fB\&.dynamic\fR section entries are not generated in the output image.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBnow\fR\fR
.ad
.sp .6
.RS 4n
Marks the object as requiring non-lazy runtime binding. This mode is similar to
adding the object to the process by using \fBdlopen\fR(3C) with the
\fBRTLD_NOW\fR mode. This mode is also similar to having the \fBLD_BIND_NOW\fR
environment variable in effect. See \fBld.so.1\fR(1).
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBorigin\fR\fR
.ad
.sp .6
.RS 4n
Marks the object as requiring immediate \fB$ORIGIN\fR processing at runtime.
This option is only maintained for historic compatibility, as the runtime
analysis of objects to provide for \fB$ORIGIN\fR processing is now default.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBpreinitarray=\fR\fIfunction\fR\fR
.ad
.sp .6
.RS 4n
Appends an entry to the \fB\&.preinitarray\fR section of the object being
built. If no \fB\&.preinitarray\fR section is present, a section is created.
The new entry is initialized to point to \fIfunction\fR. See \fIInitialization
and Termination Sections\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBredlocsym\fR\fR
.ad
.sp .6
.RS 4n
Eliminates all local symbols except for the \fISECT\fR symbols from the symbol
table \fBSHT_SYMTAB\fR. All relocations that refer to local symbols are updated
to refer to the corresponding \fISECT\fR symbol. This option allows specialized
objects to greatly reduce their symbol table sizes. Eliminated local symbols
can reduce the \fB\&.stab*\fR debugging information that is generated using the
compiler drivers \fB-g\fR option. See the \fB-s\fR and \fB-z\fR \fBnoldynsym\fR
options.
.RE
```

```
.sp
.ne 2
.na
\fB\fB-z\fR \fBrelaxreloc\fR\fR
.ad
.sp .6
.RS 4n
\fBld\fR normally issues a fatal error upon encountering a relocation using a
symbol that references an eliminated COMDAT section. If \fB-z\fR
\fBrelaxreloc\fR is enabled, \fBld\fR instead redirects such relocations to the
equivalent symbol in the COMDAT section that was kept. \fB-z\fR
\fBrelaxreloc\fR is a specialized option, mainly of interest to compiler
authors, and is not intended for general use.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBrescan-now\fR\fR
.ad
.br
.na
\fB\fB-z\fR \fBrescan\fR\fR
.ad
.sp .6
.RS 4n
These options rescan the archive files that are provided to the link-edit. By
default, archives are processed once as the archives appear on the command
line. Archives are traditionally specified at the end of the command line so
that their symbol definitions resolve any preceding references. However,
specifying archives multiple times to satisfy their own interdependencies can
be necessary.
.sp
\fB-z\fR \fBrescan-now\fR is a positional option, and is processed by the
link-editor immediately when encountered on the command line. All archives seen
on the command line up to that point are immediately reprocessed in an attempt
to locate additional archive members that resolve symbol references. This
archive rescanning is repeated until a pass over the archives occurs in which
no new members are extracted.
.sp
\fB-z\fR \fBrescan\fR is a position independent option. The link-editor defers
the rescan operation until after it has processed the entire command line, and
then initiates a final rescan operation over all archives seen on the command
line. The \fB-z\fR \fBrescan\fR operation can interact incorrectly
with objects that contain initialization (.init) or finalization (.fini)
sections, preventing the code in those sections from running. For this reason,
\fB-z\fR \fBrescan\fR is deprecated, and use of \fB-z\fR \fBrescan-now\fR is
advised.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBrescan-start\fR ... \fB-z\fR \fBrescan-end\fR\fR
.ad
.br
.na
\fB\fB--start-group\fR ... \fB--end-group\fR\fR
.ad
.br
.na
\fB\fB-(\fR ... \fB-)\fR\fR
.ad
.sp .6
.RS 4n
Defines an archive rescan group. This is a positional construct, and is
```



```
processed by the link-editor immediately upon encountering the closing
delimiter option.  Archives found within the group delimiter options are
reprocessed as a group in an attempt to locate additional archive members that
resolve symbol references. This archive rescanning is repeated until a pass
over the archives occurs in which no new members are extracted.
Archive rescan groups cannot be nested.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBtarget=sparc|x86\fR \fI\fR\fR
.ad
.sp .6
.RS 4n
Specifies the machine type for the output object. Supported targets are Sparc
and x86. The 32-bit machine type for the specified target is used unless the
\fB-64\fR option is also present, in which case the corresponding 64-bit
machine type is used. By default, the machine type of the object being
generated is determined from the first \fBELF\fR object processed from the
command line. If no objects are specified, the machine type is determined by
the first object encountered within the first archive processed from the
command line. If there are no objects or archives, the link-editor assumes the
native machine. This option is useful when creating an object directly with
\fBld\fR whose input is solely from a \fBmapfile\fR. See the \fB-M\fR option.
It can also be useful in the rare case of linking entirely from an archive that
contains objects of different machine types for which the first object is not
of the desired machine type. See \fIThe 32-bit link-editor and 64-bit
link-editor\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBtext\fR\fR
.ad
.sp .6
.RS 4n
In dynamic mode only, forces a fatal error if any relocations against
non-writable, allocatable sections remain. For historic reasons, this mode is
not the default when building an executable or shared object. However, its use
is recommended to ensure that the text segment of the dynamic object being
built is shareable between multiple running processes. A shared text segment
incurs the least relocation overhead when loaded into memory. See
\fIPosition-Independent Code\fR in \fILinker and Libraries Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBtextoff\fR\fR
.ad
.sp .6
.RS 4n
In dynamic mode only, allows relocations against all allocatable sections,
including non-writable ones. This mode is the default when building a shared
object.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBtextwarn\fR\fR
.ad
.sp .6
.RS 4n
```

```
In dynamic mode only, lists a warning if any relocations against non-writable,
allocatable sections remain. This mode is the default when building an
executable.
.RE

.sp
.ne 2
.na
\fB-z\fR \fBtype=exec|kmod|reloc|shared\fR
.ad
.sp .6
.RS 4n
Specifies the type of object to create.

