```
**********************************************************
   23180 Fri Aug  9 16:25:48 2013
new/usr/src/cmd/dis/dis_target.c
4004 dis(1) can't deal with extended sections
**********************************************************
______unchanged_portion_omitted_

 414 /*
 415  * Create a target backed by an ELF file.
 416  */
 417 dis_tgt_t *
 418 dis_tgt_create(const char *file)
 419 {
 420         dis_tgt_t *tgt, *current;
 421         int idx;
 422         Elf *elf;
 423         GElf_Ehdr ehdr;
 424         Elf_Arhdr *arhdr = NULL;
 425         int cmd;

 427         if (elf_version(EV_CURRENT) == EV_NONE)
 428                 die("libelf(3ELF) out of date");

 430         tgt = safe_malloc(sizeof (dis_tgt_t));

 432         if ((tgt->dt_fd = open(file, O_RDONLY)) < 0) {
 433                 warn("%s: failed opening file, reason: %s", file,
 434                     strerror(errno));
 435                 free(tgt);
 436                 return (NULL);
 437         }

 439         if ((tgt->dt_elf_root =
 440             elf_begin(tgt->dt_fd, ELF_C_READ, NULL)) == NULL) {
 441                 warn("%s: invalid or corrupt ELF file", file);
 442                 dis_tgt_destroy(tgt);
 443                 return (NULL);
 444         }

 446         current = tgt;
 447         cmd = ELF_C_READ;
 448         while ((elf = elf_begin(tgt->dt_fd, cmd, tgt->dt_elf_root)) != NULL) {
 449                 size_t shnum = 0;
 450 #endif /* ! codereview */

 452                 if (elf_kind(tgt->dt_elf_root) == ELF_K_AR &&
 453                     (arhdr = elf_getarhdr(elf)) == NULL) {
 454                         warn("%s: malformed archive", file);
 455                         dis_tgt_destroy(tgt);
 456                         return (NULL);
 457                 }

 459                 /*
 460                  * Make sure that this Elf file is sane
 461                  */
 462                 if (gelf_getehdr(elf, &ehdr) == NULL) {
 463                         if (arhdr != NULL) {
 464                                 /*
 465                                  * For archives, we drive on in the face of bad
 466                                  * members.  The "/" and "//" members are
 467                                  * special, and should be silently ignored.
 468                                  */
 469                                 if (strcmp(arhdr->ar_name, "/") != 0 &&
 470                                     strcmp(arhdr->ar_name, "//") != 0)
 471                                         warn("%s[%s]: invalid file type",
 472                                             file, arhdr->ar_name);
```



```
 473                                 cmd = elf_next(elf);
 474                                 (void) elf_end(elf);
 475                                 continue;
 476                         }

 478                         warn("%s: invalid file type", file);
 479                         dis_tgt_destroy(tgt);
 480                         return (NULL);
 481                 }

 483                 /*
 484                  * If we're seeing a new Elf object, then we have an
 485                  * archive. In this case, we create a new target, and chain it
 486                  * off the master target.  We can later iterate over these
 487                  * targets using dis_tgt_next().
 488                  */
 489                 if (current->dt_elf != NULL) {
 490                         dis_tgt_t *next = safe_malloc(sizeof (dis_tgt_t));
 491                         next->dt_elf_root = tgt->dt_elf_root;
 492                         next->dt_fd = -1;
 493                         current->dt_next = next;
 494                         current = next;
 495                 }
 496                 current->dt_elf = elf;
 497                 current->dt_arhdr = arhdr;

 499                 if (elf_getshdrstrndx(elf, &current->dt_shstrndx) == -1) {
 500                         warn("%s: failed to get section string table for "
 501                             "file", file);
 502                         dis_tgt_destroy(tgt);
 503                         return (NULL);
 504                 }

 506                 if (elf_getshdrnum(elf, &shnum) == -1) {
 507                         warn("%s: failed to get number of sections in file",
 508                             file);
 509                         dis_tgt_destroy(tgt);
 510                         return (NULL);
 511                 }

 513 #endif /* ! codereview */
 514                 current->dt_shnmap = safe_malloc(sizeof (dis_shnmap_t) *
 515                     shnum);
 516                 current->dt_shncount = shnum;
 449                     ehdr.e_shnum);
 450                 current->dt_shncount = ehdr.e_shnum;

 518                 idx = 0;
 519                 dis_tgt_section_iter(current, tgt_scn_init, &idx);
 520                 current->dt_filename = file;

 522                 create_addrmap(current);
 523                 if (current->dt_symidx != 0)
 524                         construct_symtab(current);

 526                 cmd = elf_next(elf);
 527         }

 529         /*
 530          * Final sanity check.  If we had an archive with no members, then bail
 531          * out with a nice message.
 532          */
 533         if (tgt->dt_elf == NULL) {
 534                 warn("%s: empty archive\n", file);
 535                 dis_tgt_destroy(tgt);
 536                 return (NULL);
```

```
 537         }

 539         return (tgt);
 540 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    5049 Fri Aug  9 16:25:49 2013
new/usr/src/cmd/sgs/ar/common/ar.msg
4011 ar does weird things with extended ELF sections
**********************************************************
```

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

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

@ _START_

# Message file for cmd/sgs/ar.

@ MSG_ID_AR

@ MSG_USAGE             "usage: ar -d[-SvV] archive file ...\n       \
                         ar -m[-abiSvV] [posname] archive file ...\n       \
                         ar -p[-vV][-sS] archive [file ...]\n       \
                         ar -q[-cuvSV] [-abi] [posname] [file ...]\n       \
                         ar -r[-cuvSV] [-abi] [posname] [file ...]\n       \
                         ar -t[-vV][-sS] archive [file ...]\n       \
                         ar -x[-vV][-sSCT] archive [file ...]\n"

@ MSG_MALLOC            "ar: could not allocate memory: %s\n"
@ MSG_TOOBIG4G          "ar: archive size exceeds capabilities of 32-bit \
                         process\n"

@ MSG_USAGE_01          "ar: one of [drqtpmx] must be specified\n"
@ MSG_USAGE_02          "ar: -%c requires an operand\n"
@ MSG_USAGE_03          "ar: bad option: -%c\n"
@ MSG_USAGE_04          "ar: only one of [drqtpmx] allowed\n"
@ MSG_USAGE_05          "ar: abi not allowed with q\n"
@ MSG_USAGE_06          "ar: %s taken as mandatory 'posname' with keys 'abi'\n"

@ MSG_INTERNAL_01       "ar: internal error: cannot tell whether file is \
                         included in archive or not\n"
@ MSG_INTERNAL_02       "ar: internal header generation error\n"

@ MSG_SYS_OPEN          "ar: cannot open %s: %s\n"
@ MSG_SYS_CLOSE         "ar: cannot close %s: %s\n"
@ MSG_SYS_WRITE         "ar: %s: cannot write: %s\n"
@ MSG_SYS_STAT          "ar: cannot stat %s: %s\n"
@ MSG_SYS_RENAME        "ar: cannot rename %s to %s: %s\n"

@ MSG_NOT_FOUND_AR      "ar: archive %s not found\n"
```



```
@ MSG_NOT_FOUND_POSNAM  "ar: posname %s: not found\n"
@ MSG_NOT_FOUND_FILE    "ar: %s not found\n"

@ MSG_ELF_LIB_FILE      "ar: %s caused libelf error: %s\n"
@ MSG_ELF_LIB_AR        "ar: %s(%s) libelf error: %s\n"
@ MSG_ELF_BEGIN_FILE    "ar: cannot elf_begin() %s: %s\n"
@ MSG_ELF_GETDATA_FILE  "ar: %s has bad elf format: %s\n"
@ MSG_ELF_GETDATA_AR    "ar: %s(%s) has bad elf format: %s\n"
@ MSG_ELF_GETSCN_FILE   "ar: %s has no section header or bad elf format: %s\n"
@ MSG_ELF_GETSCN_AR     "ar: %s(%s) has no section header or bad elf \
                         format: %s\n"
@ MSG_ELF_GETSHDRSTRNDX_FILE "ar: %s has no string table index: %s\n"
@ MSG_ELF_GETSHDRSTRNDX_AR "ar: %s(%s) has no string table index: %s\n"
#endif /* ! codereview */
@ MSG_ELF_MALARCHIVE    "ar: %s: offset %lld: malformed archive: %s\n"
@ MSG_ELF_RAWFILE       "ar: elf_rawfile() failed: %s\n"
@ MSG_ELF_VERSION       "ar: libelf.a out of date: %s\n"
@ MSG_W_ELF_NODATA_FILE "ar: %s has no data in section header table\n"
@ MSG_W_ELF_NODATA_AR   "ar: %s(%s) has no data in section header table\n"

@ MSG_BER_MES_CREATE    "ar: creating %s\n"
@ MSG_BER_MES_WRITE     "ar: writing %s\n"

@ MSG_SYMTAB_01         "ar: symbol table entry size is 0\n"
@ MSG_SYMTAB_NOSTR_FILE "ar: %s has no string table for symbol names\n"
@ MSG_SYMTAB_NOSTR_AR   "ar: %s(%s) has no string table for symbol names\n"
@ MSG_SYMTAB_NODAT_FILE "ar: %s has no data in string table\n"
@ MSG_SYMTAB_NODAT_AR   "ar: %s(%s) no data in string table\n"
@ MSG_SYMTAB_ZDAT_FILE  "ar: %s has no data in string table: size is 0\n"
@ MSG_SYMTAB_ZDAT_AR    "ar: %s(%s) no data in string table: size is 0\n"


@ MSG_PATHCONF          "ar: -T failed to calculate file name length: %s\n"
@ MSG_LOCALTIME         "ar: don't have enough space to store the date\n"
@ MSG_NOT_ARCHIVE       "ar: %s not in archive format\n"
@ MSG_OVERRIDE_WARN     "ar: %s already exists. Will not be extracted\n"

@ MSG_ERR_LONGSTRTBLSZ  "ar: archive format limit: long name string table \
                         would exceed 4GB\n"
@ MSG_ERR_MEMBER4G      "ar: archive format limit: individual archive \
                         members are limited to 4GB: %s\n"

@ _END_

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

@ MSG_STR_OPTIONS       ":a:b:i:vucsrdxtplmqVCTzMS"
@ MSG_SUNW_OST_SGS      "SUNW_OST_SGS"

@ MSG_STR_EMPTY         ""
@ MSG_STR_HYPHEN        "-"
@ MSG_STR_PERIOD        "."
@ MSG_STR_LCR           "r"
@ MSG_STR_SLASH         "/"
@ MSG_STR_DSLASH        "//"
@ MSG_STR_SYM64         "/SYM64/"

# Format for full member header
#
@ MSG_MH_FORMAT         "%-16s%-12d%-6u%-6u%-8o%-10lld%-2s"

@ MSG_FMT_VERSION       "ar: %s %s\n"
@ MSG_FMT_P_TITLE       "\n<%s>\n\n"
@ MSG_FMT_T_IDSZ        "%6d/%6d%7lld"
```

```
128 @ MSG_FMT_T_DATE        "%b %e %H:%M %Y"
129 @ MSG_FMT_SPSTRSP       " %s "
130 @ MSG_FMT_STRNL         "%s\n"
131 @ MSG_FMT_FILE          "%c - %s\n"
132 @ MSG_FMT_LLINT         "/%lld"

134 @ MSG_CMD_SWAP          "/usr/sbin/swap -s"

136 # Template for use with mktemp()
137 @ MSG_STR_MKTEMP        "arXXXXXX"
```

```
**********************************************************
   43620 Fri Aug  9 16:25:49 2013
new/usr/src/cmd/sgs/ar/common/file.c
4011 ar does weird things with extended ELF sections
**********************************************************
______unchanged_portion_omitted_

 193 /*
 194  * If the current archive item is an ELF object, then ar(1) may have added
 195  * newline padding at the end in order to bring the following object
 196  * into PADSZ alignment within the file. This padding cannot be
 197  * distinguished from data using the information kept in the member header.
 198  * This routine examines the objects, using knowledge of
 199  * ELF and how our tools lay out objects to determine whether padding was
 200  * added to an archive item. If so, it adjusts the st_size and
 201  * st_padding fields of the file argument to reflect it.
 202  */
 203 static void
 204 recover_padding(Elf *elf, ARFILE *file)
 205 {
 206         size_t          extent;
 207         size_t          padding;
 208         size_t          shnum;
 209 #endif /* ! codereview */
 210         GElf_Ehdr       ehdr;


 213         /* ar(1) only pads objects, so bail if not looking at one */
 214         if (gelf_getclass(elf) == ELFCLASSNONE)
 215                 return;

 217         /*
 218          * libelf always puts the section header array at the end
 219          * of the object, and all of our compilers and other tools
 220          * use libelf or follow this convention. So, it is extremely
 221          * likely that the section header array is at the end of this
 222          * object: Find the address at the end of the array and compare
 223          * it to the archive ar_size. If they are within PADSZ bytes, then
 224          * we've found the end, and the difference is padding (We assume
 225          * that no ELF section can fit into PADSZ bytes).
 226          */
 227         if (elf_getshdrnum(elf, &shnum) == -1)
 228                 return;

 230 #endif /* ! codereview */
 231         extent = gelf_getehdr(elf, &ehdr)
 232             ? (ehdr.e_shoff + (shnum * ehdr.e_shentsize)) : 0;
 208             ? (ehdr.e_shoff + (ehdr.e_shnum * ehdr.e_shentsize)) : 0;

 234         /*
 235          * If the extent exceeds the end of the archive member
 236          * (negative padding), then we don't know what is going on
 237          * and simply leave things alone.
 238          */
 239         if (extent > file->ar_size)
 240                 return;

 242         padding = file->ar_size - extent;
 243         if (padding >= PADSZ) {
 244                 /*
 245                  * The section header array is not at the end of the object.
 246                  * Traverse the section headers and look for the one with
 247                  * the highest used address. If this address is within
 248                  * PADSZ bytes of ar_size, then this is the end of the object.
 249                  */
 250                 Elf_Scn *scn = NULL;
```



```
 252                 do {
 253                         scn = elf_nextscn(elf, scn);
 254                         if (scn) {
 255                                 GElf_Shdr shdr;

 257                                 if (gelf_getshdr(scn, &shdr)) {
 258                                         size_t t;

 260                                         t = shdr.sh_offset + shdr.sh_size;
 261                                         if (t > extent)
 262                                                 extent = t;
 263                                 }
 264                         }
 265                 } while (scn);

 267                 if (extent > file->ar_size)
 268                         return;
 269                 padding = file->ar_size - extent;
 270         }

 272         /*
 273          * Now, test the padding. We only act on padding in the range
 274          * (0 < pad < PADSZ) (ar(1) will never add more than this). A pad
 275          * of 0 requires no action, and any other size above (PADSZ-1) means
 276          * that we don't understand the layout of this object, and as such,
 277          * cannot do anything.
 278          *
 279          * If the padding is in range, and the raw data for the
 280          * object is available, then we perform one additional sanity
 281          * check before moving forward: ar(1) always pads with newline
 282          * characters. If anything else is seen, it is not padding so
 283          * leave it alone.
 284          */
 285         if (padding < PADSZ) {
 286                 if (file->ar_contents) {
 287                         size_t cnt = padding;
 288                         char *p = file->ar_contents + extent;

 290                         while (cnt--) {
 291                                 if (*p++ != '\n') {   /* No padding */
 292                                         padding = 0;
 293                                         break;
 294                                 }
 295                         }
 296                 }

 298                 /* Remove the padding from the size */
 299                 file->ar_size -= padding;
 300                 file->ar_padding = padding;
 301         }
 302 }
______unchanged_portion_omitted_


 464 /*
 465  * Find all the global symbols exported by ELF archive members, and
 466  * build a list associating each one with the archive member that
 467  * provides it.
 468  *
 469  * exit:
 470  *      *symlist is set to the list of symbols. If any ELF object was
 471  *      found, *found_obj is set to TRUE (1). Returns the number of symbols
 472  *      located.
 473  */
 474 static size_t
```

```
mksymtab(const char *arname, ARFILEP **symlist, int *found_obj)
{
        ARFILE          *fptr;
        size_t          mem_offset = 0;
        Elf             *elf;
        Elf_Scn         *scn;
        GElf_Ehdr       ehdr;
        int             newfd;
        size_t          nsyms = 0;
        int             class = 0;
        Elf_Data        *data;
        size_t          num_errs = 0;

        newfd = 0;
        for (fptr = listhead; fptr; fptr = fptr->ar_next) {
                /* determine if file is coming from the archive or not */
                if ((fptr->ar_elf != NULL) && (fptr->ar_pathname == NULL)) {
                        /*
                         * I can use the saved elf descriptor.
                         */
                        elf = fptr->ar_elf;
                } else if ((fptr->ar_elf == NULL) &&
                    (fptr->ar_pathname != NULL)) {
#ifdef _LP64
                        /*
                         * The archive member header ar_size field is 10
                         * decimal digits, sufficient to represent a 32-bit
                         * value, but not a 64-bit one. Hence, we reject
                         * attempts to insert a member larger than 4GB.
                         *
                         * One obvious way to extend the format without altering
                         * the ar_hdr struct is to use the same mechanism used
                         * for ar_name: Put the size string into the long name
                         * string table and write a string /xxx into ar_size,
                         * where xxx is the string table offset.
                         *
                         * At the time of this writing (June 2010), the largest
                         * relocatable objects are measured in 10s or 100s
                         * of megabytes, so we still have many years to go
                         * before this becomes limiting. By that time, it may
                         * turn out that a completely new archive format is
                         * a better solution, as the current format has many
                         * warts and inefficiencies. In the meantime, we
                         * won't burden the current implementation with support
                         * for a bandaid feature that will have little use.
                         */
                        if (fptr->ar_size > 0xffffffff) {
                                (void) fprintf(stderr,
                                    MSG_INTL(MSG_ERR_MEMBER4G),
                                    fptr->ar_pathname);
                                num_errs++;
                                continue;
                        }
#endif
                        if ((newfd  =
                            open(fptr->ar_pathname, O_RDONLY)) == -1) {
                                int err = errno;
                                (void) fprintf(stderr, MSG_INTL(MSG_SYS_OPEN),
                                    fptr->ar_pathname, strerror(err));
                                num_errs++;
                                continue;
                        }

                        if ((elf = elf_begin(newfd,
                            ELF_C_READ, (Elf *)0)) == 0) {
                                (void) fprintf(stderr,
```



```
                                    MSG_INTL(MSG_ELF_BEGIN_FILE),
                                    fptr->ar_pathname, elf_errmsg(-1));
                                (void) close(newfd);
                                newfd = 0;
                                num_errs++;
                                continue;
                        }
                        if (elf_kind(elf) == ELF_K_AR) {
                                if (newfd) {
                                        (void) close(newfd);
                                        newfd = 0;
                                }
                                (void) elf_end(elf);
                                continue;
                        }
                } else {
                        (void) fprintf(stderr, MSG_INTL(MSG_INTERNAL_01));
                        exit(1);
                }
                if (gelf_getehdr(elf, &ehdr) != 0) {
                        size_t shstrndx = 0;
#endif /* ! codereview */
                        if ((class = gelf_getclass(elf)) == ELFCLASS64) {
                                fptr->ar_flag |= F_CLASS64;
                        } else if (class == ELFCLASS32)
                                fptr->ar_flag |= F_CLASS32;

                        if (elf_getshdrstrndx(elf, &shstrndx) == -1) {
                                if (fptr->ar_pathname != NULL) {
                                        (void) fprintf(stderr,
                                            MSG_INTL(MSG_ELF_GETSHDRSTRNDX_FILE)
                                            fptr->ar_pathname, elf_errmsg(-1));
                                } else {
                                        (void) fprintf(stderr,
                                            MSG_INTL(MSG_ELF_GETSHDRSTRNDX_AR),
                                            arname, fptr->ar_longname,
                                            elf_errmsg(-1));
                                }
                                num_errs++;
                                if (newfd) {
                                        (void) close(newfd);
                                        newfd = 0;
                                }
                                (void) elf_end(elf);
                                continue;
                        }

                        scn = elf_getscn(elf, shstrndx);
                        scn = elf_getscn(elf, ehdr.e_shstrndx);
                        if (scn == NULL) {
                                if (fptr->ar_pathname != NULL)
                                        (void) fprintf(stderr,
                                            MSG_INTL(MSG_ELF_GETSCN_FILE),
                                            fptr->ar_pathname, elf_errmsg(-1));
                                else
                                        (void) fprintf(stderr,
                                            MSG_INTL(MSG_ELF_GETSCN_AR),
                                            arname, fptr->ar_longname,
                                            elf_errmsg(-1));
                                num_errs++;
                                if (newfd) {
                                        (void) close(newfd);
                                        newfd = 0;
                                }
                                (void) elf_end(elf);
                                continue;
```

```
606                         }

608                         data = 0;
609                         data = elf_getdata(scn, data);
610                         if (data == NULL) {
611                                 if (fptr->ar_pathname != NULL)
612                                         (void) fprintf(stderr,
613                                             MSG_INTL(MSG_ELF_GETDATA_FILE),
614                                             fptr->ar_pathname, elf_errmsg(-1));
615                                 else
616                                         (void) fprintf(stderr,
617                                             MSG_INTL(MSG_ELF_GETDATA_AR),
618                                             arname, fptr->ar_longname,
619                                             elf_errmsg(-1));
620                                 num_errs++;
621                                 if (newfd) {
622                                         (void) close(newfd);
623                                         newfd = 0;
624                                 }
625                                 (void) elf_end(elf);
626                                 continue;
627                         }
628                         if (data->d_size == 0) {
629                                 if (fptr->ar_pathname != NULL)
630                                         (void) fprintf(stderr,
631                                             MSG_INTL(MSG_W_ELF_NODATA_FILE),
632                                             fptr->ar_pathname);
633                                 else
634                                         (void) fprintf(stderr,
635                                             MSG_INTL(MSG_W_ELF_NODATA_AR),
636                                             arname, fptr->ar_longname);
637                                 if (newfd) {
638                                         (void) close(newfd);
639                                         newfd = 0;
640                                 }
641                                 (void) elf_end(elf);
642                                 num_errs++;
643                                 continue;
644                         }

646                         /* loop through sections to find symbol table */
647                         scn = 0;
648                         while ((scn = elf_nextscn(elf, scn)) != 0) {
649                                 GElf_Shdr shdr;
650                                 if (gelf_getshdr(scn, &shdr) == NULL) {
651                                         /* BEGIN CSTYLED */
652                                         if (fptr->ar_pathname != NULL)
653                                             (void) fprintf(stderr,
654                                                 MSG_INTL(MSG_ELF_GETDATA_FILE),
655                                                 fptr->ar_pathname,
656                                                 elf_errmsg(-1));
657                                         else
658                                             (void) fprintf(stderr,
659                                                 MSG_INTL(MSG_ELF_GETDATA_AR),
660                                                 arname, fptr->ar_longname,
661                                                 elf_errmsg(-1));
662                                         /* END CSTYLED */
663                                         if (newfd) {
664                                                 (void) close(newfd);
665                                                 newfd = 0;
666                                         }
667                                         num_errs++;
668                                         (void) elf_end(elf);
669                                         continue;
670                                 }
671                                 *found_obj = 1;
```



```
672                                 if (shdr.sh_type == SHT_SYMTAB) {
621                                 if (shdr.sh_type == SHT_SYMTAB)
673                                         if (search_sym_tab(arname, fptr, elf,
674                                             scn, &nsyms, symlist,
675                                             &num_errs) == -1) {
676                                                 if (newfd) {
677                                                         (void) close(newfd);
678                                                         newfd = 0;
679                                                 }
680                                                 continue;
681                                         }
682                                 }
683                         }
684                 }
685 #endif /* ! codereview */
686                 mem_offset += sizeof (struct ar_hdr) + fptr->ar_size;
687                 if (fptr->ar_size & 01)
688                         mem_offset++;
689                 (void) elf_end(elf);
690                 if (newfd) {
691                         (void) close(newfd);
692                         newfd = 0;
693                 }
694         }
695         if (num_errs)
696                 exit(1);

698         if (found_obj) {
699                 if (nsyms == 0) {
700                         /*
701                          * It is possible, though rare, to have ELF objects
702                          * that do not export any global symbols. Presumably
703                          * such objects operate via their .init/.fini
704                          * sections. In this case, we produce an empty
705                          * symbol table, so that applications that rely
706                          * on a successful call to elf_getarsym() to determine
707                          * if ELF objects are present will succeed. To do this,
708                          * we require a small empty symbol string table.
709                          */
710                         strtbl_pad(&sym_strtbl, 4, '\0');
711                 } else {
712                         /*
713                          * Historical behavior is to pad string tables
714                          * to a multiple of 4.
715                          */
716                         strtbl_pad(&sym_strtbl, pad(sym_strtbl.used, 4), '\0');
717                 }

719         }

721         return (nsyms);
722 }

724 /*
725  * Output a member header.
726  */
727 /*ARGSUSED*/
728 static void
729 write_member_header(const char *filename, int fd, int is_elf,
730     const char *name, time_t timestamp, uid_t uid, gid_t gid, mode_t mode,
731     size_t size)
732 {
733         char    buf[sizeof (struct ar_hdr) + 1];
734         int     len;

736         len = snprintf(buf, sizeof (buf), MSG_ORIG(MSG_MH_FORMAT), name,
```

```
737             EC_WORD(timestamp), EC_WORD(uid), EC_WORD(gid), EC_WORD(mode),
738             EC_XWORD(size), ARFMAG);

740         /*
741          * If snprintf() reports that it needed more space than we gave
742          * it, it means that the caller fed us a long name, which is a
743          * fatal internal error.
744          */
745         if (len != sizeof (struct ar_hdr)) {
746                 (void) fprintf(stderr, MSG_INTL(MSG_INTERNAL_02));
747                 exit(1);
748         }

750         arwrite(filename, fd, buf, len);

752         /*
753          * We inject inter-member padding to ensure that ELF object
754          * member data is aligned on PADSZ. If this is a debug build,
755          * verify that the computations were right.
756          */
757         assert(!is_elf || (pad(lseek(fd, 0, SEEK_CUR), PADSZ) == 0));
758 }

760 /*
761  * Write the archive symbol table member to the output archive file.
762  *
763  * note:
764  *      sizeofmembers() must have been called to establish member offset
765  *      and padding values before writesymtab() is used.
766  */
767 static void
768 writesymtab(const char *filename, int fd, size_t nsyms, ARFILEP *symlist,
769     size_t eltsize)
770 {
771         size_t  i, j;
772         ARFILEP *ptr;
773         size_t  tblsize;
774         char    *buf, *dst;
775         int     is64 = (eltsize == 8);

777         /*
778          * We require a buffer large enough to hold a symbol table count,
779          * plus one offset for each symbol.
780          */
781         tblsize = (nsyms + 1) * eltsize;
782         if ((buf = dst = malloc(tblsize)) == NULL) {
783                 int err = errno;
784                 (void) fprintf(stderr, MSG_INTL(MSG_MALLOC), strerror(err));
785                 exit(1);
786         }

788         write_member_header(filename, fd, 0,
789             (is64 ? MSG_ORIG(MSG_STR_SYM64) : MSG_ORIG(MSG_STR_SLASH)),
790             time(0), 0, 0, 0, tblsize + sym_strtbl.used);

792         dst = is64 ? sputl64(nsyms, dst) : sputl32(nsyms, dst);

