```
**********************************************************
   96381 Mon Jul  7 11:53:56 2014
new/usr/src/cmd/sgs/libld/common/sections.c
4959 completely discarded merged string sections will corrupt output objects
**********************************************************
______unchanged_portion_omitted_
```

```
163 static Boolean
164 isdesc_discarded(Is_desc *isp)
165 {
166         Ifl_desc        *ifl = isp->is_file;
167         Os_desc         *osp = isp->is_osdesc;
168         Word            ptype = osp->os_sgdesc->sg_phdr.p_type;
169
170         if (isp->is_flags & FLG_IS_DISCARD)
171                 return TRUE;
172
173         /*
174          * If the file is discarded, it will take
175          * the section with it.
176          */
177         if (ifl &&
178             (((ifl->ifl_flags & FLG_IF_FILEREF) == 0) ||
179             ((ptype == PT_LOAD) &&
180             ((isp->is_flags & FLG_IS_SECTREF) == 0) &&
181             (isp->is_shdr->sh_size > 0))) &&
182             (ifl->ifl_flags & FLG_IF_IGNORE))
183                 return TRUE;
184
185         return FALSE;
186 }
187
188 #endif /* ! codereview */
189 /*
190  * There are situations where we may count output sections (ofl_shdrcnt)
191  * that are subsequently eliminated from the output object. Whether or
192  * not this happens cannot be known until all input has been seen and
193  * section elimination code has run. However, the situations where this
194  * outcome is possible are known, and are flagged by setting FLG_OF_ADJOSCNT.
195  *
196  * If FLG_OF_ADJOSCNT is set, this routine makes a pass over the output
197  * sections. If an unused output section is encountered, we decrement
198  * ofl->ofl_shdrcnt and remove the section name from the .shstrtab string
199  * table (ofl->ofl_shdrsttab).
200  *
201  * This code must be kept in sync with the similar code
202  * found in outfile.c:ld_create_outfile().
203  */
204 static void
205 adjust_os_count(Ofl_desc *ofl)
206 {
207         Sg_desc         *sgp;
208         Is_desc         *isp;
209         Os_desc         *osp;
163         Ifl_desc        *ifl;
210         Aliste          idx1;
211
212         if ((ofl->ofl_flags & FLG_OF_ADJOSCNT) == 0)
213                 return;
214
215         /*
216          * For each output section, look at the input sections to find at least
217          * one input section that has not been eliminated. If none are found,
218          * the -z ignore processing above has eliminated that output section.
219          */
220         for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
```



```
221                 Aliste  idx2;
176                 Word    ptype = sgp->sg_phdr.p_type;
222
223                 for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
224                         Aliste  idx3;
225                         int     keep = 0, os_isdescs_idx;
226
227                         OS_ISDESCS_TRAVERSE(os_isdescs_idx, osp, idx3, isp) {
183                                 ifl = isp->is_file;
184
185                                 /* Input section is tagged for discard? */
186                                 if (isp->is_flags & FLG_IS_DISCARD)
187                                         continue;
188
189                                 /*
190                                  * If the file is discarded, it will take
191                                  * the section with it.
192                                  */
193                                 if (ifl &&
194                                     (((ifl->ifl_flags & FLG_IF_FILEREF) == 0) ||
195                                     ((ptype == PT_LOAD) &&
196                                     ((isp->is_flags & FLG_IS_SECTREF) == 0) &&
197                                     (isp->is_shdr->sh_size > 0))) &&
198                                     (ifl->ifl_flags & FLG_IF_IGNORE))
199                                         continue;
200
228                                 /*
229                                  * We have found a kept input section,
230                                  * so the output section will be created.
231                                  */
232                                 if (!isdesc_discarded(isp)) {
233 #endif /* ! codereview */
234                                         keep = 1;
235                                         break;
236                                 }
237                         }
238 #endif /* ! codereview */
239                         /*
240                          * If no section of this name was kept, decrement
241                          * the count and remove the name from .shstrtab.
242                          */
243                         if (keep == 0) {
244                                 /* LINTED - only used for assert() */
245                                 int err;
246
247                                 ofl->ofl_shdrcnt--;
248                                 err = st_delstring(ofl->ofl_shdrsttab,
249                                     osp->os_name);
250                                 assert(err != -1);
251                         }
252                 }
253         }
254 }
255
256 /*
257  * If -zignore has been in effect, scan all input files to determine if the
258  * file, or sections from the file, have been referenced.  If not, the file or
259  * some of the files sections can be discarded. If sections are to be
260  * discarded, rescan the output relocations and the symbol table and remove
261  * the relocations and symbol entries that are no longer required.
262  *
263  * Note:  It's possible that a section which is being discarded has contributed
264  *        to the GOT table or the PLT table.  However, we can't at this point
265  *        eliminate the corresponding entries.  This is because there could well
266  *        be other sections referencing those same entries, but we don't have
267  *        the infrastructure to determine this.  So, keep the PLT and GOT
```

```
 *        entries in the table in case someone wants them.
 * Note:  The section to be affected needs to be allocatable.
 *        So even if -zignore is in effect, if the section is not allocatable,
 *        we do not eliminate it.
 */
static uintptr_t
ignore_section_processing(Ofl_desc *ofl)
{
        Sg_desc         *sgp;
        Is_desc         *isp;
        Os_desc         *osp;
        Ifl_desc        *ifl;
        Rel_cachebuf    *rcbp;
        Rel_desc        *rsp;
        int             allow_ldynsym = OFL_ALLOW_LDYNSYM(ofl);
        Aliste          idx1;

        for (APLIST_TRAVERSE(ofl->ofl_objs, idx1, ifl)) {
                uint_t  num, discard;

                /*
                 * Diagnose (-D unused) a completely unreferenced file.
                 */
                if ((ifl->ifl_flags & FLG_IF_FILEREF) == 0)
                        DBG_CALL(Dbg_unused_file(ofl->ofl_lml,
                            ifl->ifl_name, 0, 0));
                if (((ofl->ofl_flags1 & FLG_OF1_IGNPRC) == 0) ||