.sp
.ne 2
.na
exec
.ad
.sp .6
.RS 4n
Dynamic executable
.RE

.sp
.ne 2
.na
reloc
.ad
.sp .6
.RS 4n
Relocatable object
.RE

.sp
.ne 2
.na
shared
.ad
.sp .6
.RS 4n
Dynamic shared object
.RE

.sp
.ne 2
.na
kmod
.ad
.sp .6
.RS 4n
illumos kernel module
.RE
#endif /* ! codereview */
.RE

.sp
.ne 2
.na
\fB\fB-z\fR \fBverbose\fR\fR
.ad
.sp .6
.RS 4n
This option provides additional warning diagnostics during a link-edit.
Presently, this option conveys suspicious use of displacement relocations. This
option also conveys the restricted use of static \fBTLS\fR relocations when
```



```
building shared objects. In future, this option might be enhanced to provide
additional diagnostics that are deemed too noisy to be generated by default.
.RE

.sp
.ne 2
.na
\fB\fB-z\fR\fBwrap=\fR\fIsymbol\fR\fR
.ad
.br
.na
\fB\fB-wrap=\fR \fIsymbol\fR\fR
.ad
.br
.na
\fB\fB--wrap=\fR \fIsymbol\fR\fR
.ad
.sp .6
.RS 4n
Rename undefined references to \fIsymbol\fR in order to allow wrapper code to
be linked into the output object without having to modify source code. When
\fB-z wrap\fR is specified, all undefined references to \fIsymbol\fR are
modified to reference \fB__wrap_\fR\fIsymbol\fR, and all references to
\fB__real_\fR\fIsymbol\fR are modified to reference \fIsymbol\fR. The user is
expected to provide an object containing the \fB__wrap_\fR\fIsymbol\fR
function. This wrapper function can call \fB__real_\fR\fIsymbol\fR in order to
reference the actual function being wrapped.
.sp
The following is an example of a wrapper for the \fBmalloc\fR(3C) function:
.sp
.in +2
.nf
void *
__wrap_malloc(size_t c)
{
        (void) printf("malloc called with %zu\en", c);
        return (__real_malloc(c));
}
.fi
.in -2

If you link other code with this file using \fB-z\fR \fBwrap=malloc\fR to
compile all the objects, then all calls to \fBmalloc\fR will call the function
\fB__wrap_malloc\fR instead. The call to \fB__real_malloc\fR will call the real
\fBmalloc\fR function.
.sp
The real and wrapped functions should be maintained in separate source files.
Otherwise, the compiler or assembler may resolve the call instead of leaving
that operation for the link-editor to carry out, and prevent the wrap from
occurring.
.RE

.SH ENVIRONMENT VARIABLES
.ne 2
.na
\fB\fBLD_ALTEXEC\fR\fR
.ad
.sp .6
.RS 4n
An alternative link-editor path name. \fBld\fR executes, and passes control to
this alternative link-editor. This environment variable provides a generic
means of overriding the default link-editor that is called from the various
compiler drivers. See the \fB-z altexec64\fR option.
.RE

.sp
```

```
.ne 2
.na
\fB\fBLD_LIBRARY_PATH\fR\fR
.ad
.sp .6
.RS 4n
A list of directories in which to search for the libraries specified using the
\fB-l\fR option. Multiple directories are separated by a colon. In the most
general case, this environment variable contains two directory lists separated
by a semicolon:
.sp
.in +2
.nf
\fIdirlist1\fR\fB;\fR\fIdirlist2\fR
.fi
.in -2
.sp

If \fBld\fR is called with any number of occurrences of \fB-L\fR, as in:
.sp
.in +2
.nf
\fBld ... -L\fIpath1\fR ... -L\fIpathn\fR ...\fR
.fi
.in -2
.sp

then the search path ordering is:
.sp
.in +2
.nf
\fB\fIdirlist1 path1\fR ... \fIpathn dirlist2\fR LIBPATH\fR
.fi
.in -2
.sp

When the list of directories does not contain a semicolon, the list is
interpreted as \fIdirlist2\fR.
.sp
The \fBLD_LIBRARY_PATH\fR environment variable also affects the runtime linkers
search for dynamic dependencies.
.sp
This environment variable can be specified with a _32 or _64 suffix. This makes
the environment variable specific, respectively, to 32-bit or 64-bit processes
and overrides any non-suffixed version of the environment variable that is in
effect.
.RE

.sp
.ne 2
.na
\fB\fBLD_NOEXEC_64\fR\fR
.ad
.sp .6
.RS 4n
Suppresses the automatic execution of the 64-bit link-editor. By default, the
link-editor executes the 64-bit version when the \fBELF\fR class of the first
relocatable file identifies a 64-bit object. The 64-bit image that a 32-bit
link-editor can create, has some limitations. However, some link-edits might
find the use of the 32-bit link-editor faster.
.RE

.sp
.ne 2
.na
\fB\fBLD_OPTIONS\fR\fR
```



```
.ad
.sp .6
.RS 4n
A default set of options to \fBld\fR. \fBLD_OPTIONS\fR is interpreted by
\fBld\fR just as though its value had been placed on the command line,
immediately following the name used to invoke \fBld\fR, as in:
.sp
.in +2
.nf
\fBld $LD_OPTIONS ... \fIother-arguments\fR ...\fR
.fi
.in -2
.sp

.RE

.sp
.ne 2
.na
\fB\fBLD_RUN_PATH\fR\fR
.ad
.sp .6
.RS 4n
An alternative mechanism for specifying a runpath to the link-editor. See the
\fB-R\fR option. If both \fBLD_RUN_PATH\fR and the \fB-R\fR option are
specified, \fB-R\fR supersedes.
.RE

.sp
.ne 2
.na
\fB\fBSGS_SUPPORT\fR\fR
.ad
.sp .6
.RS 4n
Provides a colon-separated list of shared objects that are loaded with the
link-editor and given information regarding the linking process. This
environment variable can be specified with a _32 or _64 suffix. This makes the
environment variable specific, respectively, to the 32-bit or 64-bit class of
\fBld\fR and overrides any non-suffixed version of the environment variable
that is in effect. See the \fB-S\fR option.
.RE

.sp
.LP
Notice that environment variable-names that begin with the
characters ‘\fBLD_\fR’ are reserved for possible future enhancements to \fBld\fR
\fBld.so.1\fR(1).
.SH FILES
.ne 2
.na
\fB\fBlib\fIx\fR.so\fR\fR
.ad
.RS 15n
shared object libraries.
.RE