794         for (i = 0, j = SYMCHUNK, ptr = symlist; i < nsyms; i++, j--, ptr++) {
795                 if (!j) {
796                         j = SYMCHUNK;
797                         ptr = (ARFILEP *)*ptr;
798                 }
799                 dst = is64 ? sputl64((*ptr)->ar_offset, dst) :
800                     sputl32((*ptr)->ar_offset, dst);
801         }
802         arwrite(filename, fd, buf, tblsize);
```



```
803         free(buf);
804         arwrite(filename, fd, sym_strtbl.base, sym_strtbl.used);
805 }

807 /*
808  * Grow the size of the given string table so that there is room
809  * for at least need bytes.
810  *
811  * entry:
812  *      strtbl - String table to grow
813  *      need - Amount of space required by caller
814  */
815 static void
816 strtbl_alloc(ARSTRTBL *strtbl, size_t need)
817 {
818 #define STRTBL_INITSZ   8196

820         /*
821          * On 32-bit systems, we require a larger integer type in order
822          * to avoid overflow and wraparound when doing our computations.
823          */
824         uint64_t        need64 = need;
825         uint64_t        used64 = strtbl->used;
826         uint64_t        size64 = strtbl->size;
827         uint64_t        target = need64 + used64;

829         int             sys32, tbl32;

831         if (target <= size64)
832                 return;

834         /*
835          * Detect 32-bit system. We might usually do this with the preprocessor,
836          * but it can serve as a predicate in tests that also apply to 64-bit
837          * systems.
838          */
839         sys32 = (sizeof (size_t) == 4);

841         /*
842          * The symbol string table can be larger than 32-bits on a 64-bit
843          * system. However, the long name table must stay below that limit.
844          * The reason for this is that there is not enough room in the ar_name
845          * field of the member header to represent 64-bit offsets.
846          */
847         tbl32 = (strtbl == &long_strtbl);

849         /*
850          * If request is larger than 4GB and we can’t do it because we
851          * are a 32-bit program, or because the table is format limited,
852          * we can go no further.
853          */
854         if ((target > 0xffffffff) && (sys32 || tbl32))
855                 goto limit_fail;

857         /* Default starting size */
858         if (strtbl->base == NULL)
859                 size64 = STRTBL_INITSZ;

861         /*
862          * Our strategy is to double the size until we find a size that
863          * exceeds the request. However, if this table cannot exceed 4GB,
864          * then once we exceed 2GB, we switch to a strategy of taking the
865          * current request and rounding it up to STRTBL_INITSZ.
866          */
867         while (target > size64) {
868                 if ((target > 0x7fffffff) && (sys32 || tbl32)) {
```

```
869                         size64 = ((target + STRTBL_INITSZ) / STRTBL_INITSZ) *
870                             STRTBL_INITSZ;

872                         /*
873                          * If we are so close to the line that this small
874                          * increment exceeds 4GB, give it up.
875                          */
876                         if ((size64 > 0xffffffff) && (sys32 || tbl32))
877                                 goto limit_fail;

879                         break;
880                 }

882                 size64 *= 2;
883         }

885         strtbl->base = realloc(strtbl->base, size64);
886         if (strtbl->base == NULL) {
887                 int err = errno;
888                 (void) fprintf(stderr, MSG_INTL(MSG_MALLOC), strerror(err));
889                 exit(1);
890         }
891         strtbl->size = (size_t)size64;
892         return;

894 limit_fail:
895         /*
896          * Control comes here if we are unable to allocate more than 4GB of
897          * memory for the string table due to one of the following reasons:
898          *
899          * - A 32-bit process is attempting to be larger than 4GB
900          *
901          * - A 64-bit process is attempting to grow the long names string
902          *      table beyond the ar format limit of 32-bits.
903          */
904         if (sys32)
905                 (void) fprintf(stderr, MSG_INTL(MSG_MALLOC), strerror(ENOMEM));
906         else
907                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_LONGSTRTBLSZ));
908         exit(1);

910 #undef STRTBL_INITSZ
911 }

913 /*
914  * Add the specified number of pad characters to the end of the
915  * given string table.
916  *
917  * entry:
918  *      strtbl - String table to pad
919  *      n - # of pad characters to add
920  *      ch - Pad character to use
921  */
922 static void
923 strtbl_pad(ARSTRTBL *strtbl, size_t n, int ch)
924 {
925         if (n == 0)
926                 return;

928         if ((n + strtbl->used) > strtbl->size)
929                 strtbl_alloc(strtbl, n);

931         while (n--)
932                 strtbl->base[strtbl->used++] = ch;
933 }
```



```
935 /*
936  * Enter a symbol name into the symbol string table.
937  */
938 static void
939 savename(char *symbol)
940 {
941         size_t need;

943         need = strlen(symbol) + 1;
944         if ((need + sym_strtbl.used) > sym_strtbl.size)
945                 strtbl_alloc(&sym_strtbl, need);

947         (void) strcpy(sym_strtbl.base + sym_strtbl.used, symbol);
948         sym_strtbl.used += need;
949 }

951 /*
952  * Prepare an archive member with a long (>15 characters) name for
953  * the output archive.
954  *
955  * entry:
956  *      fptr - pointer to archive member with long name
957  *
958  * exit:
959  *      The long name is entered into the long name string table,
960  *      and fptr->ar_name has been replaced with the special /xxx
961  *      name used to indicate that the real name is in the string table
962  *      at offset xxx.
963  */
964 static void
965 savelongname(ARFILE *fptr)
966 {
967         size_t  len, need;
968         char    *p;

970         /* Size of new item to add */
971         len = strlen(fptr->ar_longname);
972         need = len + 2;

974         /* Ensure there's room */
975         if ((need + long_strtbl.used) > long_strtbl.size)
976                 strtbl_alloc(&long_strtbl, need);

978         /*
979          * Generate the index string to be written into the member header
980          *
981          * This will not overflow the ar_name field because that field is
982          * 16 characters in size, and a 32-bit unsigned value can be formatted
983          * in 10 characters. Allowing a character for the leading '/', and one
984          * for the NULL termination, that leaves us with 4 extra spaces.
985          */
986         (void) snprintf(fptr->ar_name, sizeof (fptr->ar_name),
987             MSG_ORIG(MSG_FMT_LLINT), EC_XWORD(long_strtbl.used));

989         /*
990          * Enter long name into reserved spot, terminated with a slash
991          * and a newline character.
992          */
993         p = long_strtbl.base + long_strtbl.used;
994         long_strtbl.used += need;
995         (void) strcpy(p, fptr->ar_longname);
996         p += len;
997         *p++ = '/';
998         *p++ = '\n';
999 }
```

```
/*
 * Determine if the archive we're about to write will exceed the
 * 32-bit limit of 4GB.
 *
 * entry:
 *      mksymtab() and mklong_tab() have been called to set up
 *      the string tables.
 *
 * exit:
 *      Returns TRUE (1) if the 64-bit symbol table is needed, and
 *      FALSE (0) otherwise.
 *
 */
static int
require64(size_t nsyms, int found_obj, size_t longnames)
{
        ARFILE          *fptr;
        uint64_t        size;

        /*
         * If there are more than 4GB symbols, we have to use
         * the 64-bit form. Note that longnames cannot exceed 4GB
         * because that symbol table is limited to a length of 4GB by
         * the archive format.
         */
        if (nsyms > 0xffffffff)
                return (1);

        /*
         * Make a worst case estimate for the size of the resulting
         * archive by assuming full padding between members.
         */
        size =  SARMAG;
        if (longnames)
                size += sizeof (struct ar_hdr) + long_strtbl.used + PADSZ;

        if (found_obj)
                size += sizeof_symtbl(nsyms, found_obj, 4) + PADSZ;

        if (size > 0xffffffff)
                return (1);

        for (fptr = listhead; fptr; fptr = fptr->ar_next) {
                size += sizeof (struct ar_hdr) + fptr->ar_size + PADSZ;

                if (size > 0xffffffff)
                        return (1);
        }

        /* 32-bit symbol table will suffice */
        return (0);
}

void
writefile(Cmd_info *cmd_info)
{
        ARFILE          *fptr;
        ARFILEP         *symlist = 0;
        size_t          longnames;
        size_t          nsyms;
        int             new_archive = 0;
        char            *name = cmd_info->arnam;
        size_t          arsize; /* Size of magic # and special members */
        size_t          symtbl_eltsize = 4;
        int             found_obj = 0;
        int             fd;
```



```
        off_t           off;
        struct stat     stbuf, ar_stbuf;
        char            pad_bytes[PADSZ];
        size_t          pad_cnt;
        int             is_elf;

        /*
         * Gather the list of symbols and associate each one to the
         * ARFILE descriptor of the object it belongs to. At the same
         * time, tag each ELF object with the appropriate F_CLASSxx
         * flag.
         */
        nsyms = mksymtab(name, &symlist, &found_obj);

        /* Generate the string table for long member names */
        longnames = mklong_tab();

        /*
         * Will this archive exceed 4GB? If we're a 32-bit process, we can't
         * do it. If we're a 64-bit process, then we'll have to use a
         * 64-bit symbol table.
         */
        if (require64(nsyms, found_obj, longnames)) {
#ifdef _LP64
                symtbl_eltsize = 8;
#else
                (void) fprintf(stderr, MSG_INTL(MSG_TOOBIG4G));
                exit(1);
#endif
        }

        /*
         * If the user requested it, use the 64-bit symbol table even if
         * a 32-bit one would suffice. 32-bit tables are more portable and
         * take up less room, so this feature is primarily for testing.
         */
        if (cmd_info->opt_flgs & S_FLAG)
                symtbl_eltsize = 8;

        /*
         * If the first non-special archive member is an ELF object, then we
         * need to arrange for its data to have an alignment of PADSZ. The
         * preceeding special member will be the symbol table, or the long
         * name string table. We pad the string table that precedes the
         * ELF member in order to achive the desired alignment.
         */
        is_elf = listhead && (listhead->ar_flag & (F_CLASS32 | F_CLASS64));
        arsize = SARMAG;
        if (found_obj) {
                arsize += sizeof_symtbl(nsyms, found_obj, symtbl_eltsize);
                if (is_elf && (longnames == 0)) {
                        pad_cnt = pad(arsize + sizeof (struct ar_hdr), PADSZ);
                        strtbl_pad(&sym_strtbl, pad_cnt, '\0');
                        arsize += pad_cnt;
                }
        }
        if (longnames > 0) {
                arsize += sizeof (struct ar_hdr) + long_strtbl.used;
                if (is_elf) {
                        pad_cnt = pad(arsize + sizeof (struct ar_hdr), PADSZ);
                        strtbl_pad(&long_strtbl, pad_cnt, '\0');
                        arsize += pad_cnt;
                }
        }

        /*
```

```
1133          * For each user visible (non-special) archive member, determine
1134          * the header offset, and the size of any required padding.
1135          */
1136         (void) sizeofmembers(arsize);

1138         /*
1139          * Is this a new archive, or are we updating an existing one?
1140          *
1141          * A subtlety here is that POSIX says we are not supposed
1142          * to replace a non-writable file. The only 100% reliable test
1143          * against this is to open the file for non-destructive
1144          * write access. If the open succeeds, we are clear to
1145          * replace it, and if not, then the error generated is
1146          * the error we need to report.
1147          */
1148         if ((fd = open(name, O_RDWR)) < 0) {
1149                 int     err = errno;

1151                 if (err != ENOENT) {
1152                         (void) fprintf(stderr, MSG_INTL(MSG_SYS_OPEN),
1153                             name, strerror(err));
1154                         exit(1);
1155                 }
1156                 new_archive = 1;
1157                 if ((cmd_info->opt_flgs & c_FLAG) == 0) {
1158                         (void) fprintf(stderr, MSG_INTL(MSG_BER_MES_CREATE),
1159                             cmd_info->arnam);
1160                 }
1161         } else {
1162                 /* Capture mode and owner information to apply to replacement */
1163                 if (fstat(fd, &ar_stbuf) < 0) {
1164                         int err = errno;
1165                         (void) fprintf(stderr, MSG_INTL(MSG_SYS_STAT),
1166                             name, strerror(err));
1167                         (void) close(fd);
1168                         exit(1);
1169                 }
1170                 (void) close(fd);
1171                 new_archive = 0;
1172         }


1175         /*
1176          * Register exit handler function to clean up after us if we exit
1177          * before completing the new archive. atexit() is defined as
1178          * only being able to fail due to memory exhaustion.
1179          */
1180         if (atexit(exit_cleanup) != 0) {
1181                 (void) fprintf(stderr, MSG_INTL(MSG_MALLOC), strerror(ENOMEM));
1182                 exit(1);
1183         }

1185         /*
1186          * If a new archive, create it in place. If updating an archive,
1187          * create the replacement under a temporary name and then rename it
1188          * into place.
1189          */
1190         ar_outfile.path = new_archive ? name : make_tmpname(name);
1191         ar_outfile.fd = open(ar_outfile.path, O_RDWR|O_CREAT|O_LARGEFILE, 0666);
1192         if (ar_outfile.fd == -1) {
1193                 int err = errno;
1194                 (void) fprintf(stderr, MSG_INTL(MSG_SYS_OPEN),
1195                     ar_outfile.path, strerror(err));
1196                 exit(1);
1197         }
```



```
1199         /* Output magic string */
1200         arwrite(name, ar_outfile.fd, ARMAG, SARMAG);

1202         /*
1203          * The symbol table member is always first if present. Note that
1204          * writesymtab() uses the member offsets computed by sizeofmembers()
1205          * above.
1206          */
1207         if (found_obj)
1208                 writesymtab(name, ar_outfile.fd, nsyms, symlist,
1209                     symtbl_eltsize);

1211         if (longnames) {
1212                 write_member_header(name, ar_outfile.fd, 0,
1213                     MSG_ORIG(MSG_STR_DSLASH), time(0), 0, 0, 0,
1214                     long_strtbl.used);
1215                 arwrite(name, ar_outfile.fd, long_strtbl.base,
1216                     long_strtbl.used);
1217         }

1219         /*
1220          * The accuracy of the symbol table depends on our having calculated
1221          * the size of the archive accurately to this point. If this is a
1222          * debug build, verify it.
1223          */
1224         assert(arsize == lseek(ar_outfile.fd, 0, SEEK_CUR));

1226 #ifndef XPG4
1227         if (cmd_info->opt_flgs & v_FLAG) {
1228                 (void) fprintf(stderr, MSG_INTL(MSG_BER_MES_WRITE),
1229                     cmd_info->arnam);
1230         }
1231 #endif

1233         /*
1234          * Fill pad_bytes array with newline characters. This array
1235          * is used to supply padding bytes at the end of ELF objects.
1236          * There can never be more tha PADSZ such bytes, so this number
1237          * will always suffice.
1238          */
1239         for (pad_cnt = 0; pad_cnt < PADSZ; pad_cnt++)
1240                 pad_bytes[pad_cnt] = '\n';

1242         for (fptr = listhead; fptr; fptr = fptr->ar_next) {
1243                 /*
1244                  * We computed the expected offset for each ELF member and
1245                  * used those offsets to fill the symbol table. If this is
1246                  * a debug build, verify that the computed offset was right.
1247                  */
1248                 is_elf = (fptr->ar_flag & (F_CLASS32 | F_CLASS64)) != 0;
1249                 assert(!is_elf ||
1250                     (fptr->ar_offset == lseek(ar_outfile.fd, 0, SEEK_CUR)));

1252                 /*
1253                  * NOTE:
1254                  * The mem_header->ar_name[] is set to a NULL string
1255                  * if the archive member header has some error.
1256                  * (See elf_getarhdr() man page.)
1257                  * It is set to NULL for example, the ar command reads
1258                  * the archive files created by SunOS 4.1 system.
1259                  * See c block comment in cmd.c, "Incompatible Archive Header".
1260                  */
1261                 if (fptr->ar_name[0] == 0) {
1262                         fptr->ar_longname = fptr->ar_rawname;
1263                         (void) strncpy(fptr->ar_name, fptr->ar_rawname, SNAME);
1264                 }
```

```
1265                write_member_header(name, ar_outfile.fd, is_elf,
1266                    (strlen(fptr->ar_longname) <= (unsigned)SNAME-2) ?
1267                    trimslash(fptr->ar_longname) : fptr->ar_name,
1268                    EC_WORD(fptr->ar_date), fptr->ar_uid, fptr->ar_gid,
1269                    fptr->ar_mode, fptr->ar_size + fptr->ar_padding);


1272                if ((fptr->ar_flag & F_ELFRAW) == 0) {
1273                        /*
1274                         * The file doesn't come from the archive, and is
1275                         * therefore not already in memory(fptr->ar_contents)
1276                         * so open it and do a direct file-to-file transfer of
1277                         * its contents. We use the sendfile() system call
1278                         * to make the kernel do the transfer, so we don't have
1279                         * to buffer data in process, and we trust that the
1280                         * kernel will use an optimal transfer strategy.
1281                         */
1282                        if ((fd = open(fptr->ar_pathname, O_RDONLY)) == -1) {
1283                                int err = errno;
1284                                (void) fprintf(stderr, MSG_INTL(MSG_SYS_OPEN),
1285                                    fptr->ar_longname, strerror(err));
1286                                exit(1);
1287                        }
1288                        if (stat(fptr->ar_pathname, &stbuf) < 0) {
1289                                int err = errno;
1290                                (void) fprintf(stderr, MSG_INTL(MSG_SYS_OPEN),
1291                                    fptr->ar_longname, strerror(err));
1292                                (void) close(fd);
1293                                exit(1);
1294                        }
1295                        off = 0;
1296                        if (sendfile(ar_outfile.fd, fd, &off,
1297                            stbuf.st_size) != stbuf.st_size) {
1298                                int err = errno;
1299                                (void) fprintf(stderr, MSG_INTL(MSG_SYS_WRITE),
1300                                    name, strerror(err));
1301                                exit(2);
1302                        }
1303                        (void) close(fd);
1304                } else {
1305                        /* Archive member is in memory. Write it out */
1306                        arwrite(name, ar_outfile.fd, fptr->ar_contents,
1307                            fptr->ar_size);
1308                }

1310                /*
1311                 * All archive members are padded to at least a boundary of 2.
1312                 * The expression ((fptr->ar_size & 0x1) != 0) yields 1 for
1313                 * odd boundaries, and 0 for even ones. To this, we add
1314                 * whatever padding is needed for ELF objects.
1315                 */
1316                pad_cnt = ((fptr->ar_size & 0x1) != 0) + fptr->ar_padding;
1317                if (pad_cnt > 0)
1318                        arwrite(name, ar_outfile.fd, pad_bytes, pad_cnt);
1319        }

1321        /*
1322         * All archive output is done.
1323         */
1324        if (close(ar_outfile.fd) < 0) {
1325                int err = errno;
1326                (void) fprintf(stderr, MSG_INTL(MSG_SYS_CLOSE), ar_outfile.path,
1327                    strerror(err));
1328                exit(1);
1329        }
1330        ar_outfile.fd = -1;     /* Prevent removal on exit */
```



```
1331        (void) elf_end(cmd_info->arf);
1332        (void) close(cmd_info->afd);

1334        /*
1335         * If updating an existing archive, rename the new version on
1336         * top of the original.
1337         */
1338        if (!new_archive) {
1339                /*
1340                 * Prevent the replacement of the original archive from
1341                 * being interrupted, to lower the possibility of an
1342                 * interrupt destroying a pre-existing archive.
1343                 */
1344                establish_sighandler(SIG_IGN);

1346                if (rename(ar_outfile.path, name) < 0) {
1347                        int err = errno;
1348                        (void) fprintf(stderr, MSG_INTL(MSG_SYS_RENAME),
1349                            ar_outfile.path, name, strerror(err));
1350                        (void) unlink(ar_outfile.path);
1351                        exit(1);
1352                }
1353                (void) chmod(name, ar_stbuf.st_mode & 0777);
1354                if (chown(name, ar_stbuf.st_uid, ar_stbuf.st_gid) >= 0)
1355                        (void) chmod(name, ar_stbuf.st_mode & 07777);

1357        }
1358 }

1360 /*
1361  * Examine all the archive members, enter any member names longer than
1362  * 15 characters into the long name string table, and count the number
1363  * of names found.
1364  *
1365  * Returns the size of the resulting archive member, including the
1366  * member header.
1367  */
1368 static size_t
1369 mklong_tab(void)
1370 {
1371        ARFILE  *fptr;
1372        size_t longnames = 0;

1374        for (fptr = listhead; fptr; fptr = fptr->ar_next) {
1375                if (strlen(fptr->ar_longname) >= (unsigned)SNAME-1) {
1376                        longnames++;
1377                        savelongname(fptr);
1378                }
1379        }

1381        /* round up table that keeps the long filenames */
1382        if (longnames > 0)
1383                strtbl_pad(&long_strtbl, pad(long_strtbl.used, 4), '\n');

1385        return (longnames);
1386 }

1388 /*
1389  * Write 32/64-bit words into buffer in archive symbol table
1390  * standard byte order (MSB).
1391  */
1392 static char *
1393 sputl32(uint32_t n, char *cp)
1394 {
1395        *cp++ = n >> 24;
1396        *cp++ = n >> 16;
```

```
1397         *cp++ = n >> 8;

1399         *cp++ = n & 255;

1401         return (cp);
1402 }

1404 static char *
1405 sputl64(uint64_t n, char *cp)
1406 {
1407         *cp++ = n >> 56;
1408         *cp++ = n >> 48;
1409         *cp++ = n >> 40;
1410         *cp++ = n >> 32;

1412         *cp++ = n >> 24;
1413         *cp++ = n >> 16;
1414         *cp++ = n >> 8;

1416         *cp++ = n & 255;

1418         return (cp);
1419 }

1421 static int
1422 search_sym_tab(const char *arname, ARFILE *fptr, Elf *elf, Elf_Scn *scn,
1423         size_t *nsyms, ARFILEP **symlist, size_t *num_errs)
1424 {
1425         Elf_Data *str_data, *sym_data; /* string table, symbol table */
1426         Elf_Scn *str_scn;
1427         GElf_Sxword no_of_symbols;
1428         GElf_Shdr shdr;
1429         int counter;
1430         int str_shtype;
1431         char *symname;
1432         static ARFILEP *sym_ptr = 0;
1433         static ARFILEP *nextsym = NULL;
1434         static int syms_left = 0;
1435         char *fname = fptr->ar_pathname;

1437         (void) gelf_getshdr(scn, &shdr);
1438         str_scn = elf_getscn(elf, shdr.sh_link); /* index for string table */
1439         if (str_scn == NULL) {
1440                 if (fname != NULL)
1441                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_GETDATA_FILE),
1442                             fname, elf_errmsg(-1));
1443                 else
1444                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_GETDATA_AR),
1445                             arname, fptr->ar_longname, elf_errmsg(-1));
1446                 (*num_errs)++;
1447                 return (-1);
1448         }

1450         no_of_symbols = shdr.sh_size / shdr.sh_entsize;
1451         if (no_of_symbols == -1) {
1452                 (void) fprintf(stderr, MSG_INTL(MSG_SYMTAB_01));
1453                 return (-1);
1454         }

1456         (void) gelf_getshdr(str_scn, &shdr);
1457         str_shtype = shdr.sh_type;
1458         if (str_shtype == -1) {
1459                 if (fname != NULL)
1460                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_GETDATA_FILE),
1461                             fname, elf_errmsg(-1));
1462                 else
```



```
1463                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_GETDATA_AR),
1464                             arname, fptr->ar_longname, elf_errmsg(-1));
1465                 (*num_errs)++;
1466                 return (-1);
1467         }

1469         /* This test must happen before testing the string table. */
1470         if (no_of_symbols == 1)
1471                 return (0);     /* no symbols; 0th symbol is the non-symbol */

1473         if (str_shtype != SHT_STRTAB) {
1474                 if (fname != NULL)
1475                         (void) fprintf(stderr, MSG_INTL(MSG_SYMTAB_NOSTR_FILE),
1476                             fname);
1477                 else
1478                         (void) fprintf(stderr, MSG_INTL(MSG_SYMTAB_NOSTR_AR),
1479                             arname, fptr->ar_longname);
1480                 return (0);
1481         }
1482         str_data = 0;
1483         if ((str_data = elf_getdata(str_scn, str_data)) == 0) {
1484                 if (fname != NULL)
1485                         (void) fprintf(stderr, MSG_INTL(MSG_SYMTAB_NODAT_FILE),
1486                             fname);
1487                 else
1488                         (void) fprintf(stderr, MSG_INTL(MSG_SYMTAB_NODAT_AR),
1489                             arname, fptr->ar_longname);
1490                 return (0);
1491         }
1492         if (str_data->d_size == 0) {
1493                 if (fname != NULL)
1494                         (void) fprintf(stderr, MSG_INTL(MSG_SYMTAB_ZDAT_FILE),
1495                             fname);
1496                 else
1497                         (void) fprintf(stderr, MSG_INTL(MSG_SYMTAB_ZDAT_AR),
1498                             arname, fptr->ar_longname);
1499                 return (0);
1500         }
1501         sym_data = 0;
1502         if ((sym_data = elf_getdata(scn, sym_data)) == NULL) {
1503                 if (fname != NULL)
1504                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_LIB_FILE),
1505                             fname, elf_errmsg(-1));
1506                 else
1507                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_LIB_AR),
1508                             arname, fptr->ar_longname, elf_errmsg(-1));
1509                 return (0);
1510         }

1512         /* start at 1, first symbol entry is ignored */
1513         for (counter = 1; counter < no_of_symbols; counter++) {
1514                 GElf_Sym sym;
1515                 (void) gelf_getsym(sym_data, counter, &sym);

1517                 symname = (char *)(str_data->d_buf) + sym.st_name;

1519                 if (((GELF_ST_BIND(sym.st_info) == STB_GLOBAL) ||
1520                     (GELF_ST_BIND(sym.st_info) == STB_WEAK)) &&
1521                     (sym.st_shndx != SHN_UNDEF)) {
1522                         if (!syms_left) {
1523                                 sym_ptr = malloc((SYMCHUNK+1)
1524                                     * sizeof (ARFILEP));
1525                                 if (sym_ptr == NULL) {
1526                                         int err = errno;
1527                                         (void) fprintf(stderr,
1528                                             MSG_INTL(MSG_MALLOC),
```

```
1529                                            strerror(err));
1530                                    exit(1);
1531                            }
1532                            syms_left = SYMCHUNK;
1533                            if (nextsym)
1534                                    *nextsym = (ARFILEP)sym_ptr;
1535                            else
1536                                    *symlist = sym_ptr;
1537                            nextsym = sym_ptr;
1538                    }
1539                    sym_ptr = nextsym;
1540                    nextsym++;
1541                    syms_left--;
1542                    (*nsyms)++;
1543                    *sym_ptr = fptr;
1544                    savename(symname);      /* put name in the archiver's */
1545                                            /* symbol table string table */
1546            }
1547    }
1548    return (0);
1549 }

1551 /*
1552  * Get the output file size
1553  */
1554 static size_t
1555 sizeofmembers(size_t psum)
1556 {
1557    size_t  sum = 0;
1558    ARFILE  *fptr;
1559    size_t  hdrsize = sizeof (struct ar_hdr);

1561    for (fptr = listhead; fptr; fptr = fptr->ar_next) {
1562            fptr->ar_offset = psum + sum;
1563            sum += fptr->ar_size;
1564            if (fptr->ar_size & 01)
1565                    sum++;
1566            sum += hdrsize;