                    ((ifl->ifl_flags & FLG_IF_IGNORE) == 0))
                        continue;

                /*
                 * Before scanning the whole symbol table to determine if
                 * symbols should be discard - quickly (relatively) scan the
                 * sections to determine if any are to be discarded.
                 */
                discard = 0;
                if (ifl->ifl_flags & FLG_IF_FILEREF) {
                        for (num = 1; num < ifl->ifl_shnum; num++) {
                                if (((isp = ifl->ifl_isdesc[num]) != NULL) &&
                                    ((isp->is_flags & FLG_IS_SECTREF) == 0) &&
                                    ((osp = isp->is_osdesc) != NULL) &&
                                    ((sgp = osp->os_sgdesc) != NULL) &&
                                    (sgp->sg_phdr.p_type == PT_LOAD)) {
                                        discard++;
                                        break;
                                }
                        }
                }

                /*
                 * No sections are to be 'ignored'
                 */
                if ((discard == 0) && (ifl->ifl_flags & FLG_IF_FILEREF))
                        continue;

                /*
                 * We know that we have discarded sections.  Scan the symbol
                 * table for this file to determine if symbols need to be
                 * discarded that are associated with the 'ignored' sections.
                 */
                for (num = 1; num < ifl->ifl_symscnt; num++) {
                        Sym_desc        *sdp;

                        /*
                         * If the symbol definition has been resolved to another
                         * file, or the symbol has already been discarded or
```



```
                         * eliminated, skip it.
                         */
                        sdp = ifl->ifl_oldndx[num];
                        if ((sdp->sd_file != ifl) ||
                            (sdp->sd_flags &
                            (FLG_SY_ISDISC | FLG_SY_INVALID | FLG_SY_ELIM)))
                                continue;

                        /*
                         * Complete the investigation of the symbol.
                         */
                        ignore_sym(ofl, ifl, sdp, allow_ldynsym);
                }
        }

        /*
         * If we were only here to solicit debugging diagnostics, we're done.
         */
        if ((ofl->ofl_flags1 & FLG_OF1_IGNPRC) == 0)
                return (1);

        /*
         * Scan all output relocations searching for those against discarded or
         * ignored sections.  If one is found, decrement the total outrel count.
         */
        REL_CACHE_TRAVERSE(&ofl->ofl_outrels, idx1, rcbp, rsp) {
                Is_desc         *isc = rsp->rel_isdesc;
                uint_t          flags, entsize;
                Shdr            *shdr;

                if ((isc == NULL) || ((isc->is_flags & (FLG_IS_SECTREF))) ||
                    ((ifl = isc->is_file) == NULL) ||
                    ((ifl->ifl_flags & FLG_IF_IGNORE) == 0) ||
                    ((shdr = isc->is_shdr) == NULL) ||
                    ((shdr->sh_flags & SHF_ALLOC) == 0))
                        continue;

                flags = rsp->rel_flags;

                if (flags & (FLG_REL_GOT | FLG_REL_BSS |
                    FLG_REL_NOINFO | FLG_REL_PLT))
                        continue;

                osp = RELAUX_GET_OSDESC(rsp);

                if (rsp->rel_flags & FLG_REL_RELA)
                        entsize = sizeof (Rela);
                else
                        entsize = sizeof (Rel);

                assert(osp->os_szoutrels > 0);
                osp->os_szoutrels -= entsize;

                if (!(flags & FLG_REL_PLT))
                        ofl->ofl_reloccntsub++;

                if (rsp->rel_rtype == ld_targ.t_m.m_r_relative)
                        ofl->ofl_relocrelcnt--;
        }

        /*
         * As a result of our work here, the number of output sections may
         * have decreased. Trigger a call to adjust_os_count().
         */
        ofl->ofl_flags |= FLG_OF_ADJOSCNT;
```

```
400         return (1);
401 }

403 /*
404  * Allocate Elf_Data, Shdr, and Is_desc structures for a new
405  * section.
406  *
407  * entry:
408  *      ofl - Output file descriptor
409  *      shtype - SHT_ type code for section.
410  *      shname - String giving the name for the new section.
411  *      entcnt - # of items contained in the data part of the new section.
412  *              This value is multiplied against the known element size
413  *              for the section type to determine the size of the data
414  *              area for the section. It is only meaningful in cases where
415  *              the section type has a non-zero element size. In other cases,
416  *              the caller must set the size fields in the *ret_data and
417  *              *ret_shdr structs manually.
418  *      ret_isec, ret_shdr, ret_data - Address of pointers to
419  *              receive address of newly allocated structs.
420  *
421  * exit:
422  *      On error, returns S_ERROR. On success, returns (1), and the
423  *      ret_ pointers have been updated to point at the new structures,
424  *      which have been filled in. To finish the task, the caller must
425  *      update any fields within the supplied descriptors that differ
426  *      from its needs, and then call ld_place_section().
427  */
428 static uintptr_t
429 new_section(Ofl_desc *ofl, Word shtype, const char *shname, Xword entcnt,
430         Is_desc **ret_isec, Shdr **ret_shdr, Elf_Data **ret_data)
431 {
432         typedef struct sec_info {
433                 Word d_type;
434                 Word align;     /* Used in both data and section header */
435                 Word sh_flags;
436                 Word sh_entsize;
437         } SEC_INFO_T;

439         const SEC_INFO_T        *sec_info;

441         Shdr            *shdr;
442         Elf_Data        *data;
443         Is_desc         *isec;
444         size_t          size;

446         /*
447          * For each type of section, we have a distinct set of
448          * SEC_INFO_T values. This macro defines a static structure
449          * containing those values and generates code to set the sec_info
450          * pointer to refer to it. The pointer in sec_info remains valid
451          * outside of the declaration scope because the info_s struct is static.
452          *
453          * We can't determine the value of M_WORD_ALIGN at compile time, so
454          * a different variant is used for those cases.
455          */
456 #define SET_SEC_INFO(d_type, d_align, sh_flags, sh_entsize) \
457         { \
458                 static const SEC_INFO_T info_s = { d_type, d_align, sh_flags, \
459                     sh_entsize}; \
460                 sec_info = &info_s; \
461         }