.sp
.ne 2
.na
\fB\fBlib\fIx\fR.a\fR\fR
.ad
.RS 15n
archive libraries.
.RE
```

```
.sp
.ne 2
.na
\fB\fBa.out\fR\fR
.ad
.RS 15n
default output file.
.RE

.sp
.ne 2
.na
\fB\fILIBPATH\fR\fR
.ad
.RS 15n
For 32-bit libraries, the default search path is \fB/usr/ccs/lib\fR, followed
by \fB/lib\fR, and finally \fB/usr/lib\fR. For 64-bit libraries, the default
search path is \fB/lib/64\fR, followed by \fB/usr/lib/64\fR.
.RE

.sp
.ne 2
.na
\fB\fB/usr/lib/ld\fR\fR
.ad
.RS 15n
A directory containing several \fBmapfiles\fR that can be used during
link-editing. These \fBmapfiles\fR provide various capabilities, such as
defining memory layouts, aligning bss, and defining non-executable stacks.
.RE

.SH ATTRIBUTES
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
Interface Stability     Committed
.TE

.SH SEE ALSO
.LP
\fBas\fR(1), \fBcrle\fR(1), \fBgprof\fR(1), \fBld.so.1\fR(1), \fBldd\fR(1),
\fBmcs\fR(1), \fBpvs\fR(1), \fBexec\fR(2), \fBstat\fR(2), \fBdlopen\fR(3C),
\fBdldump\fR(3C), \fBelf\fR(3ELF), \fBar.h\fR(3HEAD), \fBa.out\fR(4),
\fBattributes\fR(5)
.sp
.LP
\fILinker and Libraries Guide\fR
.SH NOTES
.LP
Default options applied by \fBld\fR are maintained for historic reasons. In
today’s programming environment, where dynamic objects dominate, alternative
defaults would often make more sense. However, historic defaults must be
maintained to ensure compatibility with existing program development
environments. Historic defaults are called out wherever possible in this
manual. For a description of the current recommended options, see Appendix A,
\fILink-Editor Quick Reference,\fR in \fILinker and Libraries Guide\fR.
.sp
.LP
```



```
If the file being created by \fBld\fR already exists, the file is unlinked
after all input files have been processed. A new file with the specified name
is then created. This allows \fBld\fR to create a new version of the file,
while simultaneously allowing existing processes that are accessing the old
file contents to continue running. If the old file has no other links, the disk
space of the removed file is freed when the last process referencing the file
terminates.
.sp
.LP
The behavior of \fBld\fR when the file being created already exists was changed
with \fBSXCE\fR build \fB43\fR. In older versions, the existing file was
rewritten in place, an approach with the potential to corrupt any running
processes that is using the file. This change has an implication for output
files that have multiple hard links in the file system. Previously, all links
would remain intact, with all links accessing the new file contents. The new
\fBld\fR behavior \fBbreaks\fR such links, with the result that only the
specified output file name references the new file. All the other links
continue to reference the old file. To ensure consistent behavior, applications
that rely on multiple hard links to linker output files should explicitly
remove and relink the other file names.
```

```
**********************************************************
    2618 Sun Feb 24 19:19:16 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)
**********************************************************
```

```
   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 2018, Richard Lowe.
  14 #

  16 set name=pkg.fmri value=pkg:/system/test/elftest@$(PKGVERS)
  17 set name=pkg.description value="ELF Unit Tests"
  18 set name=pkg.summary value="ELF Test Suite"
  19 set name=info.classification \
  20     value=org.opensolaris.category.2008:Development/System
  21 set name=variant.arch value=$(ARCH)
  22 dir path=opt/elf-tests
  23 dir path=opt/elf-tests/bin
  24 dir path=opt/elf-tests/runfiles
  25 dir path=opt/elf-tests/tests
  26 dir path=opt/elf-tests/tests/assert-deflib
  27 dir path=opt/elf-tests/tests/linker-sets
  28 dir path=opt/elf-tests/tests/tls
  29 dir path=opt/elf-tests/tests/tls/x64
  30 dir path=opt/elf-tests/tests/tls/x64/ie
  31 dir path=opt/elf-tests/tests/tls/x86
  32 dir path=opt/elf-tests/tests/tls/x86/ld
  33 file path=opt/elf-tests/bin/elftest mode=0555
  34 file path=opt/elf-tests/runfiles/default.run mode=0444
  35 file path=opt/elf-tests/tests/assert-deflib/link.c mode=0444
  36 file path=opt/elf-tests/tests/assert-deflib/test-deflib mode=0555
  37 file path=opt/elf-tests/tests/linker-sets/in-use-check mode=0555
  38 file path=opt/elf-tests/tests/linker-sets/simple mode=0555
  39 file path=opt/elf-tests/tests/linker-sets/simple-src.c mode=0444
  40 file path=opt/elf-tests/tests/linker-sets/simple.out mode=0444
  41 file path=opt/elf-tests/tests/tls/x64/ie/Makefile.test mode=0444
  42 file path=opt/elf-tests/tests/tls/x64/ie/style1-func-with-r12.s mode=0444
  43 file path=opt/elf-tests/tests/tls/x64/ie/style1-func-with-r13.s mode=0444
  44 file path=opt/elf-tests/tests/tls/x64/ie/style1-func.s mode=0444
  45 file path=opt/elf-tests/tests/tls/x64/ie/style1-main.s mode=0444
  46 file path=opt/elf-tests/tests/tls/x64/ie/style2-with-badness.s mode=0444
  47 file path=opt/elf-tests/tests/tls/x64/ie/style2-with-r12.s mode=0444
  48 file path=opt/elf-tests/tests/tls/x64/ie/style2-with-r13.s mode=0444
  49 file path=opt/elf-tests/tests/tls/x64/ie/style2.s mode=0444
  50 file path=opt/elf-tests/tests/tls/x64/ie/x64-ie-test mode=0555
  51 file path=opt/elf-tests/tests/tls/x86/ld/Makefile.test mode=0444
  52 file path=opt/elf-tests/tests/tls/x86/ld/half-ldm.s mode=0444
  53 file path=opt/elf-tests/tests/tls/x86/ld/x86-ld-test mode=0555
  54 license lic_CDDL license=lic_CDDL
  55 depend fmri=developer/linker type=require
  56 depend fmri=developer/object-file type=require
  57 depend fmri=system/test/testrunner type=require
  58 #endif /* ! codereview */
```

****
687 Sun Feb 24 19:19:17 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 Sun Feb 24 19:19:17 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 Sun Feb 24 19:19:17 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)
**********************************************************
```

```
   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 = scripts

  20 include $(SRC)/test/Makefile.com
  21 #endif /* ! codereview */
```

```
**********************************************************
     852 Sun Feb 24 19:19:17 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 Sun Feb 24 19:19:18 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 Sun Feb 24 19:19:18 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 Sun Feb 24 19:19:18 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 Sun Feb 24 19:19:18 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 Sun Feb 24 19:19:19 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 Sun Feb 24 19:19:19 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 Sun Feb 24 19:19:19 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 Sun Feb 24 19:19:20 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 */
```

```
**********************************************************
    1250 Sun Feb 24 19:19:20 2019
new/usr/src/test/elf-tests/tests/linker-sets/in-use-check.sh
linker_set sections shouldn't need leading '.'
code review
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 existing definitions of the start/stop symbols are reported
# as conflicting with internal symbols.

tmpdir=/tmp/test.$$
mkdir $tmpdir
cd $tmpdir

cleanup() {
    cd /
    rm -fr $tmpdir
}

trap 'cleanup' EXIT

cat > broken.c <<EOF
char foo[1024] __attribute__((section("set_foo")));
void *__start_set_foo;

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_set_foo' already defined in file
if (( $? != 0 )); then
    print -u2 "use of a reserved symbol failed for the wrong reason"
    exit 1;
fi
#endif /* ! codereview */
```

```
**********************************************************
    3414 Sun Feb 24 19:19:20 2019
new/usr/src/test/elf-tests/tests/linker-sets/simple-src.c
linker_set sections shouldn't need leading '.'
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 Sun Feb 24 19:19:20 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 */
```

```
**********************************************************
    1398 Sun Feb 24 19:19:20 2019
new/usr/src/test/elf-tests/tests/linker-sets/simple.sh
code review
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, that 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 Sun Feb 24 19:19:21 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 Sun Feb 24 19:19:21 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 Sun Feb 24 19:19:21 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 Sun Feb 24 19:19:21 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 Sun Feb 24 19:19:22 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 Sun Feb 24 19:19:23 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 Sun Feb 24 19:19:24 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)
****