1568            /*
1569             * If the current item, and the next item are both ELF
1570             * objects, then add padding to current item so that the
1571             * data in the next item will have PADSZ alignment.
1572             *
1573             * In any other case, set the padding to 0. If the
1574             * item comes from another archive, it may be carrying
1575             * a non-zero padding value from that archive that does
1576             * not apply to the one we are about to build.
1577             */
1578            if ((fptr->ar_flag & (F_CLASS32 | F_CLASS64)) &&
1579                fptr->ar_next &&
1580                (fptr->ar_next->ar_flag & (F_CLASS32 | F_CLASS64))) {
1581                    fptr->ar_padding = pad(psum + sum + hdrsize, PADSZ);
1582                    sum += fptr->ar_padding;
1583            } else {
1584                    fptr->ar_padding = 0;
1585            }
1586    }
1587    return (sum);
1588 }

1590 /*
1591  * Compute the size of the symbol table archive member.
1592  *
1593  * entry:
1594  *      nsyms - # of symbols in the table
```



```
1595  *      found_obj - TRUE if the archive contains any ELF objects
1596  *      eltsize - Size of the integer type to use for the symbol
1597  *              table. 4 for 32-bit tables, and 8 for 64-bit tables.
1598  */
1599 static size_t
1600 sizeof_symtbl(size_t nsyms, int found_obj, size_t eltsize)
1601 {
1602    size_t sum = 0;

1604    if (found_obj) {
1605            /* Member header, symbol count, and one slot per symbol */
1606            sum += sizeof (struct ar_hdr) + ((nsyms + 1) * eltsize);
1607            sum += sym_strtbl.used;
1608    }

1610    return (sum);
1611 }

1613 static void
1614 arwrite(const char *name, int nfd, const char *dst, size_t size) {
1615    if (write(nfd, dst, size) != size) {
1616            int err = errno;
1617            (void) fprintf(stderr, MSG_INTL(MSG_SYS_WRITE),
1618                name, strerror(err));
1619            exit(2);
1620    }
1621 }

1623 static const char *
1624 make_tmpname(const char *filename) {
1625    char    *slash, *tmpname;
1626    size_t  prefix_cnt = 0;

1628    /*
1629     * If there is a path prefix in front of the filename, we
1630     * want to put the temporary file in the same directory.
1631     * Determine the length of the path.
1632     */
1633    slash = strrchr(filename, '/');
1634    if (slash != NULL)
1635            prefix_cnt = slash - filename + 1;
1636    tmpname = malloc(prefix_cnt + MSG_STR_MKTEMP_SIZE + 1);
1637    if (tmpname == NULL) {
1638            int err = errno;
1639            (void) fprintf(stderr, MSG_INTL(MSG_MALLOC), strerror(err));
1640            exit(1);
1641    }

1643    if (prefix_cnt > 0)
1644            (void) strncpy(tmpname, filename, prefix_cnt);
1645    (void) strcpy(tmpname + prefix_cnt, MSG_ORIG(MSG_STR_MKTEMP));
1646    (void) mktemp(tmpname);

1648    return (tmpname);
1649 }
```

```
**********************************************************
   95301 Fri Aug  9 16:25:49 2013
new/usr/src/cmd/sgs/libld/common/syms.c
3999 libld extended section handling is broken
**********************************************************
______unchanged_portion_omitted_

1888 /*
1889  * Process the symbol table for the specified input file.  At this point all
1890  * input sections from this input file have been assigned an input section
1891  * descriptor which is saved in the 'ifl_isdesc' array.
1892  *
1893  *  -   local symbols are saved (as is) if the input file is a  relocatable
1894  *      object
1895  *
1896  *  -   global symbols are added to the linkers internal symbol table if they
1897  *      are not already present, otherwise a symbol resolution function is
1898  *      called upon to resolve the conflict.
1899  */
1900 uintptr_t
1901 ld_sym_process(Is_desc *isc, Ifl_desc *ifl, Ofl_desc *ofl)
1902 {
1903         /*
1904          * This macro tests the given symbol to see if it is out of
1905          * range relative to the section it references.
1906          *
1907          * entry:
1908          *      - ifl is a relative object (ET_REL)
1909          *      _sdp - Symbol descriptor
1910          *      _sym - Symbol
1911          *      _type - Symbol type
1912          *
1913          * The following are tested:
1914          *      - Symbol length is non-zero
1915          *      - Symbol type is a type that references code or data
1916          *      - Referenced section is not 0 (indicates an UNDEF symbol)
1917          *        and is not in the range of special values above SHN_LORESERVE
1918          *        (excluding SHN_XINDEX, which is OK).
1919          *      - We have a valid section header for the target section
1920          *
1921          * If the above are all true, and the symbol position is not
1922          * contained by the target section, this macro evaluates to
1923          * True (1). Otherwise, False(0).
1924          */
1925 #define SYM_LOC_BADADDR(_sdp, _sym, _type) \
1926         (_sym->st_size && dynsymsort_symtype[_type] && \
1927         (_sym->st_shndx != SHN_UNDEF) && \
1928         ((_sym->st_shndx < SHN_LORESERVE) || \
1929                 (_sym->st_shndx == SHN_XINDEX)) && \
1930         _sdp->sd_isc && _sdp->sd_isc->is_shdr && \
1931         ((_sym->st_value + _sym->st_size) > _sdp->sd_isc->is_shdr->sh_size))

1933         Conv_inv_buf_t  inv_buf;
1934         Sym             *sym = (Sym *)isc->is_indata->d_buf;
1935         Word            *symshndx = NULL;
1936         Shdr            *shdr = isc->is_shdr;
1937         Sym_desc        *sdp;
1938         size_t          strsize;
1939         char            *strs;
1940         uchar_t         type, bind;
1941         Word            ndx, hash, local, total;
1942         uchar_t         osabi = ifl->ifl_ehdr->e_ident[EI_OSABI];
1943         Half            mach = ifl->ifl_ehdr->e_machine;
1944         Half            etype = ifl->ifl_ehdr->e_type;
1945         int             etype_rel;
1946         const char      *symsecname, *strsecname;
```



```
1947         Word            symsecndx;
1948         avl_index_t     where;
1949         int             test_gnu_hidden_bit, weak;
1950         Cap_desc        *cdp = NULL;
1951         Alist           *cappairs = NULL;

1953         /*
1954          * Its possible that a file may contain more that one symbol table,
1955          * ie. .dynsym and .symtab in a shared library.  Only process the first
1956          * table (here, we assume .dynsym comes before .symtab).
1957          */
1958         if (ifl->ifl_symscnt)
1959                 return (1);

1961         if (isc->is_symshndx)
1962                 symshndx = isc->is_symshndx->is_indata->d_buf;

1964         DBG_CALL(Dbg_syms_process(ofl->ofl_lml, ifl));

1966         symsecndx = isc->is_scnndx;
1967         if (isc->is_name)
1968                 symsecname = isc->is_name;
1969         else
1970                 symsecname = MSG_ORIG(MSG_STR_EMPTY);

1972         /*
1973          * From the symbol tables section header information determine which
1974          * strtab table is needed to locate the actual symbol names.
1975          */
1976         if (ifl->ifl_flags & FLG_IF_HSTRTAB) {
1977                 ndx = shdr->sh_link;
1978                 if ((ndx == 0) || (ndx >= ifl->ifl_shnum)) {
1979                         ld_eprintf(ofl, ERR_FATAL,
1980                             MSG_INTL(MSG_FIL_INVSHLINK), ifl->ifl_name,
1981                             EC_WORD(symsecndx), symsecname, EC_XWORD(ndx));
1982                         return (S_ERROR);
1983                 }
1984                 strsize = ifl->ifl_isdesc[ndx]->is_shdr->sh_size;
1985                 strs = ifl->ifl_isdesc[ndx]->is_indata->d_buf;
1986                 if (ifl->ifl_isdesc[ndx]->is_name)
1987                         strsecname = ifl->ifl_isdesc[ndx]->is_name;
1988                 else
1989                         strsecname = MSG_ORIG(MSG_STR_EMPTY);
1990         } else {
1991                 /*
1992                  * There is no string table section in this input file
1993                  * although there are symbols in this symbol table section.
1994                  * This means that these symbols do not have names.
1995                  * Currently, only scratch register symbols are allowed
1996                  * not to have names.
1997                  */
1998                 strsize = 0;
1999                 strs = (char *)MSG_ORIG(MSG_STR_EMPTY);
2000                 strsecname = MSG_ORIG(MSG_STR_EMPTY);
2001         }

2003         /*
2004          * Determine the number of local symbols together with the total
2005          * number we have to process.
2006          */
2007         total = (Word)(shdr->sh_size / shdr->sh_entsize);
2008         local = shdr->sh_info;

2010         /*
2011          * Allocate a symbol table index array and a local symbol array
2012          * (global symbols are processed and added to the ofl->ofl_symbkt[]
```

```
2013          * array).  If we are dealing with a relocatable object, allocate the
2014          * local symbol descriptors.  If this isn't a relocatable object we
2015          * still have to process any shared object locals to determine if any
2016          * register symbols exist.  Although these aren't added to the output
2017          * image, they are used as part of symbol resolution.
2018          */
2019         if ((ifl->ifl_oldndx = libld_malloc((size_t)(total *
2020             sizeof (Sym_desc *)))) == NULL)
2021                 return (S_ERROR);
2022         etype_rel = (etype == ET_REL);
2023         if (etype_rel && local) {
2024                 if ((ifl->ifl_locs =
2025                     libld_calloc(sizeof (Sym_desc), local)) == NULL)
2026                         return (S_ERROR);
2027                 /* LINTED */
2028                 ifl->ifl_locscnt = (Word)local;
2029         }
2030         ifl->ifl_symscnt = total;

2032         /*
2033          * If there are local symbols to save add them to the symbol table
2034          * index array.
2035          */
2036         if (local) {
2037                 int             allow_ldynsym = OFL_ALLOW_LDYNSYM(ofl);
2038                 Sym_desc        *last_file_sdp = NULL;
2039                 int             last_file_ndx = 0;

2041                 for (sym++, ndx = 1; ndx < local; sym++, ndx++) {
2042                         sd_flag_t       sdflags = FLG_SY_CLEAN;
2043                         Word            shndx;
2044                         const char      *name;
2045                         Sym_desc        *rsdp;
2046                         int             shndx_bad = 0;
2047                         int             symtab_enter = 1;

2049                         /*
2050                          * Determine and validate the associated section index.
2051                          */
2052                         if (symshndx && (sym->st_shndx == SHN_XINDEX)) {
2053                                 shndx = symshndx[ndx];
2054                         } else if ((shndx = sym->st_shndx) >= SHN_LORESERVE) {
2055                                 sdflags |= FLG_SY_SPECSEC;
2056                         } else if (shndx > ifl->ifl_shnum) {
2056                         } else if (shndx > ifl->ifl_ehdr->e_shnum) {
2057                                 /* We need the name before we can issue error */
2058                                 shndx_bad = 1;
2059                         }

2061                         /*
2062                          * Check if st_name has a valid value or not.
2063                          */
2064                         if ((name = string(ofl, ifl, sym, strs, strsize, ndx,
2065                             shndx, symsecndx, symsecname, strsecname,
2066                             &sdflags)) == NULL)
2067                                 continue;

2069                         /*
2070                          * Now that we have the name, if the section index
2071                          * was bad, report it.
2072                          */
2073                         if (shndx_bad) {
2074                                 ld_eprintf(ofl, ERR_WARNING,
2075                                     MSG_INTL(MSG_SYM_INVSHNDX),
2076                                     demangle_symname(name, symsecname, ndx),
2077                                     ifl->ifl_name,
```



```
2078                                     conv_sym_shndx(osabi, mach, sym->st_shndx,
2079                                     CONV_FMT_DECIMAL, &inv_buf));
2080                                 continue;
2081                         }

2083                         /*
2084                          * If this local symbol table originates from a shared
2085                          * object, then we're only interested in recording
2086                          * register symbols.  As local symbol descriptors aren't
2087                          * allocated for shared objects, one will be allocated
2088                          * to associated with the register symbol.  This symbol
2089                          * won't become part of the output image, but we must
2090                          * process it to test for register conflicts.
2091                          */
2092                         rsdp = sdp = NULL;
2093                         if (sdflags & FLG_SY_REGSYM) {
2094                                 /*
2095                                  * The presence of FLG_SY_REGSYM means that
2096                                  * the pointers in ld_targ.t_ms are non-NULL.
2097                                  */
2098                                 rsdp = (*ld_targ.t_ms.ms_reg_find)(sym, ofl);
2099                                 if (rsdp != 0) {
2100                                         /*
2101                                          * The fact that another register def-
2102                                          * inition has been found is fatal.
2103                                          * Call the verification routine to get
2104                                          * the error message and move on.
2105                                          */
2106                                         (void) (*ld_targ.t_ms.ms_reg_check)
2107                                             (rsdp, sym, name, ifl, ofl);
2108                                         continue;
2109                                 }

2111                                 if (etype == ET_DYN) {
2112                                         if ((sdp = libld_calloc(
2113                                             sizeof (Sym_desc), 1)) == NULL)
2114                                                 return (S_ERROR);
2115                                         sdp->sd_ref = REF_DYN_SEEN;

2117                                         /* Will not appear in output object */
2118                                         symtab_enter = 0;
2119                                 }
2120                         } else if (etype == ET_DYN)
2121                                 continue;

2123                         /*
2124                          * Fill in the remaining symbol descriptor information.
2125                          */
2126                         if (sdp == NULL) {
2127                                 sdp = &(ifl->ifl_locs[ndx]);
2128                                 sdp->sd_ref = REF_REL_NEED;
2129                                 sdp->sd_symndx = ndx;
2130                         }
2131                         if (rsdp == NULL) {
2132                                 sdp->sd_name = name;
2133                                 sdp->sd_sym = sym;
2134                                 sdp->sd_shndx = shndx;
2135                                 sdp->sd_flags = sdflags;
2136                                 sdp->sd_file = ifl;
2137                                 ifl->ifl_oldndx[ndx] = sdp;
2138                         }

2140                         DBG_CALL(Dbg_syms_entry(ofl->ofl_lml, ndx, sdp));

2142                         /*
2143                          * Reclassify any SHN_SUNW_IGNORE symbols to SHN_UNDEF
```

```
2144                     * so as to simplify future processing.
2145                     */
2146                    if (sym->st_shndx == SHN_SUNW_IGNORE) {
2147                            sdp->sd_shndx = shndx = SHN_UNDEF;
2148                            sdp->sd_flags |= (FLG_SY_IGNORE | FLG_SY_ELIM);
2149                    }

2151                    /*
2152                     * Process any register symbols.
2153                     */
2154                    if (sdp->sd_flags & FLG_SY_REGSYM) {
2155                            /*
2156                             * Add a diagnostic to indicate we've caught a
2157                             * register symbol, as this can be useful if a
2158                             * register conflict is later discovered.
2159                             */
2160                            DBG_CALL(Dbg_syms_entered(ofl, sym, sdp));

2162                            /*
2163                             * If this register symbol hasn't already been
2164                             * recorded, enter it now.
2165                             *
2166                             * The presence of FLG_SY_REGSYM means that
2167                             * the pointers in ld_targ.t_ms are non-NULL.
2168                             */
2169                            if ((rsdp == NULL) &&
2170                                ((*ld_targ.t_ms.ms_reg_enter)(sdp, ofl) ==
2171                                0))
2172                                    return (S_ERROR);
2173                    }

2175                    /*
2176                     * Assign an input section.
2177                     */
2178                    if ((sym->st_shndx != SHN_UNDEF) &&
2179                        ((sdp->sd_flags & FLG_SY_SPECSEC) == 0))
2180                            sdp->sd_isc = ifl->ifl_isdesc[shndx];

2182                    /*
2183                     * If this symbol falls within the range of a section
2184                     * being discarded, then discard the symbol itself.
2185                     * There is no reason to keep this local symbol.
2186                     */
2187                    if (sdp->sd_isc &&
2188                        (sdp->sd_isc->is_flags & FLG_IS_DISCARD)) {
2189                            sdp->sd_flags |= FLG_SY_ISDISC;
2190                            DBG_CALL(Dbg_syms_discarded(ofl->ofl_lml, sdp));
2191                            continue;
2192                    }

2194                    /*
2195                     * Skip any section symbols as new versions of these
2196                     * will be created.
2197                     */
2198                    if ((type = ELF_ST_TYPE(sym->st_info)) == STT_SECTION) {
2199                            if (sym->st_shndx == SHN_UNDEF) {
2200                                    ld_eprintf(ofl, ERR_WARNING,
2201                                        MSG_INTL(MSG_SYM_INVSHNDX),
2202                                        demangle_symname(name, symsecname,
2203                                        ndx), ifl->ifl_name,
2204                                        conv_sym_shndx(osabi, mach,
2205                                        sym->st_shndx, CONV_FMT_DECIMAL,
2206                                        &inv_buf));
2207                            }
2208                            continue;
2209                    }
```



```
2211                    /*
2212                     * For a relocatable object, if this symbol is defined
2213                     * and has non-zero length and references an address
2214                     * within an associated section, then check its extents
2215                     * to make sure the section boundaries encompass it.
2216                     * If they don't, the ELF file is corrupt.
2217                     */
2218                    if (etype_rel) {
2219                            if (SYM_LOC_BADADDR(sdp, sym, type)) {
2220                                    issue_badaddr_msg(ifl, ofl, sdp,
2221                                        sym, shndx);
2222                                    if (ofl->ofl_flags & FLG_OF_FATAL)
2223                                            continue;
2224                            }

2226                            /*
2227                             * We have observed relocatable objects
2228                             * containing identical adjacent STT_FILE
2229                             * symbols. Discard any other than the first,
2230                             * as they are all equivalent and the extras
2231                             * do not add information.
2232                             *
2233                             * For the purpose of this test, we assume
2234                             * that only the symbol type and the string
2235                             * table offset (st_name) matter.
2236                             */
2237                            if (type == STT_FILE) {
2238                                    int toss = (last_file_sdp != NULL) &&
2239                                        ((ndx - 1) == last_file_ndx) &&
2240                                        (sym->st_name ==
2241                                        last_file_sdp->sd_sym->st_name);

2243                                    last_file_sdp = sdp;
2244                                    last_file_ndx = ndx;
2245                                    if (toss) {
2246                                            sdp->sd_flags |= FLG_SY_INVALID;
2247                                            DBG_CALL(Dbg_syms_dup_discarded(
2248                                                ofl->ofl_lml, ndx, sdp));
2249                                            continue;
2250                                    }
2251                            }
2252                    }


2255                    /*
2256                     * Sanity check for TLS
2257                     */
2258                    if ((sym->st_size != 0) && ((type == STT_TLS) &&
2259                        (sym->st_shndx != SHN_COMMON))) {
2260                            Is_desc *isp = sdp->sd_isc;

2262                            if ((isp == NULL) || (isp->is_shdr == NULL) ||
2263                                ((isp->is_shdr->sh_flags & SHF_TLS) == 0)) {
2264                                    ld_eprintf(ofl, ERR_FATAL,
2265                                        MSG_INTL(MSG_SYM_TLS),
2266                                        demangle(sdp->sd_name),
2267                                        ifl->ifl_name);
2268                                    continue;
2269                            }
2270                    }

2272                    /*
2273                     * Carry our some basic sanity checks (these are just
2274                     * some of the erroneous symbol entries we've come
2275                     * across, there's probably a lot more).  The symbol
```

```
2276                         * will not be carried forward to the output file, which
2277                         * won't be a problem unless a relocation is required
2278                         * against it.
2279                         */
2280                        if (((sdp->sd_flags & FLG_SY_SPECSEC) &&
2281                            ((sym->st_shndx == SHN_COMMON)) ||
2282                            ((type == STT_FILE) &&
2283                            (sym->st_shndx != SHN_ABS))) ||
2284                            (sdp->sd_isc && (sdp->sd_isc->is_osdesc == NULL))) {
2285                                ld_eprintf(ofl, ERR_WARNING,
2286                                    MSG_INTL(MSG_SYM_INVSHNDX),
2287                                    demangle_symname(name, symsecname, ndx),
2288                                    ifl->ifl_name,
2289                                    conv_sym_shndx(osabi, mach, sym->st_shndx,
2290                                    CONV_FMT_DECIMAL, &inv_buf));
2291                                sdp->sd_isc = NULL;
2292                                sdp->sd_flags |= FLG_SY_INVALID;
2293                                continue;
2294                        }

2296                        /*
2297                         * As these local symbols will become part of the output
2298                         * image, record their number and name string size.
2299                         * Globals are counted after all input file processing
2300                         * (and hence symbol resolution) is complete during
2301                         * sym_validate().
2302                         */
2303                        if (!(ofl->ofl_flags & FLG_OF_REDLSYM) &&
2304                            symtab_enter) {
2305                                ofl->ofl_locscnt++;

2307                                if ((((sdp->sd_flags & FLG_SY_REGSYM) == 0) ||
2308                                    sym->st_name) && (st_insert(ofl->ofl_strtab,
2309                                    sdp->sd_name) == -1))
2310                                        return (S_ERROR);

2312                                if (allow_ldynsym && sym->st_name &&
2313                                    ldynsym_symtype[type]) {
2314                                        ofl->ofl_dynlocscnt++;
2315                                        if (st_insert(ofl->ofl_dynstrtab,
2316                                            sdp->sd_name) == -1)
2317                                                return (S_ERROR);
2318                                        /* Include it in sort section? */
2319                                        DYNSORT_COUNT(sdp, sym, type, ++);
2320                                }
2321                        }
2322                }
2323        }

2325        /*
2326         * The GNU ld interprets the top bit of the 16-bit Versym value
2327         * (0x8000) as the "hidden" bit. If this bit is set, the linker
2328         * is supposed to act as if that symbol does not exist. The Solaris
2329         * linker does not support this mechanism, or the model of interface
2330         * evolution that it allows, but we honor it in GNU ld produced
2331         * objects in order to interoperate with them.
2332         *
2333         * Determine if we should honor the GNU hidden bit for this file.
2334         */
2335        test_gnu_hidden_bit = ((ifl->ifl_flags & FLG_IF_GNUVER) != 0) &&
2336            (ifl->ifl_versym != NULL);

2338        /*
2339         * Determine whether object capabilities for this file are being
2340         * converted into symbol capabilities.  If so, global function symbols,
2341         * and initialized global data symbols, need special translation and
```



```
2342         * processing.
2343         */
2344        if ((etype == ET_REL) && (ifl->ifl_flags & FLG_IF_OTOSCAP))
2345                cdp = ifl->ifl_caps;

2347        /*
2348         * Now scan the global symbols entering them in the internal symbol
2349         * table or resolving them as necessary.
2350         */
2351        sym = (Sym *)isc->is_indata->d_buf;
2352        sym += local;
2353        weak = 0;
2354        /* LINTED */
2355        for (ndx = (int)local; ndx < total; sym++, ndx++) {
2356                const char      *name;
2357                sd_flag_t       sdflags = 0;
2358                Word            shndx;
2359                int             shndx_bad = 0;
2360                Sym             *nsym = sym;
2361                Cap_pair        *cpp = NULL;
2362                uchar_t         ntype;

2364                /*
2365                 * Determine and validate the associated section index.
2366                 */
2367                if (symshndx && (nsym->st_shndx == SHN_XINDEX)) {
2368                        shndx = symshndx[ndx];
2369                } else if ((shndx = nsym->st_shndx) >= SHN_LORESERVE) {
2370                        sdflags |= FLG_SY_SPECSEC;
2371                } else if (shndx > ifl->ifl_shnum) {
2371                } else if (shndx > ifl->ifl_ehdr->e_shnum) {
2372                        /* We need the name before we can issue error */
2373                        shndx_bad = 1;
2374                }

2376                /*
2377                 * Check if st_name has a valid value or not.
2378                 */
2379                if ((name = string(ofl, ifl, nsym, strs, strsize, ndx, shndx,
2380                    symsecndx, symsecname, strsecname, &sdflags)) == NULL)
2381                        continue;

2383                /*
2384                 * Now that we have the name, report an erroneous section index.
2385                 */
2386                if (shndx_bad) {
2387                        ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_INVSHNDX),
2388                            demangle_symname(name, symsecname, ndx),
2389                            ifl->ifl_name,
2390                            conv_sym_shndx(osabi, mach, nsym->st_shndx,
2391                            CONV_FMT_DECIMAL, &inv_buf));
2392                        continue;
2393                }

2395                /*
2396                 * Test for the GNU hidden bit, and ignore symbols that
2397                 * have it set.
2398                 */
2399                if (test_gnu_hidden_bit &&
2400                    ((ifl->ifl_versym[ndx] & 0x8000) != 0))
2401                        continue;

2403                /*
2404                 * The linker itself will generate symbols for _end, _etext,
2405                 * _edata, _DYNAMIC and _PROCEDURE_LINKAGE_TABLE_, so don't
2406                 * bother entering these symbols from shared objects.  This
```

```
2407                 * results in some wasted resolution processing, which is hard
2408                 * to feel, but if nothing else, pollutes diagnostic relocation
2409                 * output.
2410                 */
2411                if (name[0] && (etype == ET_DYN) && (nsym->st_size == 0) &&
2412                    (ELF_ST_TYPE(nsym->st_info) == STT_OBJECT) &&
2413                    (name[0] == '_') && ((name[1] == 'e') ||
2414                    (name[1] == 'D') || (name[1] == 'P')) &&
2415                    ((strcmp(name, MSG_ORIG(MSG_SYM_ETEXT_U)) == 0) ||
2416                    (strcmp(name, MSG_ORIG(MSG_SYM_EDATA_U)) == 0) ||
2417                    (strcmp(name, MSG_ORIG(MSG_SYM_END_U)) == 0) ||
2418                    (strcmp(name, MSG_ORIG(MSG_SYM_DYNAMIC_U)) == 0) ||
2419                    (strcmp(name, MSG_ORIG(MSG_SYM_PLKTBL_U)) == 0))) {
2420                        ifl->ifl_oldndx[ndx] = 0;
2421                        continue;
2422                }

2424                /*
2425                 * The '-z wrap=XXX' option emulates the GNU ld --wrap=XXX
2426                 * option. When XXX is the symbol to be wrapped:
2427                 *
2428                 *  -   An undefined reference to XXX is converted to __wrap_XXX
2429                 *  -   An undefined reference to __real_XXX is converted to XXX
2430                 *
2431                 * The idea is that the user can supply a wrapper function
2432                 * __wrap_XXX that does some work, and then uses the name
2433                 * __real_XXX to pass the call on to the real function. The
2434                 * wrapper objects are linked with the original unmodified
2435                 * objects to produce a wrapped version of the output object.
2436                 */
2437                if (ofl->ofl_wrap && name[0] && (shndx == SHN_UNDEF)) {
2438                        WrapSymNode wsn, *wsnp;

2440                        /*
2441                         * If this is the __real_XXX form, advance the
2442                         * pointer to reference the wrapped name.
2443                         */
2444                        wsn.wsn_name = name;
2445                        if ((*name == '_') &&
2446                            (strncmp(name, MSG_ORIG(MSG_STR_UU_REAL_U),
2447                            MSG_STR_UU_REAL_U_SIZE) == 0))
2448                                wsn.wsn_name += MSG_STR_UU_REAL_U_SIZE;

2450                        /*
2451                         * Is this symbol in the wrap AVL tree? If so, map
2452                         * XXX to __wrap_XXX, and __real_XXX to XXX. Note that
2453                         * wsn.wsn_name will equal the current value of name
2454                         * if the __real_ prefix is not present.
2455                         */
2456                        if ((wsnp = avl_find(ofl->ofl_wrap, &wsn, 0)) != NULL) {
2457                                const char *old_name = name;