462 #define SET_SEC_INFO_WORD_ALIGN(d_type, sh_flags, sh_entsize) \
463         { \
464                 static SEC_INFO_T info_s = { d_type, 0, sh_flags, \
465                     sh_entsize}; \
```



```
466                 info_s.align = ld_targ.t_m.m_word_align; \
467                 sec_info = &info_s; \
468         }

470         switch (shtype) {
471         case SHT_PROGBITS:
472                 /*
473                  * SHT_PROGBITS sections contain are used for many
474                  * different sections. Alignments and flags differ.
475                  * Some have a standard entsize, and others don't.
476                  * We set some defaults here, but there is no expectation
477                  * that they are correct or complete for any specific
478                  * purpose. The caller must provide the correct values.
479                  */
480                 SET_SEC_INFO_WORD_ALIGN(ELF_T_BYTE, SHF_ALLOC, 0)
481                 break;

483         case SHT_SYMTAB:
484                 SET_SEC_INFO_WORD_ALIGN(ELF_T_SYM, 0, sizeof (Sym))
485                 break;

487         case SHT_DYNSYM:
488                 SET_SEC_INFO_WORD_ALIGN(ELF_T_SYM, SHF_ALLOC, sizeof (Sym))
489                 break;

491         case SHT_SUNW_LDYNSYM:
492                 ofl->ofl_flags |= FLG_OF_OSABI;
493                 SET_SEC_INFO_WORD_ALIGN(ELF_T_SYM, SHF_ALLOC, sizeof (Sym))
494                 break;

496         case SHT_STRTAB:
497                 /*
498                  * A string table may or may not be allocable, depending
499                  * on context, so we leave that flag unset and leave it to
500                  * the caller to add it if necessary.
501                  *
502                  * String tables do not have a standard entsize, so
503                  * we set it to 0.
504                  */
505                 SET_SEC_INFO(ELF_T_BYTE, 1, SHF_STRINGS, 0)
506                 break;

508         case SHT_RELA:
509                 /*
510                  * Relocations with an addend (Everything except 32-bit X86).
511                  * The caller is expected to set all section header flags.
512                  */
513                 SET_SEC_INFO_WORD_ALIGN(ELF_T_RELA, 0, sizeof (Rela))
514                 break;

516         case SHT_REL:
517                 /*
518                  * Relocations without an addend (32-bit X86 only).
519                  * The caller is expected to set all section header flags.
520                  */
521                 SET_SEC_INFO_WORD_ALIGN(ELF_T_REL, 0, sizeof (Rel))
522                 break;

524         case SHT_HASH:
525                 SET_SEC_INFO_WORD_ALIGN(ELF_T_WORD, SHF_ALLOC, sizeof (Word))
526                 break;

528         case SHT_SUNW_symsort:
529         case SHT_SUNW_tlssort:
530                 ofl->ofl_flags |= FLG_OF_OSABI;
531                 SET_SEC_INFO_WORD_ALIGN(ELF_T_WORD, SHF_ALLOC, sizeof (Word))
```

```
532                 break;

534         case SHT_DYNAMIC:
535                 /*
536                  * A dynamic section may or may not be allocable, and may or
537                  * may not be writable, depending on context, so we leave the
538                  * flags unset and leave it to the caller to add them if
539                  * necessary.
540                  */
541                 SET_SEC_INFO_WORD_ALIGN(ELF_T_DYN, 0, sizeof (Dyn))
542                 break;

544         case SHT_NOBITS:
545                 /*
546                  * SHT_NOBITS is used for BSS-type sections. The size and
547                  * alignment depend on the specific use and must be adjusted
548                  * by the caller.
549                  */
550                 SET_SEC_INFO(ELF_T_BYTE, 0, SHF_ALLOC | SHF_WRITE, 0)
551                 break;

553         case SHT_INIT_ARRAY:
554         case SHT_FINI_ARRAY:
555         case SHT_PREINIT_ARRAY:
556                 SET_SEC_INFO(ELF_T_ADDR, sizeof (Addr), SHF_ALLOC | SHF_WRITE,
557                     sizeof (Addr))
558                 break;

560         case SHT_SYMTAB_SHNDX:
561                 /*
562                  * Note that these sections are created to be associated
563                  * with both symtab and dynsym symbol tables. However, they
564                  * are non-allocable in all cases, because the runtime
565                  * linker has no need for this information. It is purely
566                  * informational, used by elfdump(1), debuggers, etc.
567                  */
568                 SET_SEC_INFO_WORD_ALIGN(ELF_T_WORD, 0, sizeof (Word));
569                 break;

571         case SHT_SUNW_cap:
572                 ofl->ofl_flags |= FLG_OF_OSABI;
573                 SET_SEC_INFO_WORD_ALIGN(ELF_T_CAP, SHF_ALLOC, sizeof (Cap));
574                 break;

576         case SHT_SUNW_capchain:
577                 ofl->ofl_flags |= FLG_OF_OSABI;
578                 SET_SEC_INFO_WORD_ALIGN(ELF_T_WORD, SHF_ALLOC,
579                     sizeof (Capchain));
580                 break;

582         case SHT_SUNW_capinfo:
583                 ofl->ofl_flags |= FLG_OF_OSABI;
584 #if     _ELF64
585                 SET_SEC_INFO(ELF_T_XWORD, sizeof (Xword), SHF_ALLOC,
586                     sizeof (Capinfo));
587 #else
588                 SET_SEC_INFO(ELF_T_WORD, sizeof (Word), SHF_ALLOC,
589                     sizeof (Capinfo));
590 #endif
591                 break;

593         case SHT_SUNW_move:
594                 ofl->ofl_flags |= FLG_OF_OSABI;
595                 SET_SEC_INFO(ELF_T_BYTE, sizeof (Lword),
596                     SHF_ALLOC | SHF_WRITE, sizeof (Move));
597                 break;
```



```
599         case SHT_SUNW_syminfo:
600                 ofl->ofl_flags |= FLG_OF_OSABI;
601                 /*
602                  * The sh_info field of the SHT_*_syminfo section points
603                  * to the header index of the associated .dynamic section,
604                  * so we also set SHF_INFO_LINK.
605                  */
606                 SET_SEC_INFO_WORD_ALIGN(ELF_T_BYTE,
607                     SHF_ALLOC | SHF_INFO_LINK, sizeof (Syminfo));
608                 break;

610         case SHT_SUNW_verneed:
611         case SHT_SUNW_verdef:
612                 ofl->ofl_flags |= FLG_OF_OSABI;
613                 /*
614                  * The info for verneed and versym happen to be the same.
615                  * The entries in these sections are not of uniform size,
616                  * so we set the entsize to 0.
617                  */
618                 SET_SEC_INFO_WORD_ALIGN(ELF_T_BYTE, SHF_ALLOC, 0);
619                 break;

621         case SHT_SUNW_versym:
622                 ofl->ofl_flags |= FLG_OF_OSABI;
623                 SET_SEC_INFO_WORD_ALIGN(ELF_T_BYTE, SHF_ALLOC,
624                     sizeof (Versym));
625                 break;

627         default:
628                 /* Should not happen: fcn called with unknown section type */
629                 assert(0);
630                 return (S_ERROR);