```
/*
 * 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.
 */

        .section        .rodata.str1.1,"aMS",@progbits,1
.LC0:
        .string "foo: %p\n"
        .text
.globl main
        .type   main, @function
main:
.LFB0:
        pushq   %rbp
.LCFI0:
        movq    %rsp, %rbp
.LCFI1:
        movq    foo@GOTTPOFF(%rip), %r12
        addq    %fs:0, %r12
        movq    %r12, %rsi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        leave
        ret
.LFE0:
        .size   main, .-main
.globl foo
        .section        .rodata.str1.1
.LC1:
        .string "foo"

#endif /* ! codereview */
        .section        .tdata,"awT",@progbits
        .align 8
        .type   foo, @object
        .size   foo, 8
foo:
        .quad   .LC1
```

```
**********************************************************
     952 Sun Feb 24 19:19:24 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 Sun Feb 24 19:19:25 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 Sun Feb 24 19:19:25 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 Sun Feb 24 19:19:26 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 Sun Feb 24 19:19:26 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 Sun Feb 24 19:19:26 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 Sun Feb 24 19:19:26 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 Sun Feb 24 19:19:26 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 */
```

```
**********************************************************
   23445 Sun Feb 24 19:19:27 2019
new/usr/src/uts/common/sys/link.h
ld: implement -ztype and rework option parsing
**********************************************************
_____unchanged_portion_omitted_
  69 #endif  /* defined(_LP64) || defined(_LONGLONG_TYPE) */
  70 #endif  /* _ASM */

  72 /*
  73  * Tag values
  74  */
  75 #define DT_NULL         0       /* last entry in list */
  76 #define DT_NEEDED       1       /* a needed object */
  77 #define DT_PLTRELSZ     2       /* size of relocations for the PLT */
  78 #define DT_PLTGOT       3       /* addresses used by procedure linkage table */
  79 #define DT_HASH         4       /* hash table */
  80 #define DT_STRTAB       5       /* string table */
  81 #define DT_SYMTAB       6       /* symbol table */
  82 #define DT_RELA         7       /* addr of relocation entries */
  83 #define DT_RELASZ       8       /* size of relocation table */
  84 #define DT_RELAENT      9       /* base size of relocation entry */
  85 #define DT_STRSZ        10      /* size of string table */
  86 #define DT_SYMENT       11      /* size of symbol table entry */
  87 #define DT_INIT         12      /* _init addr */
  88 #define DT_FINI         13      /* _fini addr */
  89 #define DT_SONAME       14      /* name of this shared object */
  90 #define DT_RPATH        15      /* run-time search path */
  91 #define DT_SYMBOLIC     16      /* shared object linked -Bsymbolic */
  92 #define DT_REL          17      /* addr of relocation entries */
  93 #define DT_RELSZ        18      /* size of relocation table */
  94 #define DT_RELENT       19      /* base size of relocation entry */
  95 #define DT_PLTREL       20      /* relocation type for PLT entry */
  96 #define DT_DEBUG        21      /* pointer to r_debug structure */
  97 #define DT_TEXTREL      22      /* text relocations remain for this object */
  98 #define DT_JMPREL       23      /* pointer to the PLT relocation entries */
  99 #define DT_BIND_NOW     24      /* perform all relocations at load of object */
 100 #define DT_INIT_ARRAY   25      /* pointer to .init_array */
 101 #define DT_FINI_ARRAY   26      /* pointer to .fini_array */
 102 #define DT_INIT_ARRAYSZ 27      /* size of .init_array */
 103 #define DT_FINI_ARRAYSZ 28      /* size of .fini_array */
 104 #define DT_RUNPATH      29      /* run-time search path */
 105 #define DT_FLAGS        30      /* state flags - see DF_* */

 107 /*
 108  * DT_* encoding rules: The value of each dynamic tag determines the
 109  * interpretation of the d_un union. This convention provides for simpler
 110  * interpretation of dynamic tags by external tools. A tag whose value
 111  * is an even number indicates a dynamic section entry that uses d_ptr.
 112  * A tag whose value is an odd number indicates a dynamic section entry
 113  * that uses d_val, or that uses neither d_ptr nor d_val.
 114  *
 115  * There are exceptions to the above rule:
 116  *      - Tags with values that are less than DT_ENCODING.
 117  *      - Tags with values that fall between DT_LOOS and DT_SUNW_ENCODING
 118  *      - Tags with values that fall between DT_HIOS and DT_LOPROC
 119  *
 120  * Third party tools must handle these exception ranges explicitly
 121  * on an item by item basis.
 122  */
 123 #define DT_ENCODING             32      /* positive tag DT_* encoding rules */
 124                                         /*      start after this */
 125 #define DT_PREINIT_ARRAY        32      /* pointer to .preinit_array */
 126 #define DT_PREINIT_ARRAYSZ      33      /* size of .preinit_array */