2459                                name = (wsn.wsn_name == name) ?
2460                                    wsnp->wsn_wrapname : wsn.wsn_name;
2461                                DBG_CALL(Dbg_syms_wrap(ofl->ofl_lml, ndx,
2462                                    old_name, name));
2463                        }
2464                }

2466                /*
2467                 * Determine and validate the symbols binding.
2468                 */
2469                bind = ELF_ST_BIND(nsym->st_info);
2470                if ((bind != STB_GLOBAL) && (bind != STB_WEAK)) {
2471                        ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_NONGLOB),
2472                            demangle_symname(name, symsecname, ndx),
```



```
2473                            ifl->ifl_name,
2474                            conv_sym_info_bind(bind, 0, &inv_buf));
2475                        continue;
2476                }
2477                if (bind == STB_WEAK)
2478                        weak++;

2480                /*
2481                 * If this symbol falls within the range of a section being
2482                 * discarded, then discard the symbol itself.
2483                 */
2484                if (((sdflags & FLG_SY_SPECSEC) == 0) &&
2485                    (nsym->st_shndx != SHN_UNDEF)) {
2486                        Is_desc *isp;

2488                        if (shndx >= ifl->ifl_shnum) {
2489                                /*
2490                                 * Carry our some basic sanity checks
2491                                 * The symbol will not be carried forward to
2492                                 * the output file, which won't be a problem
2493                                 * unless a relocation is required against it.
2494                                 */
2495                                ld_eprintf(ofl, ERR_WARNING,
2496                                    MSG_INTL(MSG_SYM_INVSHNDX),
2497                                    demangle_symname(name, symsecname, ndx),
2498                                    ifl->ifl_name,
2499                                    conv_sym_shndx(osabi, mach, nsym->st_shndx,
2500                                    CONV_FMT_DECIMAL, &inv_buf));
2501                                continue;
2502                        }

2504                        isp = ifl->ifl_isdesc[shndx];
2505                        if (isp && (isp->is_flags & FLG_IS_DISCARD)) {
2506                                if ((sdp =
2507                                    libld_calloc(sizeof (Sym_desc), 1)) == NULL)
2508                                        return (S_ERROR);

2510                                /*
2511                                 * Create a dummy symbol entry so that if we
2512                                 * find any references to this discarded symbol
2513                                 * we can compensate.
2514                                 */
2515                                sdp->sd_name = name;
2516                                sdp->sd_sym = nsym;
2517                                sdp->sd_file = ifl;
2518                                sdp->sd_isc = isp;
2519                                sdp->sd_flags = FLG_SY_ISDISC;
2520                                ifl->ifl_oldndx[ndx] = sdp;

2522                                DBG_CALL(Dbg_syms_discarded(ofl->ofl_lml, sdp));
2523                                continue;
2524                        }
2525                }

2527                /*
2528                 * If object capabilities for this file are being converted
2529                 * into symbol capabilities, then:
2530                 *
2531                 *  -   Any global function, or initialized global data symbol
2532                 *      definitions (ie., those that are not associated with
2533                 *      special symbol types, ie., ABS, COMMON, etc.), and which
2534                 *      have not been reduced to locals, are converted to symbol
2535                 *      references (UNDEF).  This ensures that any reference to
2536                 *      the original symbol, for example from a relocation, get
2537                 *      associated to a capabilities family lead symbol, ie., a
2538                 *      generic instance.
```

```
2539                     *
2540                     *  -   For each global function, or object symbol definition,
2541                     *      a new local symbol is created.  The function or object
2542                     *      is renamed using the capabilities CA_SUNW_ID definition
2543                     *      (which might have been fabricated for this purpose -
2544                     *      see get_cap_group()).  The new symbol name is:
2545                     *
2546                     *          <original name>%<capability group identifier>
2547                     *
2548                     *      This symbol is associated to the same location, and
2549                     *      becomes a capabilities family member.
2550                     */
2551                    /* LINTED */
2552                    hash = (Word)elf_hash(name);

2554                    ntype = ELF_ST_TYPE(nsym->st_info);
2555                    if (cdp && (nsym->st_shndx != SHN_UNDEF) &&
2556                        ((sdflags & FLG_SY_SPECSEC) == 0) &&
2557                        ((ntype == STT_FUNC) || (ntype == STT_OBJECT))) {
2558                            /*
2559                             * Determine this symbol's visibility.  If a mapfile has
2560                             * indicated this symbol should be local, then there's
2561                             * no point in transforming this global symbol to a
2562                             * capabilities symbol.  Otherwise, create a symbol
2563                             * capability pair descriptor to record this symbol as
2564                             * a candidate for translation.
2565                             */
2566                            if (sym_cap_vis(name, hash, sym, ofl) &&
2567                                ((cpp = alist_append(&cappairs, NULL,
2568                                sizeof (Cap_pair), AL_CNT_CAP_PAIRS)) == NULL))
2569                                    return (S_ERROR);
2570                    }

2572                    if (cpp) {
2573                            Sym     *rsym;

2575                            DBG_CALL(Dbg_syms_cap_convert(ofl, ndx, name, nsym));

2577                            /*
2578                             * Allocate a new symbol descriptor to represent the
2579                             * transformed global symbol.  The descriptor points
2580                             * to the original symbol information (which might
2581                             * indicate a global or weak visibility).  The symbol
2582                             * information will be transformed into a local symbol
2583                             * later, after any weak aliases are culled.
2584                             */
2585                            if ((cpp->c_osdp =
2586                                libld_malloc(sizeof (Sym_desc))) == NULL)
2587                                    return (S_ERROR);

2589                            cpp->c_osdp->sd_name = name;
2590                            cpp->c_osdp->sd_sym = nsym;
2591                            cpp->c_osdp->sd_shndx = shndx;
2592                            cpp->c_osdp->sd_file = ifl;
2593                            cpp->c_osdp->sd_isc = ifl->ifl_isdesc[shndx];
2594                            cpp->c_osdp->sd_ref = REF_REL_NEED;

2596                            /*
2597                             * Save the capabilities group this symbol belongs to,
2598                             * and the original symbol index.
2599                             */
2600                            cpp->c_group = cdp->ca_groups->apl_data[0];
2601                            cpp->c_ndx = ndx;

2603                            /*
2604                             * Replace the original symbol definition with a symbol
```



```
2605                             * reference.  Make sure this reference isn't left as a
2606                             * weak.
2607                             */
2608                            if ((rsym = libld_malloc(sizeof (Sym))) == NULL)
2609                                    return (S_ERROR);

2611                            *rsym = *nsym;

2613                            rsym->st_info = ELF_ST_INFO(STB_GLOBAL, ntype);
2614                            rsym->st_shndx = shndx = SHN_UNDEF;
2615                            rsym->st_value = 0;
2616                            rsym->st_size = 0;

2618                            sdflags |= FLG_SY_CAP;

2620                            nsym = rsym;
2621                    }

2623                    /*
2624                     * If the symbol does not already exist in the internal symbol
2625                     * table add it, otherwise resolve the conflict.  If the symbol
2626                     * from this file is kept, retain its symbol table index for
2627                     * possible use in associating a global alias.
2628                     */
2629                    if ((sdp = ld_sym_find(name, hash, &where, ofl)) == NULL) {
2630                            DBG_CALL(Dbg_syms_global(ofl->ofl_lml, ndx, name));
2631                            if ((sdp = ld_sym_enter(name, nsym, hash, ifl, ofl, ndx,
2632                                shndx, sdflags, &where)) == (Sym_desc *)S_ERROR)
2633                                    return (S_ERROR);

2635                    } else if (ld_sym_resolve(sdp, nsym, ifl, ofl, ndx, shndx,
2636                        sdflags) == S_ERROR)
2637                            return (S_ERROR);

2639                    /*
2640                     * Now that we have a symbol descriptor, retain the descriptor
2641                     * for later use by symbol capabilities processing.
2642                     */
2643                    if (cpp)
2644                            cpp->c_nsdp = sdp;

2646                    /*
2647                     * After we've compared a defined symbol in one shared
2648                     * object, flag the symbol so we don't compare it again.
2649                     */
2650                    if ((etype == ET_DYN) && (nsym->st_shndx != SHN_UNDEF) &&
2651                        ((sdp->sd_flags & FLG_SY_SOFOUND) == 0))
2652                            sdp->sd_flags |= FLG_SY_SOFOUND;

2654                    /*
2655                     * If the symbol is accepted from this file retain the symbol
2656                     * index for possible use in aliasing.
2657                     */
2658                    if (sdp->sd_file == ifl)
2659                            sdp->sd_symndx = ndx;

2661                    ifl->ifl_oldndx[ndx] = sdp;

2663                    /*
2664                     * If we've accepted a register symbol, continue to validate
2665                     * it.
2666                     */
2667                    if (sdp->sd_flags & FLG_SY_REGSYM) {
2668                            Sym_desc        *rsdp;

2670                            /*
```

```
2671                         * The presence of FLG_SY_REGSYM means that
2672                         * the pointers in ld_targ.t_ms are non-NULL.
2673                         */
2674                        rsdp = (*ld_targ.t_ms.ms_reg_find)(sdp->sd_sym, ofl);
2675                        if (rsdp == NULL) {
2676                                if ((*ld_targ.t_ms.ms_reg_enter)(sdp, ofl) == 0)
2677                                        return (S_ERROR);
2678                        } else if (rsdp != sdp) {
2679                                (void) (*ld_targ.t_ms.ms_reg_check)(rsdp,
2680                                    sdp->sd_sym, sdp->sd_name, ifl, ofl);
2681                        }
2682                }

2684                /*
2685                 * For a relocatable object, if this symbol is defined
2686                 * and has non-zero length and references an address
2687                 * within an associated section, then check its extents
2688                 * to make sure the section boundaries encompass it.
2689                 * If they don't, the ELF file is corrupt. Note that this
2690                 * global symbol may have come from another file to satisfy
2691                 * an UNDEF symbol of the same name from this one. In that
2692                 * case, we don't check it, because it was already checked
2693                 * as part of its own file.
2694                 */
2695                if (etype_rel && (sdp->sd_file == ifl)) {
2696                        Sym *tsym = sdp->sd_sym;

2698                        if (SYM_LOC_BADADDR(sdp, tsym,
2699                            ELF_ST_TYPE(tsym->st_info))) {
2700                                issue_badaddr_msg(ifl, ofl, sdp,
2701                                    tsym, tsym->st_shndx);
2702                                continue;
2703                        }
2704                }
2705        }
2706        DBG_CALL(Dbg_util_nl(ofl->ofl_lml, DBG_NL_STD));

2708        /*
2709         * Associate weak (alias) symbols to their non-weak counterparts by
2710         * scanning the global symbols one more time.
2711         *
2712         * This association is needed when processing the symbols from a shared
2713         * object dependency when a a weak definition satisfies a reference:
2714         *
2715         *  -   When building a dynamic executable, if a referenced symbol is a
2716         *      data item, the symbol data is copied to the executables address
2717         *      space.  In this copy-relocation case, we must also reassociate
2718         *      the alias symbol with its new location in the executable.
2719         *
2720         *  -   If the referenced symbol is a function then we may need to
2721         *      promote the symbols binding from undefined weak to undefined,
2722         *      otherwise the run-time linker will not generate the correct
2723         *      relocation error should the symbol not be found.
2724         *
2725         * Weak alias association is also required when a local dynsym table
2726         * is being created.  This table should only contain one instance of a
2727         * symbol that is associated to a given address.
2728         *
2729         * The true association between a weak/strong symbol pair is that both
2730         * symbol entries are identical, thus first we create a sorted symbol
2731         * list keyed off of the symbols section index and value.  If the symbol
2732         * belongs to the same section and has the same value, then the chances
2733         * are that the rest of the symbols data is the same.  This list is then
2734         * scanned for weak symbols, and if one is found then any strong
2735         * association will exist in the entries that follow.  Thus we just have
2736         * to scan one (typically a single alias) or more (in the uncommon
```



```
2737         * instance of multiple weak to strong associations) entries to
2738         * determine if a match exists.
2739         */
2740        if (weak && (OFL_ALLOW_LDYNSYM(ofl) || (etype == ET_DYN)) &&
2741            (total > local)) {
2742                static Sym_desc **sort;
2743                static size_t   osize = 0;
2744                size_t          nsize = (total - local) * sizeof (Sym_desc *);

2746                /*
2747                 * As we might be processing many input files, and many symbols,
2748                 * try and reuse a static sort buffer.  Note, presently we're
2749                 * playing the game of never freeing any buffers as there's a
2750                 * belief this wastes time.
2751                 */
2752                if ((osize == 0) || (nsize > osize)) {
2753                        if ((sort = libld_malloc(nsize)) == NULL)
2754                                return (S_ERROR);
2755                        osize = nsize;
2756                }
2757                (void) memcpy((void *)sort, &ifl->ifl_oldndx[local], nsize);

2759                qsort(sort, (total - local), sizeof (Sym_desc *), compare);

2761                for (ndx = 0; ndx < (total - local); ndx++) {
2762                        Sym_desc        *wsdp = sort[ndx];
2763                        Sym             *wsym;
2764                        int             sndx;

2766                        /*
2767                         * Ignore any empty symbol descriptor, or the case where
2768                         * the symbol has been resolved to a different file.
2769                         */
2770                        if ((wsdp == NULL) || (wsdp->sd_file != ifl))
2771                                continue;

2773                        wsym = wsdp->sd_sym;

2775                        if ((wsym->st_shndx == SHN_UNDEF) ||
2776                            (wsdp->sd_flags & FLG_SY_SPECSEC) ||
2777                            (ELF_ST_BIND(wsym->st_info) != STB_WEAK))
2778                                continue;

2780                        /*
2781                         * We have a weak symbol, if it has a strong alias it
2782                         * will have been sorted to one of the following sort
2783                         * table entries.  Note that we could have multiple weak
2784                         * symbols aliased to one strong (if this occurs then
2785                         * the strong symbol only maintains one alias back to
2786                         * the last weak).
2787                         */
2788                        for (sndx = ndx + 1; sndx < (total - local); sndx++) {
2789                                Sym_desc        *ssdp = sort[sndx];
2790                                Sym             *ssym;
2791                                sd_flag_t       w_dynbits, s_dynbits;

2793                                /*
2794                                 * Ignore any empty symbol descriptor, or the
2795                                 * case where the symbol has been resolved to a
2796                                 * different file.
2797                                 */
2798                                if ((ssdp == NULL) || (ssdp->sd_file != ifl))
2799                                        continue;

2801                                ssym = ssdp->sd_sym;
```

```
2803                                if (ssym->st_shndx == SHN_UNDEF)
2804                                        continue;

2806                                if ((ssym->st_shndx != wsym->st_shndx) ||
2807                                    (ssym->st_value != wsym->st_value))
2808                                        break;

2810                                if ((ssym->st_size != wsym->st_size) ||
2811                                    (ssdp->sd_flags & FLG_SY_SPECSEC) ||
2812                                    (ELF_ST_BIND(ssym->st_info) == STB_WEAK))
2813                                        continue;

2815                                /*
2816                                 * If a sharable object, set link fields so
2817                                 * that they reference each other.‘
2818                                 */
2819                                if (etype == ET_DYN) {
2820                                        ssdp->sd_aux->sa_linkndx =
2821                                            (Word)wsdp->sd_symndx;
2822                                        wsdp->sd_aux->sa_linkndx =
2823                                            (Word)ssdp->sd_symndx;
2824                                }

2826                                /*
2827                                 * Determine which of these two symbols go into
2828                                 * the sort section.  If a mapfile has made
2829                                 * explicit settings of the FLG_SY_*DYNSORT
2830                                 * flags for both symbols, then we do what they
2831                                 * say.  If one has the DYNSORT flags set, we
2832                                 * set the NODYNSORT bit in the other.  And if
2833                                 * neither has an explicit setting, then we
2834                                 * favor the weak symbol because they usually
2835                                 * lack the leading underscore.
2836                                 */
2837                                w_dynbits = wsdp->sd_flags &
2838                                    (FLG_SY_DYNSORT | FLG_SY_NODYNSORT);
2839                                s_dynbits = ssdp->sd_flags &
2840                                    (FLG_SY_DYNSORT | FLG_SY_NODYNSORT);
2841                                if (!(w_dynbits && s_dynbits)) {
2842                                        if (s_dynbits) {
2843                                                if (s_dynbits == FLG_SY_DYNSORT)
2844                                                        wsdp->sd_flags |=
2845                                                            FLG_SY_NODYNSORT;
2846                                        } else if (w_dynbits !=
2847                                            FLG_SY_NODYNSORT) {
2848                                                ssdp->sd_flags |=
2849                                                    FLG_SY_NODYNSORT;
2850                                        }
2851                                }
2852                                break;
2853                        }
2854                }
2855        }

2857        /*
2858         * Having processed all symbols, under -z symbolcap, reprocess any
2859         * symbols that are being translated from global to locals.  The symbol
2860         * pair that has been collected defines the original symbol (c_osdp),
2861         * which will become a local, and the new symbol (c_nsdp), which will
2862         * become a reference (UNDEF) for the original.
2863         *
2864         * Scan these symbol pairs looking for weak symbols, which have non-weak
2865         * aliases.  There is no need to translate both of these symbols to
2866         * locals, only the global is necessary.
2867         */
2868        if (cappairs) {
```



```
2869                Aliste          idx1;
2870                Cap_pair        *cpp1;

2872                for (ALIST_TRAVERSE(cappairs, idx1, cpp1)) {
2873                        Sym_desc        *sdp1 = cpp1->c_osdp;
2874                        Sym             *sym1 = sdp1->sd_sym;
2875                        uchar_t         bind1 = ELF_ST_BIND(sym1->st_info);
2876                        Aliste          idx2;
2877                        Cap_pair        *cpp2;

2879                        /*
2880                         * If this symbol isn’t weak, it’s capability member is
2881                         * retained for the creation of a local symbol.
2882                         */
2883                        if (bind1 != STB_WEAK)
2884                                continue;

2886                        /*
2887                         * If this is a weak symbol, traverse the capabilities
2888                         * list again to determine if a corresponding non-weak
2889                         * symbol exists.
2890                         */
2891                        for (ALIST_TRAVERSE(cappairs, idx2, cpp2)) {
2892                                Sym_desc        *sdp2 = cpp2->c_osdp;
2893                                Sym             *sym2 = sdp2->sd_sym;
2894                                uchar_t         bind2 =
2895                                    ELF_ST_BIND(sym2->st_info);

2897                                if ((cpp1 == cpp2) ||
2898                                    (cpp1->c_group != cpp2->c_group) ||
2899                                    (sym1->st_value != sym2->st_value) ||
2900                                    (bind2 == STB_WEAK))
2901                                        continue;

2903                                /*
2904                                 * The weak symbol (sym1) has a non-weak (sym2)
2905                                 * counterpart.  There’s no point in translating
2906                                 * both of these equivalent symbols to locals.
2907                                 * Add this symbol capability alias to the
2908                                 * capabilities family information, and remove
2909                                 * the weak symbol.
2910                                 */
2911                                if (ld_cap_add_family(ofl, cpp2->c_nsdp,
2912                                    cpp1->c_nsdp, NULL, NULL) == S_ERROR)
2913                                        return (S_ERROR);

2915                                free((void *)cpp1->c_osdp);
2916                                (void) alist_delete(cappairs, &idx1);
2917                        }
2918                }

2920                DBG_CALL(Dbg_util_nl(ofl->ofl_lml, DBG_NL_STD));

2922                /*
2923                 * The capability pairs information now represents all the
2924                 * global symbols that need transforming to locals.  These
2925                 * local symbols are renamed using their group identifiers.
2926                 */
2927                for (ALIST_TRAVERSE(cappairs, idx1, cpp1)) {
2928                        Sym_desc        *osdp = cpp1->c_osdp;
2929                        Objcapset       *capset;
2930                        size_t          nsize, tsize;
2931                        const char      *oname;
2932                        char            *cname, *idstr;
2933                        Sym             *csym;
```

```
2935                        /*
2936                         * If the local symbol has not yet been translated
2937                         * convert it to a local symbol with a name.
2938                         */
2939                        if ((osdp->sd_flags & FLG_SY_CAP) != 0)
2940                                continue;

2942                        /*
2943                         * As we're converting object capabilities to symbol
2944                         * capabilities, obtain the capabilities set for this
2945                         * object, so as to retrieve the CA_SUNW_ID value.
2946                         */
2947                        capset = &cpp1->c_group->cg_set;

2949                        /*
2950                         * Create a new name from the existing symbol and the
2951                         * capabilities group identifier.  Note, the delimiter
2952                         * between the symbol name and identifier name is hard-
2953                         * coded here (%), so that we establish a convention
2954                         * for transformed symbol names.
2955                         */
2956                        oname = osdp->sd_name;

2958                        idstr = capset->oc_id.cs_str;
2959                        nsize = strlen(oname);
2960                        tsize = nsize + 1 + strlen(idstr) + 1;
2961                        if ((cname = libld_malloc(tsize)) == 0)
2962                                return (S_ERROR);

2964                        (void) strcpy(cname, oname);
2965                        cname[nsize++] = '%';
2966                        (void) strcpy(&cname[nsize], idstr);

2968                        /*
2969                         * Allocate a new symbol table entry, transform this
2970                         * symbol to a local, and assign the new name.
2971                         */
2972                        if ((csym = libld_malloc(sizeof (Sym))) == NULL)
2973                                return (S_ERROR);

2975                        *csym = *osdp->sd_sym;
2976                        csym->st_info = ELF_ST_INFO(STB_LOCAL,
2977                            ELF_ST_TYPE(osdp->sd_sym->st_info));

2979                        osdp->sd_name = cname;
2980                        osdp->sd_sym = csym;
2981                        osdp->sd_flags = FLG_SY_CAP;

2983                        /*
2984                         * Keep track of this new local symbol.  As -z symbolcap
2985                         * can only be used to create a relocatable object, a
2986                         * dynamic symbol table can't exist.  Ensure there is
2987                         * space reserved in the string table.
2988                         */
2989                        ofl->ofl_caploclcnt++;
2990                        if (st_insert(ofl->ofl_strtab, cname) == -1)
2991                                return (S_ERROR);

2993                        DBG_CALL(Dbg_syms_cap_local(ofl, cpp1->c_ndx,
2994                            cname, csym, osdp));

2996                        /*
2997                         * Establish this capability pair as a family.
2998                         */
2999                        if (ld_cap_add_family(ofl, cpp1->c_nsdp, osdp,
3000                            cpp1->c_group, &ifl->ifl_caps->ca_syms) == S_ERROR)
```



```
3001                                return (S_ERROR);
3002                        }
3003                }

3005                return (1);