631         }
632 #undef  SET_SEC_INFO
633 #undef  SET_SEC_INFO_WORD_ALIGN

635         size = entcnt * sec_info->sh_entsize;

637         /*
638          * Allocate and initialize the Elf_Data structure.
639          */
640         if ((data = libld_calloc(sizeof (Elf_Data), 1)) == NULL)
641                 return (S_ERROR);
642         data->d_type = sec_info->d_type;
643         data->d_size = size;
644         data->d_align = sec_info->align;
645         data->d_version = ofl->ofl_dehdr->e_version;

647         /*
648          * Allocate and initialize the Shdr structure.
649          */
650         if ((shdr = libld_calloc(sizeof (Shdr), 1)) == NULL)
651                 return (S_ERROR);
652         shdr->sh_type = shtype;
653         shdr->sh_size = size;
654         shdr->sh_flags = sec_info->sh_flags;
655         shdr->sh_addralign = sec_info->align;
656         shdr->sh_entsize = sec_info->sh_entsize;

658         /*
659          * Allocate and initialize the Is_desc structure.
660          */
661         if ((isec = libld_calloc(1, sizeof (Is_desc))) == NULL)
662                 return (S_ERROR);
663         isec->is_name = shname;
```

```
664         isec->is_shdr = shdr;
665         isec->is_indata = data;


668         *ret_isec = isec;
669         *ret_shdr = shdr;
670         *ret_data = data;
671         return (1);
672 }

674 /*
675  * Use an existing input section as a template to create a new
676  * input section with the same values as the original, other than
677  * the size of the data area which is supplied by the caller.
678  *
679  * entry:
680  *      ofl - Output file descriptor
681  *      ifl - Input file section to use as a template
682  *      size - Size of data area for new section
683  *      ret_isec, ret_shdr, ret_data - Address of pointers to
684  *              receive address of newly allocated structs.
685  *
686  * exit:
687  *      On error, returns S_ERROR. On success, returns (1), and the
688  *      ret_ pointers have been updated to point at the new structures,
689  *      which have been filled in. To finish the task, the caller must
690  *      update any fields within the supplied descriptors that differ
691  *      from its needs, and then call ld_place_section().
692  */
693 static uintptr_t
694 new_section_from_template(Ofl_desc *ofl, Is_desc *tmpl_isp, size_t size,
695         Is_desc **ret_isec, Shdr **ret_shdr, Elf_Data **ret_data)
696 {
697         Shdr            *shdr;
698         Elf_Data        *data;
699         Is_desc         *isec;

701         /*
702          * Allocate and initialize the Elf_Data structure.
703          */
704         if ((data = libld_calloc(sizeof (Elf_Data), 1)) == NULL)
705                 return (S_ERROR);
706         data->d_type = tmpl_isp->is_indata->d_type;
707         data->d_size = size;
708         data->d_align = tmpl_isp->is_shdr->sh_addralign;
709         data->d_version = ofl->ofl_dehdr->e_version;

711         /*
712          * Allocate and initialize the Shdr structure.
713          */
714         if ((shdr = libld_malloc(sizeof (Shdr))) == NULL)
715                 return (S_ERROR);
716         *shdr = *tmpl_isp->is_shdr;
717         shdr->sh_addr = 0;
718         shdr->sh_offset = 0;
719         shdr->sh_size = size;

721         /*
722          * Allocate and initialize the Is_desc structure.
723          */
724         if ((isec = libld_calloc(1, sizeof (Is_desc))) == NULL)
725                 return (S_ERROR);
726         isec->is_name = tmpl_isp->is_name;
727         isec->is_shdr = shdr;
728         isec->is_indata = data;
```



```
731         *ret_isec = isec;
732         *ret_shdr = shdr;
733         *ret_data = data;
734         return (1);
735 }

737 /*
738  * Build a .bss section for allocation of tentative definitions.  Any 'static'
739  * .bss definitions would have been associated to their own .bss sections and
740  * thus collected from the input files.  'global' .bss definitions are tagged
741  * as COMMON and do not cause any associated .bss section elements to be
742  * generated.  Here we add up all these COMMON symbols and generate the .bss
743  * section required to represent them.
744  */
745 uintptr_t
746 ld_make_bss(Ofl_desc *ofl, Xword size, Xword align, uint_t ident)
747 {
748         Shdr            *shdr;
749         Elf_Data        *data;
750         Is_desc         *isec;
751         Os_desc         *osp;
752         Xword           rsize = (Xword)ofl->ofl_relocbsssz;

754         /*
755          * Allocate header structs. We will set the name ourselves below,
756          * and there is no entcnt for a BSS. So, the shname and entcnt
757          * arguments are 0.
758          */
759         if (new_section(ofl, SHT_NOBITS, NULL, 0,
760             &isec, &shdr, &data) == S_ERROR)
761                 return (S_ERROR);

763         data->d_size = (size_t)size;
764         data->d_align = (size_t)align;

766         shdr->sh_size = size;
767         shdr->sh_addralign = align;

769         if (ident == ld_targ.t_id.id_tlsbss) {
770                 isec->is_name = MSG_ORIG(MSG_SCN_TBSS);
771                 ofl->ofl_istlsbss = isec;
772                 shdr->sh_flags |= SHF_TLS;

774         } else if (ident == ld_targ.t_id.id_bss) {
775                 isec->is_name = MSG_ORIG(MSG_SCN_BSS);
776                 ofl->ofl_isbss = isec;

778 #if     defined(_ELF64)
779         } else if ((ld_targ.t_m.m_mach == EM_AMD64) &&
780             (ident == ld_targ.t_id.id_lbss)) {
781                 isec->is_name = MSG_ORIG(MSG_SCN_LBSS);
782                 ofl->ofl_islbss = isec;
783                 shdr->sh_flags |= SHF_AMD64_LARGE;
784 #endif
785         }

787         /*
788          * Retain this .*bss input section as this will be where global symbol
789          * references are added.
790          */
791         if ((osp = ld_place_section(ofl, isec, NULL, ident, NULL)) ==
792             (Os_desc *)S_ERROR)
793                 return (S_ERROR);

795         /*
```

```
796          * If relocations exist against a .*bss section, a section symbol must
797          * be created for the section in the .dynsym symbol table.
798          */
799         if (!(osp->os_flags & FLG_OS_OUTREL)) {
800                 ofl_flag_t      flagtotest;
801
802                 if (ident == ld_targ.t_id.id_tlsbss)
803                         flagtotest = FLG_OF1_TLSOREL;
804                 else
805                         flagtotest = FLG_OF1_BSSOREL;
806
807                 if (ofl->ofl_flags1 & flagtotest) {
808                         ofl->ofl_dynshdrcnt++;
809                         osp->os_flags |= FLG_OS_OUTREL;
810                 }
811         }
812
813         osp->os_szoutrels = rsize;
814         return (1);
815 }
816
817 /*
818  * Build a SHT_{INIT|FINI|PREINIT}ARRAY section (specified via
819  * ld -z *array=name).
820  */
821 static uintptr_t
822 make_array(Ofl_desc *ofl, Word shtype, const char *sectname, APlist *alp)
823 {
824         uint_t          entcount;