 128 #define DT_MAXPOSTAGS           34      /* number of positive tags */
```



```
 130 /*
 131  * DT_* encoding rules do not apply between DT_LOOS and DT_SUNW_ENCODING
 132  */
 133 #define DT_LOOS                 0x6000000d      /* OS specific range */
 134 #define DT_SUNW_AUXILIARY       0x6000000d      /* symbol auxiliary name */
 135 #define DT_SUNW_RTLDINF         0x6000000e      /* ld.so.1 info (private) */
 136 #define DT_SUNW_FILTER          0x6000000f      /* symbol filter name */
 137 #define DT_SUNW_CAP             0x60000010      /* hardware/software */
 138                                                 /*      capabilities */
 139 #define DT_SUNW_SYMTAB          0x60000011      /* symtab with local fcn */
 140                                                 /*      symbols immediately */
 141                                                 /*      preceding DT_SYMTAB */
 142 #define DT_SUNW_SYMSZ           0x60000012      /* Size of SUNW_SYMTAB table */

 144 /*
 145  * DT_* encoding rules apply between DT_SUNW_ENCODING and DT_HIOS
 146  */
 147 #define DT_SUNW_ENCODING        0x60000013      /* DT_* encoding rules resume */
 148                                                 /*      after this */
 149 #define DT_SUNW_SORTENT         0x60000013      /* sizeof [SYM|TLS]SORT entry */
 150 #define DT_SUNW_SYMSORT         0x60000014      /* sym indices sorted by addr */
 151 #define DT_SUNW_SYMSORTSZ       0x60000015      /* size of SUNW_SYMSORT */
 152 #define DT_SUNW_TLSSORT         0x60000016      /* tls sym ndx sort by offset */
 153 #define DT_SUNW_TLSSORTSZ       0x60000017      /* size of SUNW_TLSSORT */
 154 #define DT_SUNW_CAPINFO         0x60000018      /* capabilities symbols */
 155 #define DT_SUNW_STRPAD          0x60000019      /* # of unused bytes at the */
 156                                                 /*      end of dynstr */
 157 #define DT_SUNW_CAPCHAIN        0x6000001a      /* capabilities chain info */
 158 #define DT_SUNW_LDMACH          0x6000001b      /* EM_ machine code of linker */
 159                                                 /*      that produced object */
 160 #define DT_SUNW_CAPCHAINENT     0x6000001d      /* capabilities chain entry */
 161 #define DT_SUNW_CAPCHAINSZ      0x6000001f      /* capabilities chain size */
 162 /* 0x60000021 would be DT_SUNW_PARENT */
 163 #define DT_SUNW_ASLR            0x60000023      /* executable ASLR desire */
 164 #define DT_SUNW_KMOD            0x60000027      /* object is a kernel module */
 165 #endif /* ! codereview */

 167 /*
 168  * DT_* encoding rules do not apply between DT_HIOS and DT_LOPROC
 169  */
 170 #define DT_HIOS                 0x6ffff000

 172 /*
 173  * The following values have been deprecated and remain here to allow
 174  * compatibility with older binaries.
 175  */
 176 #define DT_DEPRECATED_SPARC_REGISTER    0x7000001

 178 /*
 179  * DT_* entries which fall between DT_VALRNGHI & DT_VALRNGLO use the
 180  * Dyn.d_un.d_val field of the Elf*_Dyn structure.
 181  */
 182 #define DT_VALRNGLO     0x6ffffd00

 184 #define DT_GNU_PRELINKED 0x6ffffdf5     /* prelinking timestamp (unused) */
 185 #define DT_GNU_CONFLICTSZ 0x6ffffdf6    /* size of conflict section (unused) */
 186 #define DT_GNU_LIBLISTSZ 0x6ffffdf7     /* size of library list (unused) */
 187 #define DT_CHECKSUM     0x6ffffdf8      /* elf checksum */
 188 #define DT_PLTPADSZ     0x6ffffdf9      /* pltpadding size */
 189 #define DT_MOVEENT      0x6ffffdfa      /* move table entry size */
 190 #define DT_MOVESZ       0x6ffffdfb      /* move table size */
 191 #define DT_FEATURE_1    0x6ffffdfc      /* feature holder (unused) */
 192 #define DT_POSFLAG_1    0x6ffffdfd      /* flags for DT_* entries, effecting */
 193                                         /*      the following DT_* entry. */
 194                                         /*      See DF_P1_* definitions */
```

```
#define DT_SYMINSZ      0x6ffffdfe      /* syminfo table size (in bytes) */
#define DT_SYMINENT     0x6ffffdff      /* syminfo entry size (in bytes) */
#define DT_VALRNGHI     0x6ffffdff

/*
 * DT_* entries which fall between DT_ADDRRNGHI & DT_ADDRRNGLO use the
 * Dyn.d_un.d_ptr field of the Elf*_Dyn structure.
 *
 * If any adjustment is made to the ELF object after it has been
 * built, these entries will need to be adjusted.
 */
#define DT_ADDRRNGLO    0x6fffe00

#define DT_GNU_HASH     0x6ffffef5      /* GNU-style hash table (unused) */
#define DT_TLSDESC_PLT  0x6ffffef6      /* GNU (unused) */
#define DT_TLSDESC_GOT  0x6ffffef7      /* GNU (unused) */
#define DT_GNU_CONFLICT 0x6ffffef8      /* start of conflict section (unused) */
#define DT_GNU_LIBLIST  0x6ffffef9      /* Library list (unused) */

#define DT_CONFIG       0x6ffffefa      /* configuration information */
#define DT_DEPAUDIT     0x6ffffefb      /* dependency auditing */
#define DT_AUDIT        0x6ffffefc      /* object auditing */
#define DT_PLTPAD       0x6ffffefd      /* pltpadding (sparcv9) */
#define DT_MOVETAB      0x6ffffefe      /* move table */
#define DT_SYMINFO      0x6ffffeff      /* syminfo table */
#define DT_ADDRRNGHI    0x6ffffeff