3007 #undef SYM_LOC_BADADDR
3008 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   25939 Fri Aug  9 16:25:50 2013
new/usr/src/cmd/sgs/librtld/common/dldump.c
4003 dldump() can't deal with extended sections
**********************************************************
```

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

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * dldump(3c) creates a new file image from the specified input file.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"

#include        <sys/param.h>
#include        <sys/procfs.h>
#include        <fcntl.h>
#include        <stdio.h>
#include        <libelf.h>
#include        <link.h>
#include        <dlfcn.h>
#include        <stdlib.h>
#include        <string.h>
#include        <unistd.h>
#include        <errno.h>
#include        "libld.h"
#include        "msg.h"
#include        "_librtld.h"

/*
 * Generic clean up routine
 */
static void
cleanup(Elf *ielf, Elf *oelf, Elf *melf, Cache *icache, Cache *mcache,
    int fd, const char *opath)
{
        if (icache) {
                Cache * _icache = icache;

                for (++_icache; _icache->c_flags != FLG_C_END; _icache++) {
                        if (_icache->c_info)
                                (void) free(_icache->c_info);
                }
                (void) free((void *)icache);
        }
        if (mcache)
```



```
                (void) free((void *)mcache);

        if (ielf)
                (void) elf_end(ielf);
        if (oelf)
                (void) elf_end(oelf);
        if (melf)
                (void) elf_end(melf);
        if (fd)
                (void) close(fd);
        if (opath)
                (void) unlink(opath);
}

/*
 * The dldump(3x) interface directs control to the runtime linker.  The runtime
 * linker brings in librtld.so.1 to provide the underlying support for this
 * call (this is because librtld.so.1 requires libelf.so.1, and the whole wad
 * is rather expensive to drag around with ld.so.1).
 *
 * rt_dldump(Rt_map * lmp, const char * opath, int flags, Addr addr)
 *
 * lmp provides the link-map of the ipath (the input file).
 *
 * opath specifies the output file.
 *
 * flags provides a variety of options that control how the new image will be
 * relocated (if required).
 *
 * addr indicates the base address at which the associated input image is mapped
 * within the process.
 *
 * The modes of operation and the various flags provide a number of combinations
 * of images that can be created, some are useful, some maybe not.  The
 * following provide a couple of basic models for dldump(3x) use:
 *
 *  new executable -    dldump(0, outfile, RTLD_MEMORY)
 *
 *                      A dynamic executable may undergo some initialization
 *                      and the results of this saved in a new file for later
 *                      execution.  The executable will presumable update
 *                      parts of its data segment and heap (note that the heap
 *                      should be acquired using malloc() so that it follows
 *                      the end of the data segment for this technique to be
 *                      useful).  These updated memory elements are saved to the
 *                      new file, including a new .SUNW_heap section if
 *                      required.
 *
 *                      For greatest flexibility, no relocated information
 *                      should be saved (by default any relocated information is
 *                      returned to the value it had in its original file).
 *                      This allows the new image to bind to new dynamic objects
 *                      when executed on the same or newer upgrades of the OS.
 *
 *                      Fixing relocations by applying RTLD_REL_ALL will bind
 *                      the image to the dependencies presently mapped as part
 *                      of the process.  Thus the new executable will only work
 *                      correctly when these same dependencies map to exactly
 *                      to the same locations. (note that RTLD_REL_RELATIVE will
 *                      have no effect as dynamic executables commonly don't
 *                      contain any relative relocations).
 *
 *  new shared object - dldump(infile, outfile, RTLD_REL_RELATIVE)
 *
 *                      A shared object can be fixed to a known address so as
 *                      to reduce its relocation overhead on startup.  Because
```

```
 *                      the new file is fixed to a new base address (which is
 *                      the address at which the object was found mapped to the
 *                      process) it is now a dynamic executable.
 *
 *                      Data changes that have occurred due to the object
 *                      gaining control (at the least this would be .init
 *                      processing) will not be carried over to the new image.
 *
 *                      By only performing relative relocations all global
 *                      relocations are available for unique binding to each
 *                      process - thus interposition etc. is still available.
 *
 *                      Using RTLD_REL_ALL will fix all relocations in the new
 *                      file, which will certainly provide for faster startup
 *                      of the new image, but at the loss of interposition
 *                      flexibility.
 */
int
rt_dldump(Rt_map *lmp, const char *opath, int flags, Addr addr)
{
        Elf *           ielf = 0, *oelf = 0, *melf = 0;
        Ehdr            *iehdr, *oehdr, *mehdr;
        Phdr            *iphdr, *ophdr, *data_phdr = 0;
        Cache           *icache = 0, *_icache, *mcache = 0, *_mcache;
        Cache           *data_cache = 0, *dyn_cache = 0;
        Xword           rel_null_no = 0, rel_data_no = 0, rel_func_no = 0;
        Xword           rel_entsize;
        Rel             *rel_base = 0, *rel_null, *rel_data, *rel_func;
        Elf_Scn         *scn;
        Shdr            *shdr;
        Elf_Data        *data;
        Half            endx = 1;
        int             fd = 0, err, num;
        size_t          shstr_size = 1, shndx;
        size_t          shstr_size = 1;
        Addr            edata;
        char            *shstr, *_shstr, *ipath = NAME(lmp);
        prstatus_t      *status = 0, _status;
        Lm_list         *lml = LIST(lmp);
        Alist           *nodirect = 0;

        if (lmp == lml_main.lm_head) {
                char    proc[16];
                int     pfd;

                /*
                 * Get a /proc descriptor.
                 */
                (void) snprintf(proc, 16, MSG_ORIG(MSG_FMT_PROC),
                    (int)getpid());
                if ((pfd = open(proc, O_RDONLY)) == -1) {
                        err = errno;
                        eprintf(lml, ERR_FATAL, MSG_INTL(MSG_SYS_OPEN), proc,
                            strerror(err));
                        return (1);
                }

                /*
                 * If we've been asked to process the dynamic executable we
                 * might not know its full path (this is prior to realpath()
                 * processing becoming default), and thus use /proc to obtain a
                 * file descriptor of the input file.
                 */
                if ((fd = ioctl(pfd, PIOCOPENM, (void *)0)) == -1) {
                        err = errno;
                        eprintf(lml, ERR_FATAL, MSG_INTL(MSG_SYS_PROC), ipath,
```



```
                            strerror(err));
                        (void) close(pfd);
                        return (1);
                }

                /*
                 * Obtain the process's status structure from which we can
                 * determine the size of the process's heap.  Note, if the
                 * application is using mapmalloc then the heap size is going
                 * to be zero, and if we're dumping a data section that makes
                 * reference to the malloc'ed area we're not going to get a
                 * useful image.
                 */
                if (!(flags & RTLD_NOHEAP)) {
                        if (ioctl(pfd, PIOCSTATUS, (void *)&_status) == -1) {
                                err = errno;
                                eprintf(lml, ERR_FATAL, MSG_INTL(MSG_SYS_PROC),
                                    ipath, strerror(err));
                                (void) close(fd);
                                (void) close(pfd);
                                return (1);
                        }
                        if ((flags & RTLD_MEMORY) && _status.pr_brksize)
                                status = &_status;
                }
                (void) close(pfd);
        } else {
                /*
                 * Open the specified file.
                 */
                if ((fd = open(ipath, O_RDONLY, 0)) == -1) {
                        err = errno;
                        eprintf(lml, ERR_FATAL, MSG_INTL(MSG_SYS_OPEN), ipath,
                            strerror(err));
                        return (1);
                }
        }

        /*
         * Initialize with the ELF library and make sure this is a suitable
         * ELF file we're dealing with.
         */
        (void) elf_version(EV_CURRENT);
        if ((ielf = elf_begin(fd, ELF_C_READ, NULL)) == NULL) {
                eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_BEGIN), ipath);
                cleanup(ielf, oelf, melf, icache, mcache, fd, 0);
                return (1);
        }
        (void) close(fd);

        if ((elf_kind(ielf) != ELF_K_ELF) ||
            ((iehdr = elf_getehdr(ielf)) == NULL) ||
            ((iehdr->e_type != ET_EXEC) && (iehdr->e_type != ET_DYN))) {
                eprintf(lml, ERR_FATAL, MSG_INTL(MSG_IMG_ELF), ipath);
                cleanup(ielf, oelf, melf, icache, mcache, 0, 0);
                return (1);
        }

        /*
         * Make sure we can create the new output file.
         */
        if ((fd = open(opath, (O_RDWR | O_CREAT | O_TRUNC), 0777)) == -1) {
                err = errno;
                eprintf(lml, ERR_FATAL, MSG_INTL(MSG_SYS_OPEN), opath,
                    strerror(err));
                cleanup(ielf, oelf, melf, icache, mcache, 0, 0);
```

```
258                 return (1);
259         }
260         if ((oelf = elf_begin(fd, ELF_C_WRITE, NULL)) == NULL) {
261                 eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_BEGIN), opath);
262                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
263                 return (1);
264         }

266         /*
267          * Obtain the input program headers.  Remember the last data segments
268          * program header entry as this will be updated later to reflect any new
269          * heap section size.
270          */
271         if ((iphdr = elf_getphdr(ielf)) == NULL) {
272                 eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETPHDR), ipath);
273                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
274                 return (1);
275         }

277         for (num = 0, ophdr = iphdr; num != iehdr->e_phnum; num++, ophdr++) {
278                 /*
279                  * Save the program header that contains the NOBITS section, or
280                  * the last loadable program header if no NOBITS exists.  A
281                  * NOBITS section translates to a memory size requirement that
282                  * is greater than the file data it is mapped from.  Note that
283                  * we inspect all headers just incase there only exist text
284                  * segments.
285                  */
286                 if (ophdr->p_type == PT_LOAD) {
287                         if (ophdr->p_filesz != ophdr->p_memsz)
288                                 data_phdr = ophdr;
289                         else if (data_phdr) {
290                                 if (data_phdr->p_vaddr < ophdr->p_vaddr)
291                                         data_phdr = ophdr;
292                         } else
293                                 data_phdr = ophdr;
294                 }
295         }

297         /*
298          * If there is no data segment, and a heap section is required,
299          * warn the user and disable the heap addition (Note that you can't
300          * simply append the heap to the last segment, as it might be a text
301          * segment, and would therefore have the wrong permissions).
302          */
303         if (status && !data_phdr) {
304                 eprintf(lml, ERR_WARNING, MSG_INTL(MSG_IMG_DATASEG), ipath);
305                 status = 0;
306         }

308         /*
309          * Obtain the input files section header string table.
310          */

312         if (elf_getshdrstrndx(ielf, &shndx) == -1) {
313                 eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETSHDRSTRNDX), ipath);
314                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
315                 return (1);
316         }
317         if ((scn = elf_getscn(ielf, shndx)) == NULL) {
312         if ((scn = elf_getscn(ielf, iehdr->e_shstrndx)) == NULL) {
318                 eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETSCN), ipath);
319                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
320                 return (1);
321         }
322         if ((data = elf_getdata(scn, NULL)) == NULL) {
```



```
323                 eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETDATA), ipath);
324                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
325                 return (1);
326         }
327         shstr = (char *)data->d_buf;

329         /*
330          * Construct a cache to maintain the input files section information.
331          * Obtain an extra cache element if a heap addition is required.  Also
332          * add an additional entry (marked FLG_C_END) to make the processing of
333          * this cache easier.
334          */

336         if (elf_getshdrnum(ielf, &shndx) == -1) {
337                 eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETSHDRNUM), opath);
338                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
339                 return (1);
340         }

342         num = shndx;

330         num = iehdr->e_shnum;
344         if (status)
345                 num++;
346         if ((icache = malloc((num + 1) * sizeof (Cache))) == 0) {
347                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
348                 return (1);
349         }
350         icache[num].c_flags = FLG_C_END;

352         _icache = icache;
353         _icache++;

355         /*
356          * Traverse each section from the input file collecting the appropriate
357          * ELF information.  Indicate how the section will be processed to
358          * generate the output image.
359          */
360         for (scn = 0; scn = elf_nextscn(ielf, scn); _icache++) {

362                 if ((_icache->c_shdr = shdr = elf_getshdr(scn)) == NULL) {
363                         eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETSHDR), ipath);
364                         cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
365                         return (1);
366                 }

368                 if ((_icache->c_data = elf_getdata(scn, NULL)) == NULL) {
369                         eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETDATA), ipath);
370                         cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
371                         return (1);
372                 }
373                 _icache->c_name = shstr + (size_t)(shdr->sh_name);
374                 _icache->c_scn = scn;
375                 _icache->c_flags = 0;
376                 _icache->c_info = 0;

378                 /*
379                  * Process any .SUNW_syminfo section.  Symbols that are tagged
380                  * as NO_DIRECT are collected, as they should not be bound to.
381                  */
382                 if ((flags & ~RTLD_REL_RELATIVE) &&
383                     (shdr->sh_type == SHT_SUNW_syminfo)) {
384                         if (syminfo(_icache, &nodirect)) {
385                                 cleanup(ielf, oelf, melf, icache, mcache,
386                                     fd, opath);
387                                 return (1);
```

```
388                         }
389                 }
391                 /*
392                  * If the section has no address it is not part of the mapped
393                  * image, and is unlikely to require any further processing.
394                  * The section header string table will be rewritten (this isn't
395                  * always necessary, it's only really required when relocation
396                  * sections are renamed or sections are stripped, but we do
397                  * things the same way regardless).
398                  */
399                 if (shdr->sh_addr == 0) {
400                         if ((shdr->sh_type == SHT_STRTAB) &&
401                             ((strcmp(_icache->c_name,
402                             MSG_ORIG(MSG_SCN_SHSTR))) == 0))
403                                 _icache->c_flags = FLG_C_SHSTR;
404                         else if (flags & RTLD_STRIP) {
405                                 _icache->c_flags = FLG_C_EXCLUDE;
406                                 continue;
407                         }
408                 }
410                 /*
411                  * Skip relocation sections for the time being, they'll be
412                  * analyzed after all sections have been processed.
413                  */
414                 if ((shdr->sh_type == M_REL_SHT_TYPE) && shdr->sh_addr)
415                         continue;
417                 /*
418                  * Sections at this point will simply be passed through to the
419                  * output file.  Keep track of the section header string table
420                  * size.
421                  */
422                 shstr_size += strlen(_icache->c_name) + 1;
424                 /*
425                  * If a heap section is to be added to the output image,
426                  * indicate that it will be added following the last data
427                  * section.
428                  */
429                 if (shdr->sh_addr && ((shdr->sh_addr + shdr->sh_size) ==
430                     (data_phdr->p_vaddr + data_phdr->p_memsz))) {
431                         data_cache = _icache;
433                         if (status) {
434                                 _icache++;
435                                 _icache->c_name =
436                                     (char *)MSG_ORIG(MSG_SCN_HEAP);
437                                 _icache->c_flags = FLG_C_HEAP;
439                                 _icache->c_scn = 0;
440                                 _icache->c_shdr = 0;
441                                 _icache->c_data = 0;
442                                 _icache->c_info = 0;
444                                 shstr_size += strlen(_icache->c_name) + 1;
445                         }
446                 }
447         }
449         /*
450          * Now that we've processed all input sections count the relocation
451          * entries (relocation sections need to reference their symbol tables).
452          */
453         _icache = icache;
```



```
454         for (_icache++; _icache->c_flags != FLG_C_END; _icache++) {
456                 if ((shdr = _icache->c_shdr) == 0)
457                         continue;
459                 /*
460                  * If any form of relocations are to be applied to the output
461                  * image determine what relocation counts exist.  These will be
462                  * used to reorganize (localize) the relocation records.
463                  */
464                 if ((shdr->sh_type == M_REL_SHT_TYPE) && shdr->sh_addr) {
465                         rel_entsize = shdr->sh_entsize;
467                         if (count_reloc(icache, _icache, lmp, flags, addr,
468                             &rel_null_no, &rel_data_no, &rel_func_no,
469                             nodirect)) {
470                                 cleanup(ielf, oelf, melf, icache, mcache,
471                                     fd, opath);
472                                 return (1);
473                         }
474                 }
475         }
477         /*
478          * If any form of relocations are to be applied to the output image
479          * then we will reorganize (localize) the relocation records.  If this
480          * reorganization occurs, the relocation sections will no longer have a
481          * one-to-one relationship with the section they relocate, hence we
482          * rename them to a more generic name.
483          */
484         _icache = icache;
485         for (_icache++; _icache->c_flags != FLG_C_END; _icache++) {
487                 if ((shdr = _icache->c_shdr) == 0)
488                         continue;
490                 if ((shdr->sh_type == M_REL_SHT_TYPE) && shdr->sh_addr) {
491                         if (rel_null_no) {
492                                 _icache->c_flags = FLG_C_RELOC;
493                                 _icache->c_name =
494                                     (char *)MSG_ORIG(MSG_SCN_RELOC);
495                         }
496                         shstr_size += strlen(_icache->c_name) + 1;
497                 }
498         }
501         /*
502          * If there is no data section, and a heap is required, warn the user
503          * and disable the heap addition.
504          */
505         if (!data_cache) {
506                 eprintf(lml, ERR_WARNING, MSG_INTL(MSG_IMG_DATASEC), ipath);
507                 status = 0;
508                 endx = 0;
509         }
511         /*
512          * Determine the value of _edata (which will also be _end) and its
513          * section index for updating the data segments phdr and symbol table
514          * information later.  If a new heap section is being added, update
515          * the values appropriately.
516          */
517         edata = data_phdr->p_vaddr + data_phdr->p_memsz;
518         if (status)
519                 edata += status->pr_brksize;
```

```
521         if (endx) {
522                 /* LINTED */
523                 endx = (Half)elf_ndxscn(data_cache->c_scn);
524                 if (status)
525                         endx++;
526         }
527
528         /*
529          * We're now ready to construct the new elf image.
530          *
531          * Obtain a new elf header and initialize it with any basic information
532          * that isn't calculated as part of elf_update().
533          */
534         if ((oehdr = elf_newehdr(oelf)) == NULL) {
535                 eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_NEWEHDR), opath);
536                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
537                 return (1);
538         }
539         oehdr->e_machine = iehdr->e_machine;
540         oehdr->e_flags = iehdr->e_flags;
541         oehdr->e_type = ET_EXEC;
542         oehdr->e_entry = iehdr->e_entry;
543         if (addr)
544                 oehdr->e_entry += addr;
545
546         /*
547          * Obtain a new set of program headers.  Initialize these with the same
548          * information as the input program headers.  Update the virtual address
549          * and the data segments size to reflect any new heap section.
550          */
551         if ((ophdr = elf_newphdr(oelf, iehdr->e_phnum)) == NULL) {
552                 eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_NEWPHDR), opath);
553                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
554                 return (1);
555         }
556         for (num = 0; num != iehdr->e_phnum; num++, iphdr++, ophdr++) {
557                 *ophdr = *iphdr;
558                 if ((ophdr->p_type != PT_INTERP) && (ophdr->p_type != PT_NOTE))
559                         ophdr->p_vaddr += addr;
560                 if (data_phdr == iphdr) {
561                         if (status)
562                                 ophdr->p_memsz = edata - ophdr->p_vaddr;
563                         ophdr->p_filesz = ophdr->p_memsz;
564                 }
565         }
566
567         /*
568          * Establish a buffer for the new section header string table.  This
569          * will be filled in as each new section is created.
570          */
571         if ((shstr = malloc(shstr_size)) == 0) {
572                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
573                 return (1);
574         }
575         _shstr = shstr;
576         *_shstr++ = '\0';
577
578         /*
579          * Use the input files cache information to generate new sections.
580          */
581         _icache = icache;
582         for (_icache++; _icache->c_flags != FLG_C_END; _icache++) {
583                 /*
584                  * Skip any excluded sections.
585                  */
```



```
586                 if (_icache->c_flags == FLG_C_EXCLUDE)
587                         continue;
588
589                 /*
590                  * Create a matching section header in the output file.
591                  */
592                 if ((scn = elf_newscn(oelf)) == NULL) {
593                         eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_NEWSCN), opath);
594                         cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
595                         return (1);
596                 }
597                 if ((shdr = elf_getshdr(scn)) == NULL) {
598                         eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_NEWSHDR), opath);
599                         cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
600                         return (1);
601                 }
602
603                 /*
604                  * If this is the heap section initialize the appropriate
605                  * entries, otherwise simply use the original section header
606                  * information.
607                  */
608                 if (_icache->c_flags == FLG_C_HEAP) {
609                         shdr->sh_type = SHT_PROGBITS;
610                         shdr->sh_flags = SHF_ALLOC | SHF_WRITE;
611                 } else
612                         *shdr = *_icache->c_shdr;
613
614                 /*
615                  * Create a matching data buffer for this section.
616                  */
617                 if ((data = elf_newdata(scn)) == NULL) {
618                         eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_NEWDATA), opath);
619                         cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
620                         return (1);
621                 }
622
623                 /*
624                  * Determine what data will be used for this section.
625                  */
626                 if (_icache->c_flags == FLG_C_SHSTR) {
627                         /*
628                          * Reassign the shstrtab to the new data buffer we're
629                          * creating.  Insure that the new elf header references
630                          * this section header table.
631                          */
632                         *data = *_icache->c_data;
633
634                         data->d_buf = (void *)shstr;
635                         data->d_size = shstr_size;
636
637                         _icache->c_info = shstr;
638
639                         /* LINTED */
640                         if (elf_ndxscn(scn) >= SHN_LORESERVE) {
641                                 Elf_Scn *_scn;
642                                 Shdr    *shdr0;
643
644                                 /*
645                                  * libelf deals with e_shnum for us, but we
646                                  * need to deal with e_shstrndx ourselves.
647                                  */
648                                 oehdr->e_shstrndx = SHN_XINDEX;
649                                 if ((_scn = elf_getscn(oelf, 0)) == NULL) {
650                                         eprintf(lml, ERR_ELF,
651                                             MSG_ORIG(MSG_ELF_GETSCN), opath);
```

```
652                                     cleanup(ielf, oelf, melf, icache,
653                                         mcache, fd, opath);
654                                     return (1);
655                             }
656                             shdr0 = elf_getshdr(_scn);
657                             shdr0->sh_link = elf_ndxscn(scn);
658                     } else {
659 #endif /* ! codereview */
660                             oehdr->e_shstrndx = (Half)elf_ndxscn(scn);
661                     }
662 #endif /* ! codereview */

664                 } else if (_icache->c_flags == FLG_C_HEAP) {
665                         /*
666                          * Assign the heap to the appropriate memory offset.
667                          */
668                         data->d_buf = status->pr_brkbase;
669                         data->d_type = ELF_T_BYTE;
670                         data->d_size = (size_t)status->pr_brksize;
671                         data->d_off = 0;
672                         data->d_align = 1;
673                         data->d_version = EV_CURRENT;

675                         shdr->sh_addr = data_cache->c_shdr->sh_addr +
676                             data_cache->c_shdr->sh_size;

678                 } else if (_icache->c_flags == FLG_C_RELOC) {
679                         /*
680                          * If some relocations are to be saved in the new image
681                          * then the relocation sections will be reorganized to
682                          * localize their contents.  These relocation sections
683                          * will no longer have a one-to-one relationship with
684                          * the section they relocate, hence we rename them and
685                          * remove their sh_info info.
686                          */
687                         *data = *_icache->c_data;

689                         shdr->sh_info = 0;

691                 } else {
692                         /*
693                          * By default simply pass the section through.  If
694                          * we've been asked to use the memory image of the
695                          * input file reestablish the data buffer address.
696                          */
697                         *data = *_icache->c_data;

699                         if ((shdr->sh_addr) && (flags & RTLD_MEMORY))
700                                 data->d_buf = (void *)(shdr->sh_addr + addr);

702                         /*
703                          * Update any NOBITS section to indicate that it now
704                          * contains data.  If this image is being created
705                          * directly from the input file, zero out the .bss
706                          * section (this saves ld.so.1 having to zero out memory
707                          * or do any /dev/zero mappings).
708                          */
709                         if (shdr->sh_type == SHT_NOBITS) {
710                                 shdr->sh_type = SHT_PROGBITS;
711                                 if (!(flags & RTLD_MEMORY)) {
712                                         if ((data->d_buf = calloc(1,
713                                             data->d_size)) == 0) {
714                                                 cleanup(ielf, oelf, melf,
715                                                     icache, mcache, fd, opath);
716                                                 return (1);
717                                         }
```



```
718                                     }
719                             }
720                     }

722                     /*
723                      * Update the section header string table.
724                      */
725                     /* LINTED */
726                     shdr->sh_name = (Word)(_shstr - shstr);
727                     (void) strcpy(_shstr, _icache->c_name);
728                     _shstr = _shstr + strlen(_icache->c_name) + 1;

730                     /*
731                      * For each section that has a virtual address update its
732                      * address to the fixed location of the new image.
733                      */
734                     if (shdr->sh_addr)
735                             shdr->sh_addr += addr;

737                     /*
738                      * If we've inserted a new section any later sections may need
739                      * their sh_link fields updated (.stabs comes to mind).
740                      */
741                     if (status && endx && (shdr->sh_link >= endx))
742                             shdr->sh_link++;
743             }

745             /*
746              * Generate the new image, and obtain a new elf descriptor that will
747              * allow us to write and update the new image.
748              */
749             if (elf_update(oelf, ELF_C_WRIMAGE) == -1) {
750                     eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_UPDATE), opath);
751                     cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
752                     return (1);
753             }
754             if ((melf = elf_begin(0, ELF_C_IMAGE, oelf)) == NULL) {
755                     eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_BEGIN), opath);
756                     cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
757                     return (1);
758             }
759             if ((mehdr = elf_getehdr(melf)) == NULL) {
760                     eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETEHDR), opath);
761                     cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
762                     return (1);
763             }

765             if (elf_getshdrnum(melf, &shndx) == -1) {
766                     eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETSHDRNUM), opath);