825         Aliste          idx;
826         Elf_Data        *data;
827         Is_desc         *isec;
828         Shdr            *shdr;
829         Sym_desc        *sdp;
830         Rel_desc        reld;
831         Rela            reloc;
832         Os_desc         *osp;
833         uintptr_t       ret = 1;
834
835         if (alp == NULL)
836                 return (1);
837
838         entcount = 0;
839         for (APLIST_TRAVERSE(alp, idx, sdp))
840                 entcount++;
841
842         if (new_section(ofl, shtype, sectname, entcount, &isec, &shdr, &data) ==
843             S_ERROR)
844                 return (S_ERROR);
845
846         if ((data->d_buf = libld_calloc(sizeof (Addr), entcount)) == NULL)
847                 return (S_ERROR);
848
849         if (ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_array, NULL) ==
850             (Os_desc *)S_ERROR)
851                 return (S_ERROR);
852
853         osp = isec->is_osdesc;
854
855         if ((ofl->ofl_osinitarray == NULL) && (shtype == SHT_INIT_ARRAY))
856                 ofl->ofl_osinitarray = osp;
857         if ((ofl->ofl_ospreinitarray == NULL) && (shtype == SHT_PREINIT_ARRAY))
858                 ofl->ofl_ospreinitarray = osp;
859         else if ((ofl->ofl_osfiniarray == NULL) && (shtype == SHT_FINI_ARRAY))
860                 ofl->ofl_osfiniarray = osp;
```



```
862         /*
863          * Create relocations against this section to initialize it to the
864          * function addresses.
865          */
866         reld.rel_isdesc = isec;
867         reld.rel_aux = NULL;
868         reld.rel_flags = FLG_REL_LOAD;
869
870         /*
871          * Fabricate the relocation information (as if a relocation record had
872          * been input - see init_rel()).
873          */
874         reld.rel_rtype = ld_targ.t_m.m_r_arrayaddr;
875         reld.rel_roffset = 0;
876         reld.rel_raddend = 0;
877
878         /*
879          * Create a minimal relocation record to satisfy process_sym_reloc()
880          * debugging requirements.
881          */
882         reloc.r_offset = 0;
883         reloc.r_info = ELF_R_INFO(0, ld_targ.t_m.m_r_arrayaddr);
884         reloc.r_addend = 0;
885
886         DBG_CALL(Dbg_reloc_generate(ofl->ofl_lml, osp,
887             ld_targ.t_m.m_rel_sht_type));
888         for (APLIST_TRAVERSE(alp, idx, sdp)) {
889                 reld.rel_sym = sdp;
890
891                 if (ld_process_sym_reloc(ofl, &reld, (Rel *)&reloc, isec,
892                     MSG_INTL(MSG_STR_COMMAND), 0) == S_ERROR) {
893                         ret = S_ERROR;
894                         continue;
895                 }
896
897                 reld.rel_roffset += (Xword)sizeof (Addr);
898                 reloc.r_offset = reld.rel_roffset;
899         }
900
901         return (ret);
902 }
903
904 /*
905  * Build a comment section (-Qy option).
906  */
907 static uintptr_t
908 make_comment(Ofl_desc *ofl)
909 {
910         Shdr            *shdr;
911         Elf_Data        *data;
912         Is_desc         *isec;
913
914         if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_COMMENT), 0,
915             &isec, &shdr, &data) == S_ERROR)
916                 return (S_ERROR);
917
918         data->d_buf = (void *)ofl->ofl_sgsid;
919         data->d_size = strlen(ofl->ofl_sgsid) + 1;
920         data->d_align = 1;
921
922         shdr->sh_size = (Xword)data->d_size;
923         shdr->sh_flags = 0;
924         shdr->sh_addralign = 1;
925
926         return ((uintptr_t)ld_place_section(ofl, isec, NULL,
927             ld_targ.t_id.id_note, NULL));
```

```
928 }
929
930 /*
931  * Make the dynamic section.  Calculate the size of any strings referenced
932  * within this structure, they will be added to the global string table
933  * (.dynstr).  This routine should be called before make_dynstr().
934  *
935  * This routine must be maintained in parallel with update_odynamic()
936  * in update.c
937  */
938 static uintptr_t
939 make_dynamic(Ofl_desc *ofl)
940 {
941         Shdr            *shdr;
942         Os_desc         *osp;
943         Elf_Data        *data;
944         Is_desc         *isec;
945         size_t          cnt = 0;
946         Aliste          idx;
947         Ifl_desc        *ifl;
948         Sym_desc        *sdp;
949         size_t          size;
950         Str_tbl         *strtbl;
951         ofl_flag_t      flags = ofl->ofl_flags;
952         int             not_relobj = !(flags & FLG_OF_RELOBJ);
953         int             unused = 0;
954
955         /*
956          * Select the required string table.
957          */
958         if (OFL_IS_STATIC_OBJ(ofl))
959                 strtbl = ofl->ofl_strtab;
960         else
961                 strtbl = ofl->ofl_dynstrtab;
962
963         /*
964          * Only a limited subset of DT_ entries apply to relocatable
965          * objects. See the comment at the head of update_odynamic() in
966          * update.c for details.
967          */
968         if (new_section(ofl, SHT_DYNAMIC, MSG_ORIG(MSG_SCN_DYNAMIC), 0,
969             &isec, &shdr, &data) == S_ERROR)
970                 return (S_ERROR);
971
972         /*
973          * new_section() does not set SHF_ALLOC.  If we're building anything
974          * besides a relocatable object, then the .dynamic section should
975          * reside in allocatable memory.
976          */
977         if (not_relobj)
978                 shdr->sh_flags |= SHF_ALLOC;
979
980         /*
981          * new_section() does not set SHF_WRITE.  If we're building an object
982          * that specifies an interpretor, then a DT_DEBUG entry is created,
983          * which is initialized to the applications link-map list at runtime.
984          */
985         if (ofl->ofl_osinterp)
986                 shdr->sh_flags |= SHF_WRITE;
987
988         osp = ofl->ofl_osdynamic =
989             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_dynamic, NULL);
990
991         /*
992          * Reserve entries for any needed dependencies.
993          */
```



```
994         for (APLIST_TRAVERSE(ofl->ofl_sos, idx, ifl)) {
995                 if (!(ifl->ifl_flags & (FLG_IF_NEEDED | FLG_IF_NEEDSTR)))
996                         continue;
997
998                 /*
999                  * If this dependency didn't satisfy any symbol references,
1000                 * generate a debugging diagnostic (ld(1) -Dunused can be used
1001                 * to display these).  If this is a standard needed dependency,
1002                 * and -z ignore is in effect, drop the dependency.  Explicitly
1003                 * defined dependencies (i.e., -N dep) don't get dropped, and
1004                 * are flagged as being required to simplify update_odynamic()
1005                 * processing.
1006                 */
1007                 if ((ifl->ifl_flags & FLG_IF_NEEDSTR) ||
1008                     ((ifl->ifl_flags & FLG_IF_DEPREQD) == 0)) {
1009                         if (unused++ == 0)