/*
 * The following DT_* entries should have been assigned within one of the
 * DT_* ranges, but existed before such ranges had been established.
 */
#define DT_VERSYM       0x6ffffff0      /* version symbol table - unused by */
                                        /*      Solaris (see libld/update.c) */

#define DT_RELACOUNT    0x6ffffff9      /* number of RELATIVE relocations */
#define DT_RELCOUNT     0x6ffffffa      /* number of RELATIVE relocations */
#define DT_FLAGS_1      0x6ffffffb      /* state flags - see DF_1_* defs */
#define DT_VERDEF       0x6ffffffc      /* version definition table and */
#define DT_VERDEFNUM    0x6ffffffd      /*      associated no. of entries */
#define DT_VERNEED      0x6ffffffe      /* version needed table and */
#define DT_VERNEEDNUM   0x6fffffff      /*      associated no. of entries */

/*
 * DT_* entries between DT_HIPROC and DT_LOPROC are reserved for processor
 * specific semantics.
 *
 * DT_* encoding rules apply to all tag values larger than DT_LOPROC.
 */
#define DT_LOPROC       0x70000000      /* processor specific range */
#define DT_AUXILIARY    0x7ffffffd      /* shared library auxiliary name */
#define DT_USED         0x7ffffffe      /* ignored - same as needed */
#define DT_FILTER       0x7fffffff      /* shared library filter name */
#define DT_HIPROC       0x7fffffff


/*
 * Values for DT_FLAGS
 */
#define DF_ORIGIN       0x00000001      /* ORIGIN processing required */
#define DF_SYMBOLIC     0x00000002      /* symbolic bindings in effect */
#define DF_TEXTREL      0x00000004      /* text relocations remain */
#define DF_BIND_NOW     0x00000008      /* process all relocations */
#define DF_STATIC_TLS   0x00000010      /* obj. contains static TLS refs */

/*
 * Values for the DT_POSFLAG_1 .dynamic entry.
```



```
 * These values only affect the following DT_* entry.
 */
#define DF_P1_LAZYLOAD  0x00000001      /* following object is to be */
                                        /*      lazy loaded */
#define DF_P1_GROUPPERM 0x00000002      /* following object's symbols are */
                                        /*      not available for general */
                                        /*      symbol bindings. */
#define DF_P1_DEFERRED  0x00000004      /* following object is deferred */

/*
 * Values for the DT_FLAGS_1 .dynamic entry.
 */
#define DF_1_NOW        0x00000001      /* set RTLD_NOW for this object */
#define DF_1_GLOBAL     0x00000002      /* set RTLD_GLOBAL for this object */
#define DF_1_GROUP      0x00000004      /* set RTLD_GROUP for this object */
#define DF_1_NODELETE   0x00000008      /* set RTLD_NODELETE for this object */
#define DF_1_LOADFLTR   0x00000010      /* trigger filtee loading at runtime */
#define DF_1_INITFIRST  0x00000020      /* set RTLD_INITFIRST for this object */
#define DF_1_NOOPEN     0x00000040      /* set RTLD_NOOPEN for this object */
#define DF_1_ORIGIN     0x00000080      /* ORIGIN processing required */
#define DF_1_DIRECT     0x00000100      /* direct binding enabled */
#define DF_1_TRANS      0x00000200      /* unused obsolete name */
#define DF_1_INTERPOSE  0x00000400      /* object is an interposer */
#define DF_1_NODEFLIB   0x00000800      /* ignore default library search path */
#define DF_1_NODUMP     0x00001000      /* object can't be dldump(3x)'ed */
#define DF_1_CONFALT    0x00002000      /* configuration alternative created */
#define DF_1_ENDFILTEE  0x00004000      /* filtee terminates filters search */
#define DF_1_DISPRELDNE 0x00008000      /* disp reloc applied at build time */
#define DF_1_DISPRELPND 0x00010000      /* disp reloc applied at run-time */
#define DF_1_NODIRECT   0x00020000      /* object contains symbols that */
                                        /*      cannot be directly bound to */
#define DF_1_IGNMULDEF  0x00040000      /* internal: krtld ignore muldefs */
#define DF_1_NOKSYMS    0x00080000      /* internal: don't export object's */
                                        /*      symbols via /dev/ksyms */
#define DF_1_NOHDR      0x00100000      /* mapfile: 1st segment mapping */
                                        /*      omits ELF & program headers */
#define DF_1_EDITED     0x00200000      /* object has been modified since */
                                        /*      being built by 'ld' */
#define DF_1_NORELOC    0x00400000      /* internal: unrelocated object */
#define DF_1_SYMINTPOSE 0x00800000      /* individual symbol interposers */
                                        /*      exist */
#define DF_1_GLOBAUDIT  0x01000000      /* establish global auditing */
#define DF_1_SINGLETON  0x02000000      /* singleton symbols exist */

/*
 * Values set to DT_FEATURE_1 tag's d_val (unused obsolete tag)
 */
#define DTF_1_PARINIT   0x00000001      /* partially initialization feature */
#define DTF_1_CONFEXP   0x00000002      /* configuration file expected */


/*
 * Version structures.  There are three types of version structure:
 *
 *  o   A definition of the versions within the image itself.
 *      Each version definition is assigned a unique index (starting from
 *      VER_NDX_BGNDEF) which is used to cross-reference symbols associated to
 *      the version.  Each version can have one or more dependencies on other
 *      version definitions within the image.  The version name, and any
 *      dependency names, are specified in the version definition auxiliary
 *      array.  Version definition entries require a version symbol index table.
 *
 *  o   A version requirement on a needed dependency.  Each needed entry
 *      specifies the shared object dependency (as specified in DT_NEEDED).
 *      One or more versions required from this dependency are specified in the
 *      version needed auxiliary array.
```