767                     cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
768                     return (1);
769             }

771 #endif /* ! codereview */
772             /*
773              * Construct a cache to maintain the memory files section information.
774              */
775             if ((mcache = malloc(shndx * sizeof (Cache))) == 0) {
627             if ((mcache = malloc(mehdr->e_shnum * sizeof (Cache))) == 0) {
776                     cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
777                     return (1);
778             }
779             _mcache = mcache;
780             _mcache++;

782             for (scn = 0; scn = elf_nextscn(melf, scn); _mcache++) {
```

```
784                 if ((_mcache->c_shdr = elf_getshdr(scn)) == NULL) {
785                         eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETSHDR), opath);
786                         cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
787                         return (1);
788                 }
789
790                 if ((_mcache->c_data = elf_getdata(scn, NULL)) == NULL) {
791                         eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_GETDATA), opath);
792                         cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
793                         return (1);
794                 }
795         }
796
797         /*
798          * Now that we have a complete description of the new image update any
799          * sections that are required.
800          *
801          *  o   reset any symbol table entries.
802          *
803          *  o   reset any relocation entries.
804          *
805          *  o   reset dynamic entries.
806          */
807         _mcache = &mcache[0];
808         for (_icache = &icache[1]; _icache->c_flags != FLG_C_END; _icache++) {
809
810                 if (_icache->c_flags == FLG_C_EXCLUDE)
811                         continue;
812
813                 _mcache++;
814                 shdr = _mcache->c_shdr;
815
816                 /*
817                  * Update the symbol table entries.  _end and _edata will be
818                  * changed to reflect any heap addition.  All global symbols
819                  * will be updated to their new fixed address.
820                  */
821                 if ((shdr->sh_type == SHT_SYMTAB) ||
822                     (shdr->sh_type == SHT_DYNSYM) ||
823                     (shdr->sh_type == SHT_SUNW_LDYNSYM)) {
824                         update_sym(mcache, _mcache, edata, endx, addr);
825                         continue;
826                 }
827
828                 /*
829                  * Update any relocations.  All relocation requirements will
830                  * have been established in count_reloc().
831                  */
832                 if (shdr->sh_type == M_REL_SHT_TYPE) {
833                         if (rel_base == (Rel *)0) {
834                                 rel_base = (Rel *)_mcache->c_data->d_buf;
835                                 rel_null = rel_base;
836                                 rel_data = (Rel *)((Xword)rel_null +
837                                     (rel_null_no * rel_entsize));
838                                 rel_func = (Rel *)((Xword)rel_data +
839                                     (rel_data_no * rel_entsize));
840                         }
841
842                         update_reloc(mcache, icache, _icache, opath, lmp,
843                             &rel_null, &rel_data, &rel_func);
844                         continue;
845                 }
846
847                 /*
848                  * Perform any dynamic entry updates after all relocation
```



```
849                  * processing has been carried out (as its possible the .dynamic
850                  * section could occur before the .rel sections, delay this
851                  * processing until last).
852                  */
853                 if (shdr->sh_type == SHT_DYNAMIC)
854                         dyn_cache = _mcache;
855         }
856
857         if (dyn_cache) {
858                 Xword   off = (Xword)rel_base - (Xword)mehdr;
859
860                 /*
861                  * If we're dumping a fixed object (typically the dynamic
862                  * executable) compensate for its real base address.
863                  */
864                 if (!addr)
865                         off += ADDR(lmp);
866
867                 if (update_dynamic(mcache, dyn_cache, lmp, flags, addr, off,
868                     opath, rel_null_no, rel_data_no, rel_func_no, rel_entsize,
869                     elf_checksum(melf))) {
870                         cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
871                         return (1);
872                 }
873         }
874
875         /*
876          * Having completed all section updates write the memory file out.
877          */
878         if (elf_update(oelf, ELF_C_WRITE) == -1) {
879                 eprintf(lml, ERR_ELF, MSG_ORIG(MSG_ELF_UPDATE), opath);
880                 cleanup(ielf, oelf, melf, icache, mcache, fd, opath);
881                 return (1);
882         }
883
884         cleanup(ielf, oelf, melf, icache, mcache, fd, 0);
885         return (0);
886 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    2333 Fri Aug  9 16:25:50 2013
new/usr/src/cmd/sgs/librtld/common/librtld.msg
4003 dldump() can't deal with extended sections
**********************************************************
```

```
  1 #
  2 # Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  3 # Use is subject to license terms.
  4 #
  5 # CDDL HEADER START
  6 #
  7 # The contents of this file are subject to the terms of the
  8 # Common Development and Distribution License, Version 1.0 only
  9 # (the "License").  You may not use this file except in compliance
 10 # with the License.
 11 #
 12 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 13 # or http://www.opensolaris.org/os/licensing.
 14 # See the License for the specific language governing permissions
 15 # and limitations under the License.
 16 #
 17 # When distributing Covered Code, include this CDDL HEADER in each
 18 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 19 # If applicable, add the following below this CDDL HEADER, with the
 20 # fields enclosed by brackets "[]" replaced with your own identifying
 21 # information: Portions Copyright [yyyy] [name of copyright owner]
 22 #
 23 # CDDL HEADER END
 24 #
 25 # ident "%Z%%M% %I%     %E% SMI"


 28 @ _START_

 30 # Messages for cmd/sgs/librtld

 32 @ MSG_ID_LIBRTLD


 35 # System call messages

 37 @ MSG_SYS_PROC          "%s: /proc error: %s"
 38 @ MSG_SYS_OPEN          "%s: open failed: %s"


 41 # Image processing messages

 43 @ MSG_IMG_ELF           "%s: is not a dynamic ELF object"
 44 @ MSG_IMG_DATASEG       "%s: data segment not found"
 45 @ MSG_IMG_DATASEC       "%s: final data section not found"


 48 # ELF processing messages

 50 @ MSG_DT_UNKNOWN        "%s: unknown dynamic entry: ignored: %lld"


 53 # Basic strings

 55 @ MSG_STR_UNKNOWN       "<unknown>"


 58 @ _END_


 61 # The following strings represent reserved section and symbol names.  Reference
```



```
 62 # to these strings is via the MSG_ORIG() macro, and thus no translations are
 63 # required.

 65 @ MSG_FMT_PROC          "/proc/%d"

 67 @ MSG_ELF_BEGIN         "%s: elf_begin"
 68 @ MSG_ELF_GETDATA       "%s: elf_getdata"
 69 @ MSG_ELF_GETEHDR       "%s: elf_getehdr"
 70 @ MSG_ELF_GETPHDR       "%s: elf_getphdr"
 71 @ MSG_ELF_GETSCN        "%s: elf_getscn"
 72 @ MSG_ELF_GETSHDR       "%s: elf_getshdr"
 73 @ MSG_ELF_GETSHDRNUM    "%s: elf_getshdrnum"
 74 @ MSG_ELF_GETSHDRSTRNDX "%s: elf_getshdrstrndx"
 75 #endif /* ! codereview */
 76 @ MSG_ELF_NEWDATA       "%s: elf_newdata"
 77 @ MSG_ELF_NEWEHDR       "%s: elf_newehdr"
 78 @ MSG_ELF_NEWPHDR       "%s: elf_newphdr"
 79 @ MSG_ELF_NEWSCN        "%s: elf_newscn"
 80 @ MSG_ELF_NEWSHDR       "%s: elf_newshdr"
 81 @ MSG_ELF_UPDATE        "%s: elf_update"

 83 @ MSG_SCN_HEAP          ".SUNW_heap"
 84 @ MSG_SCN_RELOC         ".SUNW_reloc"
 85 @ MSG_SCN_SHSTR         ".shstrtab"

 87 @ MSG_SYM_END           "_end"
 88 @ MSG_SYM_EDATA         "_edata"

 90 @ MSG_SUNW_OST_SGS      "SUNW_OST_SGS"
```

```
**********************************************************
   87934 Fri Aug  9 16:25:51 2013
new/usr/src/cmd/sgs/packages/common/SUNWonld-README
4003 dldump() can't deal with extended sections
3999 libld extended section handling is broken
**********************************************************
```

```
#
# 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 liblddbg 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
--------------------------------------------------------------------------------
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:
521         4872634 Large LD_PRELOAD values can cause SEGV of process
522 are incorporated in the following patches:
```

```
523         Solaris/SunOS 5.6_sparc         patch T107733-11
524         Solaris/SunOS 5.6_x86           patch T107734-11
525 --------------------------------------------------------------------------------

527 ------------------------------------
528 Solaris 9 1202 (2nd Q-update - s9u2)
529 ------------------------------------
530 Bugid   Risk Synopsis
531 ================================================================================
532 4546416 add help messages to ld.so mdbmodule
533 4526752 we should build and ship ld.so's mdb module
534 4624658 update 386 TLS relocation values
535 4622472 LA_SYMB_DLSYM not set for la_symbind() invocations
536 4638070 ldd/ld.so.1 could aid in detecting unreferenced dependencies
537         PSARC/2002/096 Detecting unreferenced dependencies with ldd(1)
538 4633860 Optimization for unused static global variables
539         PSARC/2002/113 ld -zignore - section elimination
540 4642829 ld.so.1 mprotect()'s text segment for weak relocations (it shouldn't)
541 4621479 'make' in $SRC/cmd/sgs/tools tries to install things in the proto area
542 4529912 purge ia64 source from sgs
543 4651709 dlopen(RTLD_NOLOAD) can disable lazy loading
544 4655066 crle: -u with nonexistent config file doesn't work
545 4654406 string tables created by the link-editor could be smaller...
546         PSARC/2002/160 ld -znocompstrtab - disable string-table compression
547 4651493 RTLD_NOW can result in binding to an object prior to its init being run.
548 4662575 linker displacement relocation checking introduces significant
549         linker overhead
550 4533195 ld interposes on malloc()/free() preventing support library from freeing
551         memory
552 4630224 crle get's confused about memory layout of objects...
553 4664855 crle on application failed with ld.so.1 encountering mmap() returning
554         ENOMEM err
555 4669582 latest dynamic linker causes libthread _init to get skipped
556 4671493 ld.so.1 inconsistantly assigns PATHNAME() on primary objects
557 4668517 compile with map.bssalign doesn't copy _iob to bss
558 --------------------------------------------------------------------------------
559 All the above changes are incorporated in the following patches:
560         Solaris/SunOS 5.9_sparc         patch T112963-01
561         Solaris/SunOS 5.8_sparc         patch T109147-17
562         Solaris/SunOS 5.8_x86           patch T109148-17
563 --------------------------------------------------------------------------------
564 4701749 On Solaris 8 + 109147-16 ld crashes when building a dynamic library.
565 4707808 The ldd command is broken in the latest 2.8 linker patch.
566 --------------------------------------------------------------------------------
567 All the above changes are incorporated in the following patches:
568         Solaris/SunOS 5.9_sparc         patch T112963-02
569         Solaris/SunOS 5.8_sparc         patch T109147-18
570         Solaris/SunOS 5.8_x86           patch T109148-18
571 --------------------------------------------------------------------------------
572 4696204 enable extended section indexes in relocatable objects
573         PSARC/2001/332 ELF gABI updates - round II
574         PSARC/2002/369 libelf interfaces to support ELF Extended Sections
575 4706503 linkers need to cope with EF_SPARCV9_PSO/EF_SPARCV9_RMO
576 4716929 updating of local register symbols in dynamic symtab busted...
577 4710814 add "official" support for the "symbolic" keyword in linker map-file
578         PSARC/2002/439 linker mapfile visibility declarations
579 --------------------------------------------------------------------------------
580 All the above changes are incorporated in the following patches:
581         Solaris/SunOS 5.9_sparc         patch T112963-03
582         Solaris/SunOS 5.8_sparc         patch T109147-19
583         Solaris/SunOS 5.8_x86           patch T109148-19
584         Solaris/SunOS 5.7_sparc         patch T106950-19
585         Solaris/SunOS 5.7_x86           patch T106951-19
586 --------------------------------------------------------------------------------

588 ------------------------------------
```



```
589 Solaris 9 403 (3nd Q-update - s9u3)
590 ------------------------------------
591 Bugid   Risk Synopsis
592 ================================================================================
593 4731174 strip(1) does not fixup SHT_GROUP data
594 4733697 -zignore with gcc may exclude C++ exception sections
595 4733317 R_SPARC_*_HIX22 calculations are wrong with 32bit LD building
596         ELF64 binaries
597 4735165 fatal linker error when compiling C++ programs with -xlinkopt
598 4736951 The mcs broken when the target file is an archive file
599 --------------------------------------------------------------------------------
600 All the above changes are incorporated in the following patches:
601         Solaris/SunOS 5.8_sparc         patch T109147-20
602         Solaris/SunOS 5.8_x86           patch T109148-20
603         Solaris/SunOS 5.7_sparc         patch T106950-20
604         Solaris/SunOS 5.7_x86           patch T106951-20
605 --------------------------------------------------------------------------------
606 4739660 Threads deadlock in schedlock and dynamic linker lock.
607 4653148 ld.so.1/libc should unregister its dlclose() exit handler via a fini.
608 4743413 ld.so.1 doesn't terminate argv with NULL pointer when invoked directly
609 4746231 linker core-dumps when SECTION relocations are made against discarded
610         sections
611 4730433 ld.so.1 wastes time repeatedly opening dependencies
612 4744337 missing RD_CONSISTENT event with dlmopen(LD_ID_NEWLM, ...)
613 4670835 rd_load_objiter can ignore callback's return value
614 4745932 strip utility doesn't strip out Dwarf2 debug section
615 4754751 "strip" command doesn't remove comdat stab sections.
616 4755674 Patch 109147-18 results in coredump.
617 --------------------------------------------------------------------------------
618 All the above changes are incorporated in the following patches:
619         Solaris/SunOS 5.9_sparc         patch T112963-04
620         Solaris/SunOS 5.7_sparc         patch T106950-21
621         Solaris/SunOS 5.7_x86           patch T106951-21
622 --------------------------------------------------------------------------------
623 4772927 strip core dumps on an archive library
624 4774727 direct-bindings can fail against copy-reloc symbols
625 --------------------------------------------------------------------------------
626 All the above changes are incorporated in the following patches:
627         Solaris/SunOS 5.9_sparc         patch T112963-05
628         Solaris/SunOS 5.9_x86           patch T113986-01
629         Solaris/SunOS 5.8_sparc         patch T109147-21
630         Solaris/SunOS 5.8_x86           patch T109148-21
631         Solaris/SunOS 5.7_sparc         patch T106950-22
632         Solaris/SunOS 5.7_x86           patch T106951-22
633 --------------------------------------------------------------------------------

635 ------------------------------------
636 Solaris 9 803 (4th Q-update - s9u4)
637 ------------------------------------
638 Bugid   Risk Synopsis
639 ================================================================================
640 4730110 ld.so.1 list implementation could scale better
641 4728822 restrict the objects dlsym() searches.
642         PSARC/2002/478 New dlopen(3dl) flag - RTLD_FIRST
643 4714146 crle: 64-bit secure pathname is incorrect.
644 4504895 dlclose() does not remove all objects
645 4698800 Wrong comments in /usr/lib/ld/sparcv9/map.*
646 4745129 dldump is inconsistent with .dynamic processing errors.
647 4753066 LD_SIGNAL isn't very useful in a threaded environment
648         PSARC/2002/569 New dlinfo(3dl) flag - RTLD_DI_SIGNAL
649 4765536 crle: symbolic links can confuse alternative object configuration info
650 4766815 ld -r of object the TLS data fails
651 4770484 elfdump can not handle stripped archive file
652 4770494 The ld command gives improper error message handling broken archive
653 4775738 overwriting output relocation table when 'ld -zignore' is used
654 4778247 elfdump -e of core files fails
```

```
4779976 elfdump dies on bad relocation entries
4787579 invalid SHT_GROUP entries can cause linker to seg fault
4783869 dlclose: filter closure exhibits hang/failure - introduced with 4504895
4778418 ld.so.1: there be nits out there
4792461 Thread-Local Storage - x86 instruction sequence updates
        PSARC/2002/746 Thread-Local Storage - x86 instruction sequence updates
4461340 sgs: ugly build output while suppressing ia64 (64-bit) build on Intel
4790194 dlopen(..., RTLD_GROUP) has an odd interaction with interposition
4804328 auditing of threaded applications results in deadlock
4806476 building relocatable objects with SHF_EXCLUDE loses relocation
        information
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-06
        Solaris/SunOS 5.9_x86           patch T113986-02
        Solaris/SunOS 5.8_sparc         patch T109147-22
        Solaris/SunOS 5.8_x86           patch T109148-22
--------------------------------------------------------------------------------
4731183 compiler creates .tlsbss section instead of .tbss as documented
4816378 TLS: a tls test case dumps core with C and C++ compilers
4817314 TLS_GD relocations against local symbols do not reference symbol...
4811951 non-default symbol visibility overriden by definition in shared object
4802194 relocation error of mozilla built by K2 compiler
4715815 ld should allow linking with no output file (or /dev/null)
4793721 Need a way to null all code in ISV objects enabling ld performance
        tuning
--------------------------------------------------------------------------------
All the above changes plus:
        4796237 RFE: link-editor became extremely slow with patch 109147-20 and
                static libraries
are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-07
        Solaris/SunOS 5.9_x86           patch T113986-03
        Solaris/SunOS 5.8_sparc         patch T109147-23
        Solaris/SunOS 5.8_x86           patch T109148-23
--------------------------------------------------------------------------------

-----------------------------------
Solaris 9 1203 (5th Q-update - s9u5)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4830584 mmap for the padding region doesn't get freed after dlclose
4831650 ld.so.1 can walk off the end of it's call_init() array...
4831544 ldd using .so modules compiled with FD7 compiler caused a core dump
4834784 Accessing members in a TLS structure causes a core dump in Oracle
4824026 segv when -z combreloc is used with -xlinkopt
4825296 typo in elfdump
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-08
        Solaris/SunOS 5.9_x86           patch T113986-04
        Solaris/SunOS 5.8_sparc         patch T109147-24
        Solaris/SunOS 5.8_x86           patch T109148-24
--------------------------------------------------------------------------------
4470917 Solaris Process Model Unification (link-editor components only)
        PSARC/2002/117 Solaris Process Model Unification
4744411 Bloomberg wants a faster linker.
4811969 64-bit links can be much slower than 32-bit.
4825065 ld(1) should ignore consecutive empty sections.
4838226 unrelocated shared objects may be erroneously collected for init firing
4830889 TLS: testcase coredumps with -xarch=v9 and -g
4845764 filter removal can leave dangling filtee pointer
4811093 apptrace -F libc date core dumps
4826315 Link editors need to be pre- and post- Unified Process Model aware
4868300 interposing on direct bindings can fail
```



```
4872634 Large LD_PRELOAD values can cause SEGV of process
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-09
        Solaris/SunOS 5.9_x86           patch T113986-05
        Solaris/SunOS 5.8_sparc         patch T109147-25
        Solaris/SunOS 5.8_x86           patch T109148-25
--------------------------------------------------------------------------------

-----------------------------------
Solaris 9 404 (6th Q-update - s9u6)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4870260 The elfdump command should produce more warning message on invalid move
        entries.
4865418 empty PT_TLS program headers cause problems in TLS enabled applications
4825151 compiler core dumped with a -mt -xF=%all test
4845829 The runtime linker fails to dlopen() long path name.
4900684 shared libraries with more then 32768 plt's fail for sparc ELF64
4906062 Makefiles under usr/src/cmd/sgs needs to be updated
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-10
        Solaris/SunOS 5.9_x86           patch T113986-06
        Solaris/SunOS 5.8_sparc         patch T109147-26
        Solaris/SunOS 5.8_x86           patch T109148-26
        Solaris/SunOS 5.7_sparc         patch T106950-24
        Solaris/SunOS 5.7_x86           patch T106951-24
--------------------------------------------------------------------------------
4900320 rtld library mapping could be faster
4911775 implement GOTDATA proposal in ld
        PSARC/2003/477 SPARC GOTDATA instruction sequences
4904565 Functionality to ignore relocations against external symbols
4764817 add section types SHT_DEBUG and SHT_DEBUGSTR
        PSARC/2003/510 New ELF DEBUG and ANNOTATE sections
4850703 enable per-symbol direct bindings
4716275 Help required in the link analysis of runtime interfaces
        PSARC/2003/519 Link-editors: Direct Binding Updates
4904573 elfdump may hang when processing archive files
4918310 direct binding from an executable can't be interposed on
4918938 ld.so.1 has become SPARC32PLUS - breaks 4.x binary compatibility
4911796 S1S8 C++: ld dump core when compiled and linked with xlinkopt=1.
4889914 ld crashes with SEGV using -M mapfile under certain conditions
4911936 exception are not catch from shared library with -zignore
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-11
        Solaris/SunOS 5.9_x86           patch T113986-07
        Solaris/SunOS 5.8_sparc         patch T109147-27
        Solaris/SunOS 5.8_x86           patch T109148-27
        Solaris/SunOS 5.7_sparc         patch T106950-25
        Solaris/SunOS 5.7_x86           patch T106951-25
--------------------------------------------------------------------------------
4946992 ld crashes due to huge number of sections (>65,000)
4951840 mcs -c goes into a loop on executable program
4939869 Need additional relocation types for abs34 code model
        PSARC/2003/684 abs34 ELF relocations
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-12
        Solaris/SunOS 5.9_x86           patch T113986-08
        Solaris/SunOS 5.8_sparc         patch T109147-28
        Solaris/SunOS 5.8_x86           patch T109148-28
--------------------------------------------------------------------------------
```

```
 787 ------------------------------------
 788 Solaris 9 904 (7th Q-update - s9u7)
 789 ------------------------------------
 790 Bugid   Risk Synopsis
 791 ============================================================================
 792 4912214 Having multiple of libc.so.1 in a link map causes malloc() to fail
 793 4526878 ld.so.1 should pass MAP_ALIGN flag to give kernel more flexibility
 794 4930997 sgs bld_vernote.ksh script needs to be hardend...
 795 4796286 ld.so.1: scenario for trouble?
 796 4930985 clean up cruft under usr/src/cmd/sgs/tools
 797 4933300 remove references to Ultra-1 in librtld_db demo
 798 4936305 string table compression is much too slow...
 799 4939626 SUNWonld internal package must be updated...
 800 4939565 per-symbol filtering required
 801 4948119 ld(1) -z loadfltr fails with per-symbol filtering
 802 4948427 ld.so.1 gives fatal error when multiple RTLDINFO objects are loaded
 803 4940894 ld core dumps using "-xldscope=symbolic
 804 4955373 per-symbol filtering refinements
 805 4878827 crle(1M) - display post-UPM search paths, and compensate for pre-UPM.
 806 4955802 /usr/ccs/bin/ld dumps core in process_reld()
 807 4964415 elfdump issues wrong relocation error message
 808 4966465 LD_NOAUXFLTR fails when object is both a standard and auxiliary filter
 809 4973865 the link-editor does not scale properly when linking objects with
 810         lots of syms
 811 4975598 SHT_SUNW_ANNOTATE section relocation not resolved
 812 4974828 nss_files nss_compat r_mt tests randomly segfaulting
 813 ----------------------------------------------------------------------------
 814 All the above changes are incorporated in the following patches:
 815         Solaris/SunOS 5.9_sparc         patch T112963-13
 816         Solaris/SunOS 5.9_x86           patch T113986-09
 817 ----------------------------------------------------------------------------
 818 4860508 link-editors should create/promote/verify hardware capabilities
 819 5002160 crle: reservation for dumped objects gets confused by mmaped object
 820 4967869 linking stripped library causes segv in linker
 821 5006657 link-editor doesn't always handle nodirect binding syminfo information
 822 4915901 no way to see ELF information
 823 5021773 ld.so.1 has trouble with objects having more than 2 segments.
 824 ----------------------------------------------------------------------------
 825 All the above changes are incorporated in the following patches:
 826         Solaris/SunOS 5.9_sparc         patch T112963-14
 827         Solaris/SunOS 5.9_x86           patch T113986-10
 828         Solaris/SunOS 5.8_sparc         patch T109147-29
 829         Solaris/SunOS 5.8_x86           patch T109148-29
 830 ----------------------------------------------------------------------------
 831 All the above changes plus:
 832         6850124 dlopen reports "No such file or directory" in spite of ENOMEM
 833                 when mmap fails in anon_map()
 834 are incorporated in the following patches:
 835         Solaris/SunOS 5.9_sparc         patch TXXXXXX-XX
 836         Solaris/SunOS 5.9_x86           patch TXXXXXX-XX
 837 ----------------------------------------------------------------------------

 839 ----------
 840 Solaris 10
 841 ----------
 842 Bugid   Risk Synopsis
 843 ============================================================================
 844 5044797 ld.so.1: secure directory testing is being skipped during filtee
 845         processing
 846 4963676 Remove remaining static libraries
 847 5021541 unnecessary PT_SUNWBSS segment may be created
 848 5031495 elfdump complains about bad symbol entries in core files
 849 5012172 Need error when creating shared object with .o compiled
 850         -xarch=v9 -xcode=abs44
 851 4994738 rd_plt_resolution() resolves ebx-relative PLT entries incorrectly
 852 5023493 ld -m output with patch 109147-25 missing .o information
```



```
 853 ----------------------------------------------------------------------------
 854 All the above changes are incorporated in the following patches:
 855         Solaris/SunOS 5.9_sparc         patch T112963-15
 856         Solaris/SunOS 5.9_x86           patch T113986-11
 857         Solaris/SunOS 5.8_sparc         patch T109147-30
 858         Solaris/SunOS 5.8_x86           patch T109148-30
 859 ----------------------------------------------------------------------------
 860 5071614 109147-29 & -30 break the build of on28-patch on Solaris 8 2/04
 861 5029830 crle: provide for optional alternative dependencies.
 862 5034652 ld.so.1 should save, and print, more error messages
 863 5036561 ld.so.1 outputs non-fatal fatal message about auxiliary filter libraries
 864 5042713 4866170 broke ld.so's ::setenv
 865 5047082 ld can core dump on bad gcc objects
 866 5047612 ld.so.1: secure pathname verification is flawed with filter use
 867 5047235 elfdump can core dump printing PT_INTERP section
 868 4798376 nits in demo code
 869 5041446 gelf_update_*() functions inconsistently return NULL or 0
 870 5032364 M_ID_TLSBSS and M_ID_UNKNOWN have the same value
 871 4707030 Empty LD_PRELOAD_64 doesn't override LD_PRELOAD
 872 4968618 symbolic linkage causes core dump
 873 5062313 dladdr() can cause deadlock in MT apps.
 874 5056867 $ISALIST/$HWCAP expansion should be more flexible.
 875 4918303 0@0.so.1 should not use compiler-supplied crt*.o files
 876 5058415 whocalls cannot take more than 10 arguments
 877 5067518 The fix for 4918303 breaks the build if a new work space is used.
 878 ----------------------------------------------------------------------------
 879 All the above changes are incorporated in the following patches:
 880         Solaris/SunOS 5.9_sparc         patch T112963-16
 881         Solaris/SunOS 5.9_x86           patch T113986-12
 882         Solaris/SunOS 5.8_sparc         patch T109147-31
 883         Solaris/SunOS 5.8_x86           patch T109148-31
 884 ----------------------------------------------------------------------------
 885 5013759 *file* should report hardware/software capabilities (link-editor
 886         components only)
 887 5063580 libldstab: file /tmp/posto..: .stab[.index|.sbfocus] found with no
 888         matching stri
 889 5076838 elfdump(1) is built with a CTF section (the wrong one)
 890 5080344 Hardware capabilities are not enforced for a.out
 891 5079061 RTLD_DEFAULT can be expensive
 892         PSARC/2004/747 New dlsym(3c) Handle - RTLD_PROBE
 893 5064973 allow normal relocs against TLS symbols for some sections
 894 5085792 LD_XXXX_64 should override LD_XXXX
 895 5096272 every executable or library has a .SUNW_dof section
 896 5094135 Bloomberg wants a faster ldd.
 897 5086352 libld.so.3 should be built with a .SUNW_ctf ELF section, ready for CR
 898 5098205 elfdump gives wrong section name for the global offset table
 899 5092414 Linker patch 109147-29 makes Broadvison One-To-One server v4.1
 900         installation fail
 901 5080256 dump(1) doesn't list ELF hardware capabilities
 902 5097347 recursive read lock in gelf_getsym()
 903 ----------------------------------------------------------------------------
 904 All the above changes are incorporated in the following patches:
 905         Solaris/SunOS 5.9_sparc         patch T112963-17
 906         Solaris/SunOS 5.9_x86           patch T113986-13
 907         Solaris/SunOS 5.8_sparc         patch T109147-32
 908         Solaris/SunOS 5.8_x86           patch T109148-32
 909 ----------------------------------------------------------------------------
 910 5106206 ld.so.1 fail to run a Solaris9 program that has libc linked with
 911         -z lazyload
 912 5102601 ON should deliver a 64-bit operating system for Opteron systems
 913         (link-editor components only)
 914 6173852 enable link_auditing technology for amd64
 915 6174599 linker does not create .eh_frame_hdr sections for eh_frame sections
 916         with SHF_LINK_ORDER
 917 6175609 amd64 run-time linker has a corrupted note section
 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 ----------------------------------------------------------------------------
1049 6314115 Checkpoint refuses to start, crashes on start, after application of
1050         linker patch 112963-22
```

```
1051 ------------------------------------------------------------------------------
1052 All the above changes are incorporated in the following patches:
1053         Solaris/SunOS 5.9_sparc         patch T112963-24
1054         Solaris/SunOS 5.9_x86           patch T113986-20
1055         Solaris/SunOS 5.8_sparc         patch T109147-39
1056         Solaris/SunOS 5.8_x86           patch T109148-39
1057 ------------------------------------------------------------------------------
1058 6318306 a dlsym() from a filter should be redirected to an associated filtee
1059 6318401 mis-aligned TLS variable
1060 6324019 ld.so.1: malloc alignment is insufficient for new compilers
1061 6324589 psh coredumps on x86 machines on snv_23
1062 6236594 AMD64: Linker needs to handle the new .lbss section (D)
1063         PSARC 2005/514 AMD64 - large section support
1064 6314743 Linker: incorrect resolution for R_AMD64_GOTPC32
1065 6311865 Linker: x86 medium model; invalid ELF program header
1066 ------------------------------------------------------------------------------
1067 All the above changes are incorporated in the following patches:
1068         Solaris/SunOS 5.10_sparc        patch T117461-07
1069         Solaris/SunOS 5.10_x86          patch T118345-12
1070 ------------------------------------------------------------------------------
1071 6309061 link_audit should use __asm__ with gcc
1072 6310736 gcc and sgs/libld don't get along on SPARC
1073 6329796 Memory leak with iconv_open/iconv_close with patch 109147-33
1074 6332983 s9 linker patches 112963-24/113986-20 causing cluster machines not
1075         to boot
1076 ------------------------------------------------------------------------------
1077 All the above changes are incorporated in the following patches:
1078         Solaris/SunOS 5.10_sparc        patch T117461-08
1079         Solaris/SunOS 5.10_x86          patch T121208-02
1080         Solaris/SunOS 5.9_sparc         patch T112963-25
1081         Solaris/SunOS 5.9_x86           patch T113986-21
1082         Solaris/SunOS 5.8_sparc         patch T109147-40
1083         Solaris/SunOS 5.8_x86           patch T109148-40
1084 ------------------------------------------------------------------------------
1085 6445311 The sparc S8/S9/S10 linker patches which include the fix for the