1010                                 DBG_CALL(Dbg_util_nl(ofl->ofl_lml, DBG_NL_STD));
1011                         DBG_CALL(Dbg_unused_file(ofl->ofl_lml, ifl->ifl_soname,
1012                             (ifl->ifl_flags & FLG_IF_NEEDSTR), 0));
1013
1014                         /*
1015                          * Guidance: Remove unused dependency.
1016                          *
1017                          * If -z ignore is in effect, this warning is not
1018                          * needed because we will quietly remove the unused
1019                          * dependency.
1020                          */
1021                         if (OFL_GUIDANCE(ofl, FLG_OFG_NO_UNUSED) &&
1022                             ((ifl->ifl_flags & FLG_IF_IGNORE) == 0))
1023                                 ld_eprintf(ofl, ERR_GUIDANCE,
1024                                     MSG_INTL(MSG_GUIDE_UNUSED),
1025                                     ifl->ifl_soname);
1026
1027                         if (ifl->ifl_flags & FLG_IF_NEEDSTR)
1028                                 ifl->ifl_flags |= FLG_IF_DEPREQD;
1029                         else if (ifl->ifl_flags & FLG_IF_IGNORE)
1030                                 continue;
1031                 }
1032
1033                 /*
1034                  * If this object requires a DT_POSFLAG_1 entry, reserve it.
1035                  */
1036                 if ((ifl->ifl_flags & MSK_IF_POSFLAG1) && not_relobj)
1037                         cnt++;
1038
1039                 if (st_insert(strtbl, ifl->ifl_soname) == -1)
1040                         return (S_ERROR);
1041                 cnt++;
1042
1043                 /*
1044                  * If the needed entry contains the $ORIGIN token make sure
1045                  * the associated DT_1_FLAGS entry is created.
1046                  */
1047                 if (strstr(ifl->ifl_soname, MSG_ORIG(MSG_STR_ORIGIN))) {
1048                         ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1049                         ofl->ofl_dtflags |= DF_ORIGIN;
1050                 }
1051         }
1052
1053         if (unused)
1054                 DBG_CALL(Dbg_util_nl(ofl->ofl_lml, DBG_NL_STD));
1055
1056         if (not_relobj) {
1057                 /*
1058                  * Reserve entries for any per-symbol auxiliary/filter strings.
1059                  */
```

```
1060                cnt += alist_nitems(ofl->ofl_dtsfltrs);

1062                /*
1063                 * Reserve entries for _init() and _fini() section addresses.
1064                 */
1065                if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_INIT_U),
1066                    SYM_NOHASH, NULL, ofl)) != NULL) &&
1067                    (sdp->sd_ref == REF_REL_NEED) &&
1068                    (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
1069                        sdp->sd_flags |= FLG_SY_UPREQD;
1070                        cnt++;
1071                }
1072                if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_FINI_U),
1073                    SYM_NOHASH, NULL, ofl)) != NULL) &&
1074                    (sdp->sd_ref == REF_REL_NEED) &&
1075                    (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
1076                        sdp->sd_flags |= FLG_SY_UPREQD;
1077                        cnt++;
1078                }

1080                /*
1081                 * Reserve entries for any soname, filter name (shared libs
1082                 * only), run-path pointers, cache names and audit requirements.
1083                 */
1084                if (ofl->ofl_soname) {
1085                        cnt++;
1086                        if (st_insert(strtbl, ofl->ofl_soname) == -1)
1087                                return (S_ERROR);
1088                }
1089                if (ofl->ofl_filtees) {
1090                        cnt++;
1091                        if (st_insert(strtbl, ofl->ofl_filtees) == -1)
1092                                return (S_ERROR);

1094                        /*
1095                         * If the filtees entry contains the $ORIGIN token
1096                         * make sure the associated DT_1_FLAGS entry is created.
1097                         */
1098                        if (strstr(ofl->ofl_filtees,
1099                            MSG_ORIG(MSG_STR_ORIGIN))) {
1100                                ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1101                                ofl->ofl_dtflags |= DF_ORIGIN;
1102                        }
1103                }
1104        }

1106        if (ofl->ofl_rpath) {
1107                cnt += 2;       /* DT_RPATH & DT_RUNPATH */
1108                if (st_insert(strtbl, ofl->ofl_rpath) == -1)
1109                        return (S_ERROR);

1111                /*
1112                 * If the rpath entry contains the $ORIGIN token make sure
1113                 * the associated DT_1_FLAGS entry is created.
1114                 */
1115                if (strstr(ofl->ofl_rpath, MSG_ORIG(MSG_STR_ORIGIN))) {
1116                        ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1117                        ofl->ofl_dtflags |= DF_ORIGIN;
1118                }
1119        }

1121        if (not_relobj) {
1122                Aliste  idx;
1123                Sg_desc *sgp;

1125                if (ofl->ofl_config) {
```



```
1126                        cnt++;
1127                        if (st_insert(strtbl, ofl->ofl_config) == -1)
1128                                return (S_ERROR);

1130                        /*
1131                         * If the config entry contains the $ORIGIN token
1132                         * make sure the associated DT_1_FLAGS entry is created.
1133                         */
1134                        if (strstr(ofl->ofl_config, MSG_ORIG(MSG_STR_ORIGIN))) {
1135                                ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1136                                ofl->ofl_dtflags |= DF_ORIGIN;
1137                        }
1138                }
1139                if (ofl->ofl_depaudit) {
1140                        cnt++;
1141                        if (st_insert(strtbl, ofl->ofl_depaudit) == -1)
1142                                return (S_ERROR);
1143                }
1144                if (ofl->ofl_audit) {
1145                        cnt++;
1146                        if (st_insert(strtbl, ofl->ofl_audit) == -1)
1147                                return (S_ERROR);
1148                }

1150                /*
1151                 * Reserve entries for the DT_HASH, DT_STRTAB, DT_STRSZ,
1152                 * DT_SYMTAB, DT_SYMENT, and DT_CHECKSUM.
1153                 */
1154                cnt += 6;

1156                /*
1157                 * If we are including local functions at the head of
1158                 * the dynsym, then also reserve entries for DT_SUNW_SYMTAB
1159                 * and DT_SUNW_SYMSZ.
1160                 */
1161                if (OFL_ALLOW_LDYNSYM(ofl))