```
 *
 *  o   A version symbol index table.  Each symbol indexes into this array
 *      to determine its version index.  Index values of VER_NDX_BGNDEF or
 *      greater indicate the version definition to which a symbol is associated.
 *      (the size of a symbol index entry is recorded in the sh_info field).
 */
#ifndef _ASM

typedef struct {                        /* Version Definition Structure. */
        Elf32_Half      vd_version;     /* this structures version revision */
        Elf32_Half      vd_flags;       /* version information */
        Elf32_Half      vd_ndx;         /* version index */
        Elf32_Half      vd_cnt;         /* no. of associated aux entries */
        Elf32_Word      vd_hash;        /* version name hash value */
        Elf32_Word      vd_aux;         /* no. of bytes from start of this */
                                        /*      verdef to verdaux array */
        Elf32_Word      vd_next;        /* no. of bytes from start of this */
} Elf32_Verdef;                         /*      verdef to next verdef entry */

typedef struct {                        /* Verdef Auxiliary Structure. */
        Elf32_Word      vda_name;       /* first element defines the version */
                                        /*      name. Additional entries */
                                        /*      define dependency names. */
        Elf32_Word      vda_next;       /* no. of bytes from start of this */
} Elf32_Verdaux;                        /*      verdaux to next verdaux entry */


typedef struct {                        /* Version Requirement Structure. */
        Elf32_Half      vn_version;     /* this structures version revision */
        Elf32_Half      vn_cnt;         /* no. of associated aux entries */
        Elf32_Word      vn_file;        /* name of needed dependency (file) */
        Elf32_Word      vn_aux;         /* no. of bytes from start of this */
                                        /*      verneed to vernaux array */
        Elf32_Word      vn_next;        /* no. of bytes from start of this */
} Elf32_Verneed;                        /*      verneed to next verneed entry */

typedef struct {                        /* Verneed Auxiliary Structure. */
        Elf32_Word      vna_hash;       /* version name hash value */
        Elf32_Half      vna_flags;      /* version information */
        Elf32_Half      vna_other;
        Elf32_Word      vna_name;       /* version name */
        Elf32_Word      vna_next;       /* no. of bytes from start of this */
} Elf32_Vernaux;                        /*      vernaux to next vernaux entry */

typedef Elf32_Half      Elf32_Versym;   /* Version symbol index array */

typedef struct {
        Elf32_Half      si_boundto;     /* direct bindings - symbol bound to */
        Elf32_Half      si_flags;       /* per symbol flags */
} Elf32_Syminfo;


#if defined(_LP64) || defined(_LONGLONG_TYPE)
typedef struct {
        Elf64_Half      vd_version;     /* this structures version revision */
        Elf64_Half      vd_flags;       /* version information */
        Elf64_Half      vd_ndx;         /* version index */
        Elf64_Half      vd_cnt;         /* no. of associated aux entries */
        Elf64_Word      vd_hash;        /* version name hash value */
        Elf64_Word      vd_aux;         /* no. of bytes from start of this */
                                        /*      verdef to verdaux array */
        Elf64_Word      vd_next;        /* no. of bytes from start of this */
} Elf64_Verdef;                         /*      verdef to next verdef entry */

typedef struct {
        Elf64_Word      vda_name;       /* first element defines the version */
```



```
                                        /*      name. Additional entries */
                                        /*      define dependency names. */
        Elf64_Word      vda_next;       /* no. of bytes from start of this */
} Elf64_Verdaux;                        /*      verdaux to next verdaux entry */

typedef struct {
        Elf64_Half      vn_version;     /* this structures version revision */
        Elf64_Half      vn_cnt;         /* no. of associated aux entries */
        Elf64_Word      vn_file;        /* name of needed dependency (file) */
        Elf64_Word      vn_aux;         /* no. of bytes from start of this */
                                        /*      verneed to vernaux array */
        Elf64_Word      vn_next;        /* no. of bytes from start of this */
} Elf64_Verneed;                        /*      verneed to next verneed entry */

typedef struct {
        Elf64_Word      vna_hash;       /* version name hash value */
        Elf64_Half      vna_flags;      /* version information */
        Elf64_Half      vna_other;
        Elf64_Word      vna_name;       /* version name */
        Elf64_Word      vna_next;       /* no. of bytes from start of this */
} Elf64_Vernaux;                        /*      vernaux to next vernaux entry */

typedef Elf64_Half      Elf64_Versym;

typedef struct {
        Elf64_Half      si_boundto;     /* direct bindings - symbol bound to */
        Elf64_Half      si_flags;       /* per symbol flags */
} Elf64_Syminfo;
#endif  /* defined(_LP64) || defined(_LONGLONG_TYPE) */

#endif  /* _ASM */

/*
 * Versym symbol index values.  Values greater than VER_NDX_GLOBAL
 * and less then VER_NDX_LORESERVE associate symbols with user
 * specified version descriptors.
 */
#define VER_NDX_LOCAL           0       /* symbol is local */
#define VER_NDX_GLOBAL          1       /* symbol is global and assigned to */
                                        /*      the base version */
#define VER_NDX_LORESERVE       0xff00  /* beginning of RESERVED entries */
#define VER_NDX_ELIMINATE       0xff01  /* symbol is to be eliminated */

/*
 * Verdef (vd_flags) and Vernaux (vna_flags) flags values.
 */
#define VER_FLG_BASE            0x1     /* version definition of file itself */
                                        /*      (Verdef only) */
#define VER_FLG_WEAK            0x2     /* weak version identifier */
#define VER_FLG_INFO            0x4     /* version is recorded in object for */
                                        /*      informational purposes */
                                        /*      (Versym reference) only. No */
                                        /*      runtime verification is */
                                        /*      required. (Vernaux only) */

/*
 * Verdef version values.
 */
#define VER_DEF_NONE            0       /* Ver_def version */
#define VER_DEF_CURRENT         1
#define VER_DEF_NUM             2

/*
 * Verneed version values.
 */
#define VER_NEED_NONE           0       /* Ver_need version */
```

```
459 #define VER_NEED_CURRENT        1
460 #define VER_NEED_NUM            2


463 /*
464  * Syminfo flag values
465  */
466 #define SYMINFO_FLG_DIRECT      0x0001  /* symbol ref has direct association */
467                                         /*      to object containing defn. */
468 #define SYMINFO_FLG_FILTER      0x0002  /* symbol ref is associated to a */
469                                         /*      standard filter */
470 #define SYMINFO_FLG_PASSTHRU    SYMINFO_FLG_FILTER /* unused obsolete name */
471 #define SYMINFO_FLG_COPY        0x0004  /* symbol is a copy-reloc */
472 #define SYMINFO_FLG_LAZYLOAD    0x0008  /* object containing defn. should be */
473                                         /*      lazily-loaded */
474 #define SYMINFO_FLG_DIRECTBIND  0x0010  /* ref should be bound directly to */
475                                         /*      object containing defn. */
476 #define SYMINFO_FLG_NOEXTDIRECT 0x0020  /* don't let an external reference */
477                                         /*      directly bind to this symbol */
478 #define SYMINFO_FLG_AUXILIARY   0x0040  /* symbol ref is associated to a */
479                                         /*      auxiliary filter */
480 #define SYMINFO_FLG_INTERPOSE   0x0080  /* symbol defines an interposer */
481 #define SYMINFO_FLG_CAP         0x0100  /* symbol is capabilities specific */
482 #define SYMINFO_FLG_DEFERRED    0x0200  /* symbol should not be included in */
483                                         /*      BIND_NOW relocations */