1086         CR6222525 are hit by the CR6439613.
1087 ------------------------------------------------------------------------------
1088 All the above changes are incorporated in the following patches:
1089         Solaris/SunOS 5.9_sparc         patch T112963-26
1090         Solaris/SunOS 5.8_sparc         patch T109147-41
1091 ------------------------------------------------------------------------------

1093 ------------------------------------
1094 Solaris 10 807 (4th Q-update - s10u4)
1095 ------------------------------------
1096 Bugid   Risk Synopsis
1097 ==============================================================================
1098 6487273 ld.so.1 may open arbitrary locale files when relative path is built
1099         from locale environment vars
1100 6487284 ld.so.1: buffer overflow in doprf() function
1101 ------------------------------------------------------------------------------
1102 All the above changes are incorporated in the following patches:
1103         Solaris/SunOS 5.10_sparc        patch T124922-01
1104         Solaris/SunOS 5.10_x86          patch T124923-01
1105         Solaris/SunOS 5.9_sparc         patch T112963-27
1106         Solaris/SunOS 5.9_x86           patch T113986-22
1107         Solaris/SunOS 5.8_sparc         patch T109147-42
1108         Solaris/SunOS 5.8_x86           patch T109148-41
1109 ------------------------------------------------------------------------------
1110 6477132 ld.so.1: memory leak when running set*id application
1111 ------------------------------------------------------------------------------
1112 All the above changes are incorporated in the following patches:
1113         Solaris/SunOS 5.10_sparc        patch T124922-02
1114         Solaris/SunOS 5.10_x86          patch T124923-02
1115         Solaris/SunOS 5.9_sparc         patch T112963-30
1116         Solaris/SunOS 5.9_x86           patch T113986-24
```



```
1117 ------------------------------------------------------------------------------
1118 6340814 ld.so.1 core dump with HWCAP relocatable object + updated statistics
1119 6307274 crle bug with LD_LIBRARY_PATH
1120 6317969 elfheader limited to 65535 segments (link-editor components only)
1121 6350027 ld.so.1 aborts with assertion failed on amd64
1122 6362044 ld(1) inconsistencies with LD_DEBUG=-Dunused and -zignore
1123 6362047 ld.so.1 dumps core when combining HWCAP and LD_PROFILE
1124 6304206 runtime linker may respect LANG and LC_MESSAGE more than LC_ALL
1125 6363495 Catchup required with Intel relocations
1126 6326497 ld.so not properly processing LD_LIBRARY_PATH ending in :
1127 6307146 mcs dumps core when appending null string to comment section
1128 6371877 LD_PROFILE_64 with gprof does not produce correct results on amd64
1129 6372082 ld -r erroneously creates .got section on i386
1130 6201866 amd64: linker symbol elimination is broken
1131 6372620 printstack() segfaults when called from static function (D)
1132 6380470 32-bit ld(1) incorrectly builds 64-bit relocatable objects
1133 6391407 Insufficient alignment of 32-bit object in archive makes ld segfault
1134         (libelf component only) (D)
1135 6316708 LD_DEBUG should provide a means of identifying/isolating individual
1136         link-map lists (P)
1137 6280209 elfdump cores on memory model 0x3
1138 6197234 elfdump and dump don't handle 64-bit symbols correctly
1139 6398893 Extended section processing needs some work
1140 6397256 ldd dumps core in elf_fix_name
1141 6327926 ld does not set etext symbol correctly for AMD64 medium model (D)
1142 6390410 64-bit LD_PROFILE can fail: relocation error when binding profile plt
1143 6382945 AMD64-GCC: dbx: internal error: dwarf reference attribute out of bounds
1144 6262333 init section of .so dlopened from audit interface not being called
1145 6409613 elf_outsync() should fsync()
1146 6426048 C++ exceptions broken in Nevada for amd64
1147 6429418 ld.so.1: need work-around for Nvidia drivers use of static TLS
1148 6429504 crle(1) shows wrong defaults for non-existent 64-bit config file
1149 6431835 data corruption on x64 in 64-bit mode while LD_PROFILE is in effect
1150 6423051 static TLS support within the link-editors needs a major face lift (D)
1151 6388946 attempting to dlopen a .o file mislabeled as .so fails
1152 6446740 allow mapfile symbol definitions to create backing storage (D)
1153 4986360 linker crash on exec of .so (as opposed to a.out) -- error preferred
1154         instead
1155 6229145 ld: initarray/finiarray processing occurs after got size is determined
1156 6324924 the linker should warn if there's a .init section but not _init
1157 6424132 elfdump inserts extra whitespace in bitmap value display
1158 6449485 ld(1) creates misaligned TLS in binary compiled with -xpg
1159 6424550 Write to unallocated (wua) errors when libraries are built with
1160         -z lazyload
1161 6464235 executing the 64-bit ld(1) should be easy (D)
1162 6465623 need a way of building unix without an interpreter
1163 6467925 ld: section deletion (-z ignore) requires improvement
1164 6357230 specfiles should be nuked (link-editor components only)
1165 ------------------------------------------------------------------------------
1166 All the above changes are incorporated in the following patches:
1167         Solaris/SunOS 5.10_sparc        patch T124922-03
1168         Solaris/SunOS 5.10_x86          patch T124923-03

1170 These patches also include the framework changes for the following bug fixes.
1171 However, the associated feature has not been enabled in Solaris 10 or earlier
1172 releases:

1174 6174390 crle configuration files are inconsistent across platforms (D, P)
1175 6432984 ld(1) output file removal - change default behavior (D)
1176         PSARC/2006/353 ld(1) output file removal - change default behavior
1177 ------------------------------------------------------------------------------

1179 ------------------------------------
1180 Solaris 10 508 (5th Q-update - s10u5)
1181 ------------------------------------
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:
```

```
        Solaris/SunOS 5.10_sparc        patch TXXXXXX-XX
        Solaris/SunOS 5.10_x86          patch TXXXXXX-XX
-------------------------------------------------------------------------------

-------------------------------------------
Solaris Nevada (OpenSolaris 2008.05, snv_86)
-------------------------------------------
Bugid   Risk Synopsis
===============================================================================
6409350 BrandZ project integration into Solaris (link-editor components only)
6459189 UNIX03: *VSC* c99 compiler overwrites non-writable file
6423746 add an option to relax the resolution of COMDAT relocs (D)
4934427 runtime linker should load up static symbol names visible to
        dladdr() (D)
        PSARC 2006/526 SHT_SUNW_LDYNSYM - default local symbol addition
6448719 sys/elf.h could be updated with additional machine and ABI types
6336605 link-editors need to support R_*_SIZE relocations
        PSARC/2006/558 R_*_SIZE relocation support
6475375 symbol search optimization to reduce rescans
6475497 elfdump(1) is misreporting sh_link
6482058 lari(1) could be faster, and handle per-symbol filters better
6482974 defining virtual address of text segment can result in an invalid data
        segment
6476734 crle(1m) "-l" as described fails system, crle cores trying to fix
        /a/var/ld/ld.config in failsafe
6487499 link_audit "make clobber" creates and populates proto area
6488141 ld(1) should detect attempt to reference 0-length .bss section
6496718 restricted visibility symbol references should trigger archive
        extraction
6515970 HWCAP processing doesn't clean up fmap structure - browser fails to
        run java applet
6494214 Refinements to symbolic binding, symbol declarations and
        interposition (D)
        PSARC/2006/714 ld(1) mapfile: symbol interpose definition
6475344 DTrace needs ELF function and data symbols sorted by address (D)
        PSARC/2007/026 ELF symbol sort sections
6518480 ld -melf_i386 doesn't complain (D)
6519951 bfu is just another word for exit today (RPATH -> RUNPATH conversion
        bites us) (D)
6521504 ld: hardware capabilities processing from relocatables objects needs
        hardening.
6518322 Some ELF utilities need updating for .SUNW_ldynsym section (D)
        PSARC/2007/074 -L option for nm(1) to display SHT_SUNW_LDYNSYM symbols
6523787 dlopen() handle gets mistakenly orphaned - results in access to freed
        memory
6531189 SEGV in dladdr()
6527318 dlopen(name, RTLD_NOLOAD) returns handle for unloaded library
6518359 extern mapfiles references to _init/_fini can create INIT/FINI
        addresses of 0
6533587 ld.so.1: init/fini processing needs to compensate for interposer
        expectations
6516118 Reserved space needed in ELF dynamic section and string table (D)
        PSARC/2007/127 Reserved space for editing ELF dynamic sections
6535688 elfdump could be more robust in the face of Purify (D)
6516665 The link-editors should be more resilient against gcc's symbol
        versioning
6541004 hwcap filter processing can leak memory
5108874 elfdump SEGVs on bad object file
6547441 Uninitialized variable causes ld.so.1 to crash on object cleanup
6341667 elfdump should check alignments of ELF header elements
6387860 elfdump cores, when processing linux built ELF file
6198202 mcs -d dumps core
6246083 elfdump should allow section index specification
        (numeric -N equivalent) (D)
        PSARC/2007/247 Add -I option to elfdump
6556563 elfdump section overlap checking is too slow for large files
```



```
5006034 need ?E mapfile feature extension (D)
6565476 rtld symbol version check prevents GNU ld binary from running
6567670 ld(1) symbol size/section size verification uncovers Haskell
        compiler inconsistency
6530249 elfdump should handle ELF files with no section header table (D)
        PSARC/2007/395 Add -P option to elfdump
6573641 ld.so.1 does not maintain parent relationship to a dlopen() caller.
6577462 Additional improvements needed to handling of gcc's symbol versioning
6583742 ELF string conversion library needs to lose static writable buffers
6589819 ld generated reference to __tls_get_addr() fails when resolving to a
        shared object reference
6595139 various applications should export yy* global variables for libl
        PSARC/2007/474 new ldd(1) -w option
6597841 gelf_getdyn() reads one too many dynamic entries
6603313 dlclose() can fail to unload objects after fix for 6573641
6234471 need a way to edit ELF objects (D)
        PSARC/2007/509 elfedit
5035454 mixing -Kpic and -KPIC may cause SIGSEGV with -xarch=v9
6473571 strip and mcs get confused and corrupt files when passed
        non-ELF arguments
6253589 mcs has problems handling multiple SHT_NOTE sections
6610591 do_reloc() should not require unused arguments
6602451 new symbol visibilities required: EXPORTED, SINGLETON and ELIMINATE (D)
        PSARC/2007/559 new symbol visibilities - EXPORTED, SINGLETON, and
        ELIMINATE
6570616 elfdump should display incorrectly aligned note section
6614968 elfedit needs string table module (D)
6620533 HWCAP filtering can leave uninitialized data behind - results in
        "rejected: Invalid argument"
6617855 nodirect tag can be ignored when other syminfo tags are available
        (link-editor components only)
6621066 Reduce need for new elfdump options with every section type (D)
        PSARC/2007/620 elfdump -T, and simplified matching
6627765 soffice failure after integration of 6603313 - dangling GROUP pointer.
6319025 SUNWbtool packaging issues in Nevada and S10u1.
6626135 elfedit capabilities str->value mapping should come from
        usr/src/common/elfcap
6642769 ld(1) -z combreloc should become default behavior (D)
        PSARC/2008/006 make ld(1) -z combreloc become default behavior
6634436 XFFLAG should be updated.  (link-editor components only)
6492726 Merge SHF_MERGE|SHF_STRINGS input sections (D)
4947191 OSNet should use direct bindings  (link-editor components only)
6654381 lazy loading fall-back needs optimizing
6658385 ld core dumps when building Xorg on nv_82
6516808 ld.so.1's token expansion provides no escape for platforms that don't
        report HWCAP
6668534 Direct bindings can compromise function address comparisons from
        executables
6667661 Direct bindings can compromise executables with insufficient copy
        relocation information
6357282 ldd should recognize PARENT and EXTERN symbols (D)
        PSARC/2008/148 new ldd(1) -p option
6672394 ldd(1) unused dependency processing is tricked by relocations errors
-------------------------------------------------------------------------------

-------------------------------------------
Solaris Nevada (OpenSolaris 2008.11, snv_101)
-------------------------------------------
Bugid   Risk Synopsis
===============================================================================
6671255 link-editor should support cross linking (D)
        PSARC/2008/179 cross link-editor
6674666 elfedit dyn:posflag1 needs option to locate element via NEEDED item
6675591 elfwrap - wrap data in an ELF file (D,P)
        PSARC/2008/198 elfwrap - wrap data in an ELF file
6678244 elfdump dynamic section sanity checking needs refinement
```

```
6679212 sgs use of SCCS id for versioning is obstacle to mercurial migration
6681761 lies, darn lies, and linker README files
6509323 Need to disable the Multiple Files loading - same name, different
        directories (or its stat() use)
6686889 ld.so.1 regression - bad pointer created with 6509323 integration
6695681 ldd(1) crashes when run from a chrooted environment
6516212 usr/src/cmd/sgs/libelf warlock targets should be fixed or abandoned
6678310 using LD_AUDIT, ld.so.1 calls shared library's .init before library is
        fully relocated (link-editor components only)
6699594 The ld command has a problem handling 'protected' mapfile keyword.
6699131 elfdump should display core file notes (D)
6702260 single threading .init/.fini sections breaks staroffice
6703919 boot hangs intermittently on x86 with onnv daily.0430 and on
6701798 ld can enter infinite loop processing bad mapfile
6706401 direct binding copy relocation fallback is insufficient for ild
        generated objects
6705846 multithreaded C++ application seems to get deadlocked in the dynamic
        linker code
6686343 ldd(1) - unused search path diagnosis should be enabled
6712292 ld.so.1 should fall back to an interposer for failed direct bindings
6716350 usr/src/cmd/sgs should be linted by nightly builds
6720509 usr/src/cmd/sgs/sgsdemangler should be removed
6617475 gas creates erroneous FILE symbols [was: ld.so.1 is reported as
        false positive by wsdiff]
6724311 dldump() mishandles R_AMD64_JUMP_SLOT relocations
6724774 elfdump -n doesn't print siginfo structure
6728555 Fix for amd64 aw (6617475) breaks pure gcc builds
6734598 ld(1) archive processing failure due to mismatched file descriptors (D)
6735939 ld(1) discarded symbol relocations errors (Studio and GNU).
6354160 Solaris linker includes more than one copy of code in binary when
        linking gnu object code
6744003 ld(1) could provide better argument processing diagnostics (D)
        PSARC 2008/583 add gld options to ld(1)
6749055 ld should generate GNU style VERSYM indexes for VERNEED records (D)
        PSARC/2008/603 ELF objects to adopt GNU-style Versym indexes
6752728 link-editor can enter UNDEF symbols in symbol sort sections
6756472 AOUT search path pruning (D)
--------------------------------------------------------------------------------

----------------------------------------------
Solaris Nevada (OpenSolaris 2009.06, snv_111)
----------------------------------------------
Bugid   Risk Synopsis
================================================================================

6754965 introduce the SF1_SUNW_ADDR32 bit in software capabilities (D)
        (link-editor components only)
        PSARC/2008/622 32-bit Address Restriction Software Capabilities Flag
6756953 customer requests that DT_CONFIG strings be honored for secure apps (D)
6765299 ld --version-script option not compatible with GNU ld (D)
6748160 problem with -zrescan (D)
        PSARC/2008/651 New ld archive rescan options
6763342 sloppy relocations need to get sloppier
6736890 PT_SUNWBSS should be disabled (D)
        PSARC/2008/715 PT_SUNWBSS removal
6772661 ldd/lddstub/ld.so.1 dump core in current nightly while processing
        libsoftcrypto_hwcap.so.1
6765931 mcs generates unlink(NULL) system calls
6775062 remove /usr/lib/libldstab.so (D)
6782977 ld segfaults after support lib version error sends bad args to vprintf()
6773695 ld -z nopartial can break non-pic objects
6778453 RTLD_GROUP prevents use of application defined malloc
6789925 64-bit applications with SF1_SUNW_ADDR32 require non-default starting
        address
6792906 ld -z nopartial fix breaks TLS
6686372 ld.so.1 should use mmapobj(2)
```



```
6726108 dlopen() performance could be improved.
6792836 ld is slow when processing GNU linkonce sections
6797468 ld.so.1: orphaned handles aren't processed correctly
6798676 ld.so.1: enters infinite loop with realloc/defragmentation logic
6237063 request extension to dl* family to provide segment bounds
        information (D)
        PSARC/2009/054 dlinfo(3c) - segment mapping retrieval
6800388 shstrtab can be sized incorrectly when -z ignore is used
6805009 ld.so.1: link map control list tear down leaves dangling pointer -
        pfinstall does it again.
6807050 GNU linkonce sections can create duplicate and incompatible
        eh_frame FDE entries
--------------------------------------------------------------------------------

--------------
Solaris Nevada
--------------
Bugid   Risk Synopsis
================================================================================
6813909 generalize eh_frame support to non-amd64 platforms
6801536 ld: mapfile processing oddities unveiled through mmapobj(2) observations
6802452 libelf shouldn't use MS_SYNC
6818012 nm tries to modify readonly segment and dumps core
6821646 xVM dom0 doesn't boot on daily.0324 and beyond
6822828 librtld_db can return RD_ERR before RD_NOMAPS, which compromises dbx
        expectations.
6821619 Solaris linkers need systematic approach to ELF OSABI (D)
        PSARC/2009/196 ELF objects to set OSABI / elfdump -O option
6827468 6801536 breaks 'ld -s' if there are weak/strong symbol pairs
6715578 AOUT (BCP) symbol lookup can be compromised with lazy loading.
6752883 ld.so.1 error message should be buffered (not sent to stderr).
6577982 ld.so.1 calls getpid() before it should when any LD_* are set
6831285 linker LD_DEBUG support needs improvements (D)
6806791 filter builds could be optimized (link-editor components only)
6823371 calloc() uses suboptimal memset() causing 15% regression in SpecCPU2006
        gcc code (link-editor components only)
6831308 ld.so.1: symbol rescanning does a little too much work
6837777 ld ordered section code uses too much memory and works too hard
6841199 Undo 10 year old workaround and use 64-bit ld on 32-bit objects
6784790 ld should examine archives to determine output object class/machine (D)
        PSARC/2009/305 ld -32 option
6849998 remove undocumented mapfile $SPECVERS and $NEED options
6851224 elf_getshnum() and elf_getshstrndx() incompatible with 2002 ELF gABI
        agreement (D)
        PSARC/2009/363 replace elf_getphnum, elf_getshnum, and elf_getshstrndx
6853809 ld.so.1: rescan fallback optimization is invalid
6854158 ld.so.1: interposition can be skipped because of incorrect
        caller/destination validation
6862967 rd_loadobj_iter() failing for core files
6856173 streams core dumps when compiled in 64bit with a very large static
        array size
6834197 ld pukes when given an empty plate
6516644 per-symbol filtering shouldn't be allowed in executables
6878605 ld should accept '%' syntax when matching input SHT_PROGBITS sections
6850768 ld option to autogenerate wrappers/interposers similar to GNU ld
        --wrap (D)
        PSARC/2009/493 ld -z wrap option
6888489 Null environment variables are not overriding crle(1) replaceable
        environment variables.
6885456 Need to implement GNU-ld behavior in construction of .init/.fini
        sections
6900241 ld should track SHT_GROUP sections by symbol name, not section name
6901773 Special handling of STT_SECTION group signature symbol for GNU objects
6901895 Failing asserts in ld update_osym() trying to build gcc 4.5 develpment
        head
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 #endif /* ! codereview */
```

```
**********************************************************
   33170 Fri Aug  9 16:25:51 2013
new/usr/src/cmd/sgs/pvs/common/pvs.c
4000 pvs can't deal with extended sections
**********************************************************
______unchanged_portion_omitted_

1002 int
1003 main(int argc, char **argv, char **envp)
1004 {
1005         GElf_Shdr       shdr;
1006         Elf             *elf;
1007         Elf_Scn         *scn;
1008         Elf_Data        *data;
1009         GElf_Ehdr       ehdr;
1010         int             nfile, var;
1011         char            *names;
1012         Cache           *cache, *_cache;
1013         Cache           *_cache_def, *_cache_need, *_cache_sym, *_cache_loc;
1014         int             error = 0;
1015         Gver_sym_data   vsdata_s;
1016         const Gver_sym_data     *vsdata = NULL;

1018         /*
1019          * Check for a binary that better fits this architecture.
1020          */
1021         (void) conv_check_native(argv, envp);

1023         /*
1024          * Establish locale.
1025          */
1026         (void) setlocale(LC_MESSAGES, MSG_ORIG(MSG_STR_EMPTY));
1027         (void) textdomain(MSG_ORIG(MSG_SUNW_OST_SGS));

1029         cname = argv[0];
1030         Cflag = dflag = lflag = nflag = oflag = rflag = sflag = vflag = 0;

1032         opterr = 0;
1033         while ((var = getopt(argc, argv, MSG_ORIG(MSG_STR_OPTIONS))) != EOF) {
1034                 switch (var) {
1035                 case 'C':
1036                         Cflag = USR_DEFINED;
1037                         break;
1038                 case 'd':
1039                         dflag = USR_DEFINED;
1040                         break;
1041                 case 'l':
1042                         lflag = sflag = USR_DEFINED;
1043                         break;
1044                 case 'n':
1045                         nflag = USR_DEFINED;
1046                         break;
1047                 case 'o':
1048                         oflag = USR_DEFINED;
1049                         break;
1050                 case 'r':
1051                         rflag = USR_DEFINED;
1052                         break;
1053                 case 's':
1054                         sflag = USR_DEFINED;
1055                         break;
1056                 case 'v':
1057                         vflag = USR_DEFINED;
1058                         break;
1059                 case 'I':
1060                 case 'N':
```



```
1061                         add_match_record(var, optarg);
1062                         break;
1063                 case '?':
1064                         (void) fprintf(stderr, MSG_INTL(MSG_USAGE_BRIEF),
1065                             cname);
1066                         (void) fprintf(stderr, MSG_INTL(MSG_USAGE_DETAIL));
1067                         exit(1);
1068                 default:
1069                         break;
1070                 }
1071         }

1073         /*
1074          * No files specified on the command line?
1075          */
1076         if ((nfile = argc - optind) == 0) {
1077                 (void) fprintf(stderr, MSG_INTL(MSG_USAGE_BRIEF), cname);
1078                 exit(1);
1079         }

1081         /*
1082          * By default print both version definitions and needed dependencies.
1083          */
1084         if ((dflag == 0) && (rflag == 0) && (lflag == 0))
1085                 dflag = rflag = DEF_DEFINED;

1087         /*
1088          * Open the input file and initialize the elf interface.
1089          */
1090         for (; optind < argc; optind++) {
1091                 int             derror = 0, nerror = 0, err;
1092                 const char      *file = argv[optind];
1093                 size_t          shnum = 0;
1094 #endif /* ! codereview */

1096                 if ((var = open(file, O_RDONLY)) == -1) {
1097                         err = errno;
1098                         (void) fprintf(stderr, MSG_INTL(MSG_SYS_OPEN),
1099                             cname, file, strerror(err));
1100                         error = 1;
1101                         continue;
1102                 }
1103                 (void) elf_version(EV_CURRENT);
1104                 if ((elf = elf_begin(var, ELF_C_READ, NULL)) == NULL) {
1105                         (void) fprintf(stderr, MSG_ORIG(MSG_ELF_BEGIN), cname,
1106                             file, elf_errmsg(elf_errno()));
1107                         error = 1;
1108                         (void) close(var);
1109                         continue;
1110                 }
1111                 if (elf_kind(elf) != ELF_K_ELF) {
1112                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_NOTELF), cname,
1113                             file);
1114                         error = 1;
1115                         (void) close(var);
1116                         (void) elf_end(elf);
1117                         continue;
1118                 }
1119                 if (gelf_getehdr(elf, &ehdr) == NULL) {
1120                         (void) fprintf(stderr, MSG_ORIG(MSG_ELF_GETEHDR), cname,
1121                             file, elf_errmsg(elf_errno()));
1122                         error = 1;
1123                         (void) close(var);
1124                         (void) elf_end(elf);
1125                         continue;
1126                 }
```

```
1128                /*
1129                 *  Obtain the .shstrtab data buffer to provide the required
1130                 * section name strings.
1131                 */
1132                if ((scn = elf_getscn(elf, ehdr.e_shstrndx)) == NULL) {
1133                        (void) fprintf(stderr, MSG_ORIG(MSG_ELF_GETSCN), cname,
1134                            file, elf_errmsg(elf_errno()));
1135                        error = 1;
1136                        (void) close(var);
1137                        (void) elf_end(elf);
1138                        continue;
1139                }
1140                if ((data = elf_getdata(scn, NULL)) == NULL) {
1141                        (void) fprintf(stderr, MSG_ORIG(MSG_ELF_GETDATA), cname,
1142                            file, elf_errmsg(elf_errno()));
1143                        error = 1;
1144                        (void) close(var);
1145                        (void) elf_end(elf);
1146                        continue;
1147                }
1148                names = data->d_buf;

1150                /*
1151                 * Fill in the cache descriptor with information for each
1152                 * section we might need.   We probably only need to save
1153                 * read-only allocable sections as this is where the version
1154                 * structures and their associated symbols and strings live.
1155                 * However, God knows what someone can do with a mapfile, and
1156                 * as elf_begin has already gone through all the overhead we
1157                 * might as well set up the cache for every section.
1158                 */
1159                if (elf_getshdrnum(elf, &shnum) == -1) {
1160                        (void) fprintf(stderr, MSG_ORIG(MSG_ELF_GETSHDRNUM),
1161                            cname, file, elf_errmsg(elf_errno()));
1162                        exit(1);
1163                }

1165                if ((cache = calloc(shnum, sizeof (Cache))) == NULL) {
1093                *if ((cache = calloc(ehdr.e_shnum, sizeof (Cache))) == NULL) {*
1166                        int err = errno;
1167                        (void) fprintf(stderr, MSG_INTL(MSG_SYS_MALLOC), cname,
1168                            file, strerror(err));
1169                        exit(1);
1170                }

1172                _cache_def = _cache_need = _cache_sym = _cache_loc = NULL;
1173                _cache = cache;
1174                _cache++;
1175                for (scn = NULL; scn = elf_nextscn(elf, scn); _cache++) {
1176                        if (gelf_getshdr(scn, &shdr) == NULL) {
1177                                (void) fprintf(stderr,
1178                                    MSG_ORIG(MSG_ELF_GETSHDR), cname, file,
1179                                    elf_errmsg(elf_errno()));
1180                                error = 1;
1181                                continue;
1182                        }
1183                        if ((_cache->c_data = elf_getdata(scn, NULL)) ==
1184                            NULL) {
1185                                (void) fprintf(stderr,
1186                                    MSG_ORIG(MSG_ELF_GETDATA), cname, file,
1187                                    elf_errmsg(elf_errno()));
1188                                error = 1;
1189                                continue;
1190                        }
1191                        _cache->c_scn = scn;
```



```
1192                        _cache->c_name = names + shdr.sh_name;

1194                        /*
1195                         * Remember the version sections and symbol table.
1196                         */
1197                        switch (shdr.sh_type) {
1198                        case SHT_SUNW_verdef:
1199                                if (dflag)
1200                                        _cache_def = _cache;
1201                                break;
1202                        case SHT_SUNW_verneed:
1203                                if (rflag)
1204                                        _cache_need = _cache;
1205                                break;
1206                        case SHT_SUNW_versym:
1207                                if (sflag)
1208                                        _cache_sym = _cache;
1209                                break;
1210                        case SHT_SYMTAB:
1211                                if (lflag)
1212                                        _cache_loc = _cache;
1213                                break;
1214                        }
1215                }

1217                /*
1218                 * Before printing anything out determine if any warnings are
1219                 * necessary.
1220                 */
1221                if (lflag && (_cache_loc == NULL)) {
1222                        (void) fprintf(stderr, MSG_INTL(MSG_VER_UNREDSYMS),
1223                            cname, file);
1224                        (void) fprintf(stderr, MSG_INTL(MSG_VER_NOSYMTAB));
1225                }

1227                /*
1228                 * If there is more than one input file, and we're not printing
1229                 * one-line output, display the filename being processed.
1230                 */
1231                if ((nfile > 1) && !oflag)
1232                        (void) printf(MSG_ORIG(MSG_FMT_FILE), file);

1234                /*
1235                 * If we're printing symbols, then collect the data
1236                 * necessary to do that.
1237                 */
1238                if (_cache_sym != NULL) {
1239                        vsdata = &vsdata_s;
1240                        (void) gelf_getshdr(_cache_sym->c_scn, &shdr);
1241                        vsdata_s.vsd_vsp =
1242                            (GElf_Versym *)_cache_sym->c_data->d_buf;
1243                        vsdata_s.vsd_sym_data = cache[shdr.sh_link].c_data;
1244                        (void) gelf_getshdr(cache[shdr.sh_link].c_scn, &shdr);
1245                        vsdata_s.vsd_symn = shdr.sh_size / shdr.sh_entsize;
1246                        vsdata_s.vsd_strs =
1247                            (const char *)cache[shdr.sh_link].c_data->d_buf;
1248                }


1251                /*
1252                 * Print the files version needed sections.
1253                 */
1254                if (_cache_need)
1255                        nerror = gvers_need(cache, _cache_need, vsdata, file);

1257                /*
```

```
1258                  * Print the files version definition sections.
1259                  */
1260                 if (_cache_def)
1261                         derror = gvers_def(cache, _cache_def, vsdata, file);

1263                 /*
1264                  * Print any local symbol reductions.
1265                  */
1266                 if (_cache_loc)
1267                         sym_local(cache, _cache_loc, file);

1269                 /*
1270                  * Determine the error return.  There are three conditions that
1271                  * may produce an error (a non-zero return):
1272                  *
1273                  *  o   if the user specified -d and no version definitions
1274                  *      were found.
1275                  *
1276                  *  o   if the user specified -r and no version requirements
1277                  *      were found.
1278                  *
1279                  *  o   if the user specified neither -d or -r, (thus both are
1280                  *      enabled by default), and no version definitions or
1281                  *      version dependencies were found.
1282                  */
1283                 if (((dflag == USR_DEFINED) && (derror == 0)) ||
1284                     ((rflag == USR_DEFINED) && (nerror == 0)) ||
1285                     (rflag && dflag && (derror == 0) && (nerror == 0)))
1286                         error = 1;

1288                 (void) close(var);
1289                 (void) elf_end(elf);
1290                 free(cache);
1291         }
1292         return (error);
1293 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    3475 Fri Aug  9 16:25:52 2013
new/usr/src/cmd/sgs/pvs/common/pvs.msg
4000 pvs can't deal with extended sections
**********************************************************
```