1162                        cnt += 2;

1164                if ((ofl->ofl_dynsymsortcnt > 0) ||
1165                    (ofl->ofl_dyntlssortcnt > 0))
1166                        cnt++;          /* DT_SUNW_SORTENT */

1168                if (ofl->ofl_dynsymsortcnt > 0)
1169                        cnt += 2;       /* DT_SUNW_[SYMSORT|SYMSORTSZ] */

1171                if (ofl->ofl_dyntlssortcnt > 0)
1172                        cnt += 2;       /* DT_SUNW_[TLSSORT|TLSSORTSZ] */

1174                if ((flags & (FLG_OF_VERDEF | FLG_OF_NOVERSEC)) ==
1175                    FLG_OF_VERDEF)
1176                        cnt += 2;               /* DT_VERDEF & DT_VERDEFNUM */

1178                if ((flags & (FLG_OF_VERNEED | FLG_OF_NOVERSEC)) ==
1179                    FLG_OF_VERNEED)
1180                        cnt += 2;               /* DT_VERNEED & DT_VERNEEDNUM */

1182                if ((flags & FLG_OF_COMREL) && ofl->ofl_relocrelcnt)
1183                        cnt++;                  /* DT_RELACOUNT */

1185                if (flags & FLG_OF_TEXTREL)     /* DT_TEXTREL */
1186                        cnt++;

1188                if (ofl->ofl_osfiniarray)       /* DT_FINI_ARRAY */
1189                        cnt += 2;               /*    DT_FINI_ARRAYSZ */

1191                if (ofl->ofl_osinitarray)       /* DT_INIT_ARRAY */
```

```
1192                    cnt += 2;               /*    DT_INIT_ARRAYSZ */

1194            if (ofl->ofl_ospreinitarray)    /* DT_PREINIT_ARRAY & */
1195                    cnt += 2;               /*    DT_PREINIT_ARRAYSZ */

1197            /*
1198             * If we have plt's reserve a DT_PLTRELSZ, DT_PLTREL and
1199             * DT_JMPREL.
1200             */
1201            if (ofl->ofl_pltcnt)
1202                    cnt += 3;

1204            /*
1205             * If plt padding is needed (Sparcv9).
1206             */
1207            if (ofl->ofl_pltpad)
1208                    cnt += 2;               /* DT_PLTPAD & DT_PLTPADSZ */

1210            /*
1211             * If we have any relocations reserve a DT_REL, DT_RELSZ and
1212             * DT_RELENT entry.
1213             */
1214            if (ofl->ofl_relocsz)
1215                    cnt += 3;

1217            /*
1218             * If a syminfo section is required create DT_SYMINFO,
1219             * DT_SYMINSZ, and DT_SYMINENT entries.
1220             */
1221            if (flags & FLG_OF_SYMINFO)
1222                    cnt += 3;

1224            /*
1225             * If there are any partially initialized sections allocate
1226             * DT_MOVETAB, DT_MOVESZ and DT_MOVEENT.
1227             */
1228            if (ofl->ofl_osmove)
1229                    cnt += 3;

1231            /*
1232             * Allocate one DT_REGISTER entry for every register symbol.
1233             */
1234            cnt += ofl->ofl_regsymcnt;

1236            /*
1237             * Reserve a entry for each '-zrtldinfo=...' specified
1238             * on the command line.
1239             */
1240            for (APLIST_TRAVERSE(ofl->ofl_rtldinfo, idx, sdp))
1241                    cnt++;

1243            /*
1244             * The following entry should only be placed in a segment that
1245             * is writable.
1246             */
1247            if (((sgp = osp->os_sgdesc) != NULL) &&
1248                (sgp->sg_phdr.p_flags & PF_W) && ofl->ofl_osinterp)
1249                    cnt++;          /* DT_DEBUG */

1251            /*
1252             * Capabilities require a .dynamic entry for the .SUNW_cap
1253             * section.
1254             */
1255            if (ofl->ofl_oscap)
1256                    cnt++;                  /* DT_SUNW_CAP */
```



```
1258            /*
1259             * Symbol capabilities require a .dynamic entry for the
1260             * .SUNW_capinfo section.
1261             */
1262            if (ofl->ofl_oscapinfo)
1263                    cnt++;                  /* DT_SUNW_CAPINFO */

1265            /*
1266             * Capabilities chain information requires a .SUNW_capchain
1267             * entry (DT_SUNW_CAPCHAIN), entry size (DT_SUNW_CAPCHAINENT),
1268             * and total size (DT_SUNW_CAPCHAINSZ).
1269             */
1270            if (ofl->ofl_oscapchain)
1271                    cnt += 3;

1273            if (flags & FLG_OF_SYMBOLIC)
1274                    cnt++;                  /* DT_SYMBOLIC */
1275    }

1277    /*
1278     * Account for Architecture dependent .dynamic entries, and defaults.
1279     */
1280    (*ld_targ.t_mr.mr_mach_make_dynamic)(ofl, &cnt);

1282    /*
1283     * DT_FLAGS, DT_FLAGS_1, DT_SUNW_STRPAD, and DT_NULL. Also,
1284     * allow room for the unused extra DT_NULLs. These are included
1285     * to allow an ELF editor room to add items later.
1286     */
1287    cnt += 4 + DYNAMIC_EXTRA_ELTS;

1289    /*
1290     * DT_SUNW_LDMACH. Used to hold the ELF machine code of the
1291     * linker that produced the output object. This information
1292     * allows us to determine whether a given object was linked
1293     * natively, or by a linker running on a different type of
1294     * system. This information can be valuable if one suspects
1295     * that a problem might be due to alignment or byte order issues.
1296     */
1297    cnt++;

1299    /*
1300     * Determine the size of the section from the number of entries.
1301     */
1302    size = cnt * (size_t)shdr->sh_entsize;

1304    shdr->sh_size = (Xword)size;
1305    data->d_size = size;

1307    /*
1308     * There are several tags that are specific to the Solaris osabi
1309     * range which we unconditionally put into any dynamic section
1310     * we create (e.g. DT_SUNW_STRPAD or DT_SUNW_LDMACH). As such,
1311     * any Solaris object with a dynamic section should be tagged as
1312     * ELFOSABI_SOLARIS.
1313     */
1314    ofl->ofl_flags |= FLG_OF_OSABI;

1316    return ((uintptr_t)ofl->ofl_osdynamic);
1317 }

1319 /*
1320  * Build the GOT section and its associated relocation entries.
1321  */
1322 uintptr_t
1323 ld_make_got(Ofl_desc *ofl)
```

```
{
        Elf_Data        *data;
        Shdr    *shdr;
        Is_desc *isec;
        size_t  size = (size_t)ofl->ofl_gotcnt * ld_targ.t_m.m_got_entsize;
        size_t  rsize = (size_t)ofl->ofl_relocgotsz;

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

        data->d_size = size;

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

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

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

        return (1);
}

/*
 * Build an interpreter section.
 */
static uintptr_t
make_interp(Ofl_desc *ofl)
{
        Shdr            *shdr;
        Elf_Data        *data;
        Is_desc         *isec;
        const char      *iname = ofl->ofl_interp;
        size_t          size;

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

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

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

        size = strlen(iname) + 1;
```