485 /*
486  * Syminfo.si_boundto values.
487  */
488 #define SYMINFO_BT_SELF         0xffff  /* symbol bound to self */
489 #define SYMINFO_BT_PARENT       0xfffe  /* symbol bound to parent */
490 #define SYMINFO_BT_NONE         0xfffd  /* no special symbol binding */
491 #define SYMINFO_BT_EXTERN       0xfffc  /* symbol defined as external */
492 #define SYMINFO_BT_LOWRESERVE   0xff00  /* beginning of reserved entries */

494 /*
495  * Syminfo version values.
496  */
497 #define SYMINFO_NONE            0       /* Syminfo version */
498 #define SYMINFO_CURRENT         1
499 #define SYMINFO_NUM             2


502 /*
503  * Public structure defined and maintained within the runtime linker
504  */
505 #ifndef _ASM

507 typedef struct link_map Link_map;

509 struct link_map {
510         unsigned long   l_addr;         /* address at which object is mapped */
511         char            *l_name;        /* full name of loaded object */
512 #ifdef _LP64
513         Elf64_Dyn       *l_ld;          /* dynamic structure of object */
514 #else
515         Elf32_Dyn       *l_ld;          /* dynamic structure of object */
516 #endif
517         Link_map        *l_next;        /* next link object */
518         Link_map        *l_prev;        /* previous link object */
519         char            *l_refname;     /* filters reference name */
520 };

522 #ifdef _SYSCALL32
523 typedef struct link_map32 Link_map32;
```



```
525 struct link_map32 {
526         Elf32_Word      l_addr;
527         Elf32_Addr      l_name;
528         Elf32_Addr      l_ld;
529         Elf32_Addr      l_next;
530         Elf32_Addr      l_prev;
531         Elf32_Addr      l_refname;
532 };
533 #endif

535 typedef enum {
536         RT_CONSISTENT,
537         RT_ADD,
538         RT_DELETE
539 } r_state_e;

541 typedef enum {
542         RD_FL_NONE = 0,         /* no flags */
543         RD_FL_ODBG = (1<<0),    /* old style debugger present */
544         RD_FL_DBG = (1<<1)      /* debugging enabled */
545 } rd_flags_e;



549 /*
550  * Debugging events enabled inside of the runtime linker.  To
551  * access these events see the librtld_db interface.
552  */
553 typedef enum {
554         RD_NONE = 0,            /* no event */
555         RD_PREINIT,             /* the Initial rendezvous before .init */
556         RD_POSTINIT,            /* the Second rendezvous after .init */
557         RD_DLACTIVITY           /* a dlopen or dlclose has happened */
558 } rd_event_e;

560 struct r_debug {
561         int             r_version;      /* debugging info version no. */
562         Link_map        *r_map;         /* address of link_map */
563         unsigned long   r_brk;          /* address of update routine */
564         r_state_e       r_state;
565         unsigned long   r_ldbase;       /* base addr of ld.so */
566         Link_map        *r_ldsomap;     /* address of ld.so.1's link map */
567         rd_event_e      r_rdevent;      /* debug event */
568         rd_flags_e      r_flags;        /* misc flags. */
569 };

571 #ifdef _SYSCALL32
572 struct r_debug32 {
573         Elf32_Word      r_version;      /* debugging info version no. */
574         Elf32_Addr      r_map;          /* address of link_map */
575         Elf32_Word      r_brk;          /* address of update routine */
576         r_state_e       r_state;
577         Elf32_Word      r_ldbase;       /* base addr of ld.so */
578         Elf32_Addr      r_ldsomap;      /* address of ld.so.1's link map */
579         rd_event_e      r_rdevent;      /* debug event */
580         rd_flags_e      r_flags;        /* misc flags. */
581 };
582 #endif


585 #define R_DEBUG_VERSION 2               /* current r_debug version */
586 #endif  /* _ASM */

588 /*
589  * Attribute/value structures used to bootstrap ELF-based dynamic linker.
590  */
```

```
#ifndef _ASM
typedef struct {
        Elf32_Sword eb_tag;             /* what this one is */
        union {                         /* possible values */
                Elf32_Word eb_val;
                Elf32_Addr eb_ptr;
                Elf32_Off  eb_off;
        } eb_un;
} Elf32_Boot;

#if defined(_LP64) || defined(_LONGLONG_TYPE)
typedef struct {
        Elf64_Xword eb_tag;             /* what this one is */
        union {                         /* possible values */
                Elf64_Xword eb_val;
                Elf64_Addr eb_ptr;
                Elf64_Off eb_off;
        } eb_un;
} Elf64_Boot;
#endif  /* defined(_LP64) || defined(_LONGLONG_TYPE) */
#endif  /* _ASM */

/*
 * Attributes
 */
#define EB_NULL         0               /* (void) last entry */
#define EB_DYNAMIC      1               /* (*) dynamic structure of subject */
#define EB_LDSO_BASE    2               /* (caddr_t) base address of ld.so */
#define EB_ARGV         3               /* (caddr_t) argument vector */
#define EB_ENVP         4               /* (char **) environment strings */
#define EB_AUXV         5               /* (auxv_t *) auxiliary vector */
#define EB_DEVZERO      6               /* (int) fd for /dev/zero */
#define EB_PAGESIZE     7               /* (int) page size */
#define EB_MAX          8               /* number of "EBs" */
#define EB_MAX_SIZE32   64              /* size in bytes, _ILP32 */
#define EB_MAX_SIZE64   128             /* size in bytes, _LP64 */


#ifndef _ASM

/*
 * Concurrency communication structure for libc callbacks.
 */
extern void     _ld_libc(void *);

#pragma unknown_control_flow(_ld_libc)
#endif /* _ASM */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LINK_H */
```