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

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

@ _START_

# Message file for cmd/sgs/pvs.

@ MSG_ID_PVS


# Argument usage messages.

@ MSG_USAGE_BRIEF       "usage: %s [-Cdlnorsv] [-I index] [-N Name] file(s)\n"
@ MSG_USAGE_DETAIL      "\t[-C]\t\tdemangle C++ symbol names\n\
                         \t[-d]\t\tprint version definition information\n\
                         \t[-I index]\tqualify version with an index\n\
                         \t[-l]\t\tprint reduced symbols\n\
                         \t[-N name]\tqualify version with 'name'\n\
                         \t[-n]\t\tnormalize version definition information\n\
                         \t[-o]\t\tone-line version definition output\n\
                         \t[-r]\t\tprint version dependency (requirements) \
                          information\n\
                         \t[-s]\t\tprint version definition symbols\n\
                         \t[-v]\t\tprint verbose information\n"


# Versioning messages.

@ MSG_VER_HIGHREV       "%s: warning: %s: version revision %d is \
                         higher than expected %d\n"
@ MSG_VER_UNREDSYMS     "%s: warning: %s: unable to deduce reduced symbols:\n"
@ MSG_VER_NOTSTTFILE    "\tmalformed symbol table[1]=%s: does not equal \
                         STT_FILE\n"
@ MSG_VER_NOSYMTAB      "\tno symbol table (SHT_SYMTAB) found\n"


# ELF messages
@ MSG_ELF_NOTELF        "%s: %s: is not ELF\n"
```



```
# System error messages

@ MSG_SYS_OPEN          "%s: %s: cannot open file: %s\n"
@ MSG_SYS_MALLOC        "%s: %s: malloc failed: %s\n"
@ MSG_STR_MATCH_RECORD  "match record (-I/-N)"

@ _END_

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

@ MSG_ELF_BEGIN         "%s: %s: elf_begin: %s\n"
@ MSG_ELF_GETEHDR       "%s: %s: elf_getehdr: %s\n"
@ MSG_ELF_GETSCN        "%s: %s: elf_getscn: %s\n"
@ MSG_ELF_GETSHDR       "%s: %s: elf_getshdr: %s\n"
@ MSG_ELF_GETDATA       "%s: %s: elf_getdata: %s\n"
@ MSG_ELF_GETSHDRNUM    "%s: %s: elf_getshdrnum: %s\n"
#endif /* ! codereview */

@ MSG_STR_EMPTY         ""
@ MSG_STR_OPTIONS       "CdI:lnorsvN:"

@ MSG_FMT_COL_NL        ":\n"
@ MSG_FMT_SEM_NL        ";\n"
@ MSG_FMT_LIST_BEGIN    "\t%s ("
@ MSG_FMT_LIST_FIRST    "%s"
@ MSG_FMT_LIST_NEXT     ", %s"
@ MSG_FMT_LIST_END_SEM  ");\n"
@ MSG_FMT_LIST_END_COL  "):\n"
@ MSG_FMT_OFIL          "%s -"
@ MSG_FMT_TNCO          "\t%s:\n"
@ MSG_FMT_TNSE          "\t%s;\n"
@ MSG_FMT_VER_NAME      "\t%s"
@ MSG_FMT_VER_FLG       " [%s]"

@ MSG_FMT_FILE          "%s:\n"

@ MSG_FMT_SYM_OFIL      "%s -\t%s: "
@ MSG_FMT_SYM_NEED_OFIL "%s -\t%s (%s): "
@ MSG_FMT_SYM_OFLG      "%s;\n"
@ MSG_FMT_SYM_SZ_OFLG   "%s (%ld);\n"
@ MSG_FMT_SYM           "\t\t%s;\n"
@ MSG_FMT_SYM_SZ        "\t\t%s (%ld);\n"

@ MSG_FMT_IN            ":       \t{%s"
@ MSG_FMT_IN_OFLG       ": {%s"
@ MSG_FMT_IN_WEAK       ":\t{%s"
@ MSG_FMT_IN_END        "}"

@ MSG_FMT_LOCSYM_HDR    "\t_LOCAL_:\n"
@ MSG_FMT_LOCSYM        "\t\t%s;\n"
@ MSG_FMT_LOCSYM_OFLG   "%s -\t_LOCAL_: %s\n"

@ MSG_SUNW_OST_SGS      "SUNW_OST_SGS"
```

```
**********************************************************
    8198 Fri Aug  9 16:25:52 2013
new/usr/src/cmd/sgs/size/common/process.c
4009 size(1) can't find sections in relocatable objects with extended sections
**********************************************************
______unchanged_portion_omitted_

  65 static void     process_phdr(Elf *elf, GElf_Half num);

  67 void
  68 process(Elf * elf)
  69 {
  70         /* EXTERNAL VARIABLES USED */
  71         extern int      fflag; /* full format for sections */
  72         extern int      Fflag; /* full format for segments */
  73         extern int      nflag; /* include non-loadable segments or sections */
  71         extern int      fflag, /* full format for sections */
  72                         Fflag, /* full format for segments */
  73                         nflag; /* include non-loadable segments or sections */
  74         extern int      numbase; /* hex, octal, or decimal */
  75         extern char     *fname;
  76         extern char     *archive;
  77         extern int      is_archive;
  78         extern int      oneflag;

  80         /* LOCAL VARIABLES */
  81         GElf_Xword      size; /* total size in non-default case for sections */
  81         GElf_Xword      size, /* total size in non-default case for sections */
  82         /*
  83          * size of first, second, third number and total size
  84          * in default case for sections.
  85          */
  86         GElf_Xword      first;
  87         GElf_Xword      second;
  88         GElf_Xword      third;
  89         GElf_Xword      totsize;
  86                         first,
  87                         second,
  88                         third,
  89                         totsize;
  90         GElf_Ehdr       ehdr;
  91         GElf_Shdr       shdr;
  92         Elf_Scn         *scn;
  93         size_t          ndx = 0, shnum = 0;
  93         unsigned        ndx = 0;
  94         int             numsect = 0;
  95         int             notfirst = 0;
  96         int             i;
  97         char            *name = 0;

 100 #endif /* ! codereview */
 101 /*
 102  * If there is a program header and the -f flag requesting section infor-
 103  * mation is not set, then process segments with the process_phdr function.
 104  * Otherwise, process sections.  For the default case, the first number
 105  * shall be the size of all sections that are allocatable, nonwritable and
 106  * not of type NOBITS; the second number shall be the size of all sections
 107  * that are allocatable, writable, and not of type NOBITS; the third number
 108  * is the size of all sections that are writable and not of type NOBITS.
 109  * If -f is set, print the size of each allocatable section, followed by
 110  * the section name in parentheses.
 111  * If -n is set, print the size of all sections, followed by the section
 112  * name in parentheses.
 113  */
```



```
 115         if (gelf_getehdr(elf, &ehdr) == 0) {
 116                 error(fname, "invalid file type");
 117                 return;
 118         }
 119         if ((ehdr.e_phnum != 0) && !(fflag)) {
 120                 process_phdr(elf, ehdr.e_phnum);
 121                 return;
 122         }

 124         if (is_archive) {
 125                 (void) printf("%s[%s]: ", archive, fname);
 126         } else if (!oneflag && !is_archive) {
 127                 (void) printf("%s: ", fname);
 128         }
 129         if (elf_getshdrstrndx(elf, &ndx) == -1)
 130                 error(fname, "no string table");
  99         ndx = ehdr.e_shstrndx;
 131         scn = 0;
 132         size = 0;
 133         first = second = third = totsize = 0;

 135         if (elf_getshdrnum(elf, &shnum) == -1)
 136                 error(fname, "can't get number of sections");

 138         if (shnum == 0)
 103         if (ehdr.e_shnum == 0) {
 139                 error(fname, "no section data");

 141         numsect = shnum;
 105         }
 106         numsect = ehdr.e_shnum;
 142         for (i = 0; i < numsect; i++) {
 143                 if ((scn = elf_nextscn(elf, scn)) == 0) {
 144                         break;
 145                 }
 146                 if (gelf_getshdr(scn, &shdr) == 0) {
 147                         error(fname, "could not get section header");
 148                         break;
 149                 }
 150                 if ((Fflag) && !(fflag)) {
 151                         error(fname, "no segment data");
 152                         return;
 153                 } else if ((!(shdr.sh_flags & SHF_ALLOC)) &&
 154                     fflag && !(nflag)) {
 155                         continue;
 156                 } else if ((!(shdr.sh_flags & SHF_ALLOC)) && !(nflag)) {
 157                         continue;
 158                 } else if ((shdr.sh_flags & SHF_ALLOC) &&
 159                     (!(shdr.sh_flags & SHF_WRITE)) &&
 160                     (!(shdr.sh_type == SHT_NOBITS)) &&
 161                     !(fflag) && !(nflag)) {
 162                         first += shdr.sh_size;
 163                 } else if ((shdr.sh_flags & SHF_ALLOC) &&
 164                     (shdr.sh_flags & SHF_WRITE) &&
 165                     (!(shdr.sh_type == SHT_NOBITS)) &&
 166                     !(fflag) && !(nflag)) {
 167                         second += shdr.sh_size;
 168                 } else if ((shdr.sh_flags & SHF_WRITE) &&
 169                     (shdr.sh_type == SHT_NOBITS) &&
 170                     !(fflag) && !(nflag)) {
 171                         third += shdr.sh_size;
 172                 }
 173                 name = elf_strptr(elf, ndx, (size_t)shdr.sh_name);

 175                 if (fflag || nflag) {
 176                         size += shdr.sh_size;
```

```
177                         if (notfirst) {
178                                 (void) printf(" + ");
179                         }
180                         (void) printf(prusect[numbase], shdr.sh_size);
181                         (void) printf("(%s)", name);
182                 }
183                 notfirst++;
184         }
185         if ((fflag || nflag) && (numsect > 0)) {
186                 (void) printf(prusum[numbase], size);
187         }

189         if (!fflag && !nflag) {
190                 totsize = first + second + third;
191                 (void) printf(format[numbase],
192                     first, second, third, totsize);
193         }

195         if (Fflag) {
196                 if (ehdr.e_phnum != 0) {
197                         process_phdr(elf, ehdr.e_phnum);
198                         return;
199                 } else {
200                         error(fname, "no segment data");
201                         return;
202                 }
203         }
204 }

206 /*
207  * If there is a program exection header, process segments. In the default
208  * case, the first number is the file size of all nonwritable segments
209  * of type PT_LOAD; the second number is the file size of all writable
210  * segments whose type is PT_LOAD; the third number is the memory size
211  * minus the file size of all writable segments of type PT_LOAD.
212  * If the -F flag is set, size will print the memory size of each loadable
213  * segment, followed by its permission flags.
214  * If -n is set, size will print the memory size of all loadable segments
215  * and the file size of all non-loadable segments, followed by their
216  * permission flags.
217  */

219 static void
220 process_phdr(Elf * elf, GElf_Half num)
221 {
222         int             i;
223         int             notfirst = 0;
224         GElf_Phdr       p;
225         GElf_Xword      memsize;
226         GElf_Xword      total;
227         GElf_Xword      First;
228         GElf_Xword      Second;
229         GElf_Xword      Third;
230         GElf_Xword      Totsize;
190         GElf_Xword      memsize,
191                         total,
192                         First,
193                         Second,
194                         Third,
195                         Totsize;
231         extern int Fflag;
232         extern int nflag;
233         extern int numbase;
234         extern char *fname;
235         extern char *archive;
236         extern int is_archive;
```



```
237         extern int oneflag;

239         memsize = total = 0;
240         First = Second = Third = Totsize = 0;

242         if (is_archive) {
243                 (void) printf("%s[%s]: ", archive, fname);
244         } else if (!oneflag && !is_archive) {
245                 (void) printf("%s: ", fname);
246         }

248         for (i = 0; i < (int)num; i++) {
249                 if (gelf_getphdr(elf, i, &p) == NULL) {
250                         error(fname, "no segment data");
251                         return;
252                 }
253                 if ((!(p.p_flags & PF_W)) &&
254                     (p.p_type == PT_LOAD) && !(Fflag)) {
255                         First += p.p_filesz;
256                 } else if ((p.p_flags & PF_W) &&
257                     (p.p_type == PT_LOAD) && !(Fflag)) {
258                         Second += p.p_filesz;
259                         Third += p.p_memsz;
260                 }
261                 memsize += p.p_memsz;
262                 if ((p.p_type == PT_LOAD) && nflag) {
263                         if (notfirst) {
264                                 (void) printf(" + ");
265                         }
266                         (void) printf(prusect[numbase], p.p_memsz);
267                         total += p.p_memsz;
268                         notfirst++;
269                 }
270                 if (!(p.p_type == PT_LOAD) && nflag) {
271                         if (notfirst) {
272                                 (void) printf(" + ");
273                         }
274                         (void) printf(prusect[numbase], p.p_filesz);
275                         total += p.p_filesz;
276                         notfirst++;
277                 }
278                 if ((p.p_type == PT_LOAD) && Fflag && !nflag) {
279                         if (notfirst) {
280                                 (void) printf(" + ");
281                         }
282                         (void) printf(prusect[numbase], p.p_memsz);
283                         notfirst++;
284                 }
285                 if ((Fflag) && !(nflag) && (!(p.p_type == PT_LOAD))) {
286                         continue;
287                 }
288                 if (Fflag || nflag) {
289                         switch (p.p_flags) {
290                         case 0: (void) printf("(---)"); break;
291                         case PF_X: (void) printf("(--x)"); break;
292                         case PF_W: (void) printf("(-w-)"); break;
293                         case PF_W+PF_X: (void) printf("(-wx)"); break;
294                         case PF_R: (void) printf("(r--)"); break;
295                         case PF_R+PF_X: (void) printf("(r-x)"); break;
296                         case PF_R+PF_W: (void) printf("(rw-)"); break;
297                         case PF_R+PF_W+PF_X: (void) printf("(rwx)"); break;
298                         default: (void) printf("flags(%#x)", p.p_flags);
299                         }
300                 }
301         }
302         if (nflag) {
```

```
303                 (void) printf(prusum[numbase], total);
304         }
305         if (Fflag && !nflag) {
306                 (void) printf(prusum[numbase], memsize);
307         }
308         if (!Fflag && !nflag) {
309                 Totsize = First + Second + (Third - Second);
310                 (void) printf(format[numbase],
311                     First, Second, Third - Second, Totsize);
312         }
313 }
```

unchanged_portion_omitted_

```
**********************************************************
   13376 Fri Aug  9 16:25:53 2013
new/usr/src/lib/libctf/common/ctf_lib.c
4005 libctf can't deal with extended sections
**********************************************************
______unchanged_portion_omitted_

 186 /*
 187  * Open the specified file descriptor and return a pointer to a CTF container.
 188  * The file can be either an ELF file or raw CTF file.  The caller is
 189  * responsible for closing the file descriptor when it is no longer needed.
 190  */
 191 ctf_file_t *
 192 ctf_fdopen(int fd, int *errp)
 193 {
 194         ctf_sect_t ctfsect, symsect, strsect;
 195         ctf_file_t *fp = NULL;
 196         size_t shstrndx, shnum;
 197 #endif /* ! codereview */

 199         struct stat64 st;
 200         ssize_t nbytes;

 202         union {
 203                 ctf_preamble_t ctf;
 204                 Elf32_Ehdr e32;
 205                 GElf_Ehdr e64;
 206         } hdr;

 208         bzero(&ctfsect, sizeof (ctf_sect_t));
 209         bzero(&symsect, sizeof (ctf_sect_t));
 210         bzero(&strsect, sizeof (ctf_sect_t));
 211         bzero(&hdr.ctf, sizeof (hdr));

 213         if (fstat64(fd, &st) == -1)
 214                 return (ctf_set_open_errno(errp, errno));

 216         if ((nbytes = pread64(fd, &hdr.ctf, sizeof (hdr), 0)) <= 0)
 217                 return (ctf_set_open_errno(errp, nbytes < 0? errno : ECTF_FMT));

 219         /*
 220          * If we have read enough bytes to form a CTF header and the magic
 221          * string matches, attempt to interpret the file as raw CTF.
 222          */
 223         if (nbytes >= sizeof (ctf_preamble_t) &&
 224             hdr.ctf.ctp_magic == CTF_MAGIC) {
 225                 if (hdr.ctf.ctp_version > CTF_VERSION)
 226                         return (ctf_set_open_errno(errp, ECTF_CTFVERS));

 228                 ctfsect.cts_data = mmap64(NULL, st.st_size, PROT_READ,
 229                     MAP_PRIVATE, fd, 0);

 231                 if (ctfsect.cts_data == MAP_FAILED)
 232                         return (ctf_set_open_errno(errp, errno));

 234                 ctfsect.cts_name = _CTF_SECTION;
 235                 ctfsect.cts_type = SHT_PROGBITS;
 236                 ctfsect.cts_flags = SHF_ALLOC;
 237                 ctfsect.cts_size = (size_t)st.st_size;
 238                 ctfsect.cts_entsize = 1;
 239                 ctfsect.cts_offset = 0;

 241                 if ((fp = ctf_bufopen(&ctfsect, NULL, NULL, errp)) == NULL)
 242                         ctf_sect_munmap(&ctfsect);

 244                 return (fp);
```



```
 245         }

 247         /*
 248          * If we have read enough bytes to form an ELF header and the magic
 249          * string matches, attempt to interpret the file as an ELF file.  We
 250          * do our own largefile ELF processing, and convert everything to
 251          * GElf structures so that clients can operate on any data model.
 252          */
 253         if (nbytes >= sizeof (Elf32_Ehdr) &&
 254             bcmp(&hdr.e32.e_ident[EI_MAG0], ELFMAG, SELFMAG) == 0) {
 255 #ifdef  _BIG_ENDIAN
 256                 uchar_t order = ELFDATA2MSB;
 257 #else
 258                 uchar_t order = ELFDATA2LSB;
 259 #endif
 196                 GElf_Half i, n;
 260                 GElf_Shdr *sp;

 262                 void *strs_map;
 263                 size_t strs_mapsz, i;
 200                 size_t strs_mapsz;
 264                 const char *strs;

 266                 if (hdr.e32.e_ident[EI_DATA] != order)
 267                         return (ctf_set_open_errno(errp, ECTF_ENDIAN));
 268                 if (hdr.e32.e_version != EV_CURRENT)
 269                         return (ctf_set_open_errno(errp, ECTF_ELFVERS));

 271                 if (hdr.e32.e_ident[EI_CLASS] == ELFCLASS64) {
 272                         if (nbytes < sizeof (GElf_Ehdr))
 273                                 return (ctf_set_open_errno(errp, ECTF_FMT));
 274                 } else {
 275                         Elf32_Ehdr e32 = hdr.e32;
 276                         ehdr_to_gelf(&e32, &hdr.e64);
 277                 }

 279                 shnum = hdr.e64.e_shnum;
 280                 shstrndx = hdr.e64.e_shstrndx;

 282                 /* Extended ELF sections */
 283                 if ((shstrndx == SHN_XINDEX) || (shnum == 0)) {
 284                         if (hdr.e32.e_ident[EI_CLASS] == ELFCLASS32) {
 285                                 Elf32_Shdr x32;

 287                                 if (pread64(fd, &x32, sizeof (x32),
 288                                     hdr.e64.e_shoff) != sizeof (x32))
 289                                         return (ctf_set_open_errno(errp,
 290                                             errno));

 292                                 shnum = x32.sh_size;
 293                                 shstrndx = x32.sh_link;
 294                         } else {
 295                                 Elf64_Shdr x64;

 297                                 if (pread64(fd, &x64, sizeof (x64),
 298                                     hdr.e64.e_shoff) != sizeof (x64))
 299                                         return (ctf_set_open_errno(errp,
 300                                             errno));

 302                                 shnum = x64.sh_size;
 303                                 shstrndx = x64.sh_link;
 304                         }
 305                 }

 307                 if (shstrndx >= shnum)
 216                 if (hdr.e64.e_shstrndx >= hdr.e64.e_shnum)
```

```
308                         return (ctf_set_open_errno(errp, ECTF_CORRUPT));

310                 nbytes = sizeof (GElf_Shdr) * shnum;
219                 n = hdr.e64.e_shnum;
220                 nbytes = sizeof (GElf_Shdr) * n;

312                 if ((sp = malloc(nbytes)) == NULL)
313                         return (ctf_set_open_errno(errp, errno));

315                 /*
316                  * Read in and convert to GElf the array of Shdr structures
317                  * from e_shoff so we can locate sections of interest.
318                  */
319                 if (hdr.e32.e_ident[EI_CLASS] == ELFCLASS32) {
320                         Elf32_Shdr *sp32;

322                         nbytes = sizeof (Elf32_Shdr) * shnum;
232                         nbytes = sizeof (Elf32_Shdr) * n;

324                         if ((sp32 = malloc(nbytes)) == NULL || pread64(fd,
325                             sp32, nbytes, hdr.e64.e_shoff) != nbytes) {
326                                 free(sp);
327                                 return (ctf_set_open_errno(errp, errno));
328                         }

330                         for (i = 0; i < shnum; i++)
240                         for (i = 0; i < n; i++)
331                                 shdr_to_gelf(&sp32[i], &sp[i]);

333                         free(sp32);

335                 } else if (pread64(fd, sp, nbytes, hdr.e64.e_shoff) != nbytes) {
336                         free(sp);
337                         return (ctf_set_open_errno(errp, errno));
338                 }

340                 /*
341                  * Now mmap the section header strings section so that we can
342                  * perform string comparison on the section names.
343                  */
344                 strs_mapsz = sp[shstrndx].sh_size +
345                     (sp[shstrndx].sh_offset & ~_PAGEMASK);
254                 strs_mapsz = sp[hdr.e64.e_shstrndx].sh_size +
255                     (sp[hdr.e64.e_shstrndx].sh_offset & ~_PAGEMASK);

347                 strs_map = mmap64(NULL, strs_mapsz, PROT_READ, MAP_PRIVATE,
348                     fd, sp[shstrndx].sh_offset & _PAGEMASK);
258                     fd, sp[hdr.e64.e_shstrndx].sh_offset & _PAGEMASK);

350                 strs = (const char *)strs_map +
351                     (sp[shstrndx].sh_offset & ~_PAGEMASK);
261                     (sp[hdr.e64.e_shstrndx].sh_offset & ~_PAGEMASK);

353                 if (strs_map == MAP_FAILED) {
354                         free(sp);
355                         return (ctf_set_open_errno(errp, ECTF_MMAP));
356                 }

358                 /*
359                  * Iterate over the section header array looking for the CTF
360                  * section and symbol table.  The strtab is linked to symtab.
361                  */
362                 for (i = 0; i < shnum; i++) {
272                 for (i = 0; i < n; i++) {
363                         const GElf_Shdr *shp = &sp[i];
364                         const GElf_Shdr *lhp = &sp[shp->sh_link];
```



```
366                         if (shp->sh_link >= shnum)
276                         if (shp->sh_link >= hdr.e64.e_shnum)
367                                 continue; /* corrupt sh_link field */

369                         if (shp->sh_name >= sp[shstrndx].sh_size ||
370                             lhp->sh_name >= sp[shstrndx].sh_size)
279                         if (shp->sh_name >= sp[hdr.e64.e_shstrndx].sh_size ||
280                             lhp->sh_name >= sp[hdr.e64.e_shstrndx].sh_size)
371                                 continue; /* corrupt sh_name field */

373                         if (shp->sh_type == SHT_PROGBITS &&
374                             strcmp(strs + shp->sh_name, _CTF_SECTION) == 0) {
375                                 ctfsect.cts_name = strs + shp->sh_name;
376                                 ctfsect.cts_type = shp->sh_type;
377                                 ctfsect.cts_flags = shp->sh_flags;
378                                 ctfsect.cts_size = shp->sh_size;
379                                 ctfsect.cts_entsize = shp->sh_entsize;
380                                 ctfsect.cts_offset = (off64_t)shp->sh_offset;

382                         } else if (shp->sh_type == SHT_SYMTAB) {
383                                 symsect.cts_name = strs + shp->sh_name;
384                                 symsect.cts_type = shp->sh_type;
385                                 symsect.cts_flags = shp->sh_flags;
386                                 symsect.cts_size = shp->sh_size;
387                                 symsect.cts_entsize = shp->sh_entsize;
388                                 symsect.cts_offset = (off64_t)shp->sh_offset;

390                                 strsect.cts_name = strs + lhp->sh_name;
391                                 strsect.cts_type = lhp->sh_type;
392                                 strsect.cts_flags = lhp->sh_flags;
393                                 strsect.cts_size = lhp->sh_size;
394                                 strsect.cts_entsize = lhp->sh_entsize;
395                                 strsect.cts_offset = (off64_t)lhp->sh_offset;
396                         }
397                 }

399                 free(sp); /* free section header array */

401                 if (ctfsect.cts_type == SHT_NULL) {
402                         (void) munmap(strs_map, strs_mapsz);
403                         return (ctf_set_open_errno(errp, ECTF_NOCTFDATA));
404                 }

406                 /*
407                  * Now mmap the CTF data, symtab, and strtab sections and
408                  * call ctf_bufopen() to do the rest of the work.
409                  */
410                 if (ctf_sect_mmap(&ctfsect, fd) == MAP_FAILED) {
411                         (void) munmap(strs_map, strs_mapsz);
412                         return (ctf_set_open_errno(errp, ECTF_MMAP));
413                 }

415                 if (symsect.cts_type != SHT_NULL &&
416                     strsect.cts_type != SHT_NULL) {
417                         if (ctf_sect_mmap(&symsect, fd) == MAP_FAILED ||
418                             ctf_sect_mmap(&strsect, fd) == MAP_FAILED) {
419                                 (void) ctf_set_open_errno(errp, ECTF_MMAP);
420                                 goto bad; /* unmap all and abort */
421                         }
422                         fp = ctf_bufopen(&ctfsect, &symsect, &strsect, errp);
423                 } else
424                         fp = ctf_bufopen(&ctfsect, NULL, NULL, errp);
425 bad:
426                 if (fp == NULL) {
427                         ctf_sect_munmap(&ctfsect);
```

```
 428                         ctf_sect_munmap(&symsect);
 429                         ctf_sect_munmap(&strsect);
 430                 } else
 431                         fp->ctf_flags |= LCTF_MMAP;

 433                 (void) munmap(strs_map, strs_mapsz);
 434                 return (fp);
 435         }

 437         return (ctf_set_open_errno(errp, ECTF_FMT));
 438 }
```

_____***unchanged_portion_omitted_***