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

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

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

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

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

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

/*
 * Determine whether a symbol capability is redundant because the object
 * capabilities are more restrictive.
 */
inline static int
is_cap_redundant(Objcapset *ocapset, Objcapset *scapset)
{
        Alist           *oalp, *salp;
        elfcap_mask_t   omsk, smsk;

        /*
         * Inspect any platform capabilities.  If the object defines platform
         * capabilities, then the object will only be loaded for those
         * platforms.  A symbol capability set that doesn't define the same
         * platforms is redundant, and a symbol capability that does not provide
         * at least one platform name that matches a platform name in the object
         * capabilities will never execute (as the object wouldn't have been
         * loaded).
         */
        oalp = ocapset->oc_plat.cl_val;
        salp = scapset->oc_plat.cl_val;
        if (oalp && ((salp == NULL) || cap_names_match(oalp, salp)))
                return (1);

        /*
         * If the symbol capability set defines platforms, and the object
         * doesn't, then the symbol set is more restrictive.
         */
        if (salp && (oalp == NULL))
```

```
1456                 return (0);

1458         /*
1459          * Next, inspect any machine name capabilities.  If the object defines
1460          * machine name capabilities, then the object will only be loaded for
1461          * those machines.  A symbol capability set that doesn't define the same
1462          * machine names is redundant, and a symbol capability that does not
1463          * provide at least one machine name that matches a machine name in the
1464          * object capabilities will never execute (as the object wouldn't have
1465          * been loaded).
1466          */
1467         oalp = ocapset->oc_plat.cl_val;
1468         salp = scapset->oc_plat.cl_val;
1469         if (oalp && ((salp == NULL) || cap_names_match(oalp, salp)))
1470                 return (1);

1472         /*
1473          * If the symbol capability set defines machine names, and the object
1474          * doesn't, then the symbol set is more restrictive.
1475          */
1476         if (salp && (oalp == NULL))
1477                 return (0);

1479         /*
1480          * Next, inspect any hardware capabilities.  If the objects hardware
1481          * capabilities are greater than or equal to that of the symbols
1482          * capabilities, then the symbol capability set is redundant.  If the
1483          * symbols hardware capabilities are greater that the objects, then the
1484          * symbol set is more restrictive.
1485          *
1486          * Note that this is a somewhat arbitrary definition, as each capability
1487          * bit is independent of the others, and some of the higher order bits
1488          * could be considered to be less important than lower ones.  However,
1489          * this is the only reasonable non-subjective definition.
1490          */
1491         omsk = ocapset->oc_hw_2.cm_val;
1492         smsk = scapset->oc_hw_2.cm_val;
1493         if ((omsk > smsk) || (omsk && (omsk == smsk)))
1494                 return (1);
1495         if (omsk < smsk)
1496                 return (0);

1498         /*
1499          * Finally, inspect the remaining hardware capabilities.
1500          */
1501         omsk = ocapset->oc_hw_1.cm_val;
1502         smsk = scapset->oc_hw_1.cm_val;
1503         if ((omsk > smsk) || (omsk && (omsk == smsk)))
1504                 return (1);

1506         return (0);
1507 }

1509 /*
1510  * Capabilities values might have been assigned excluded values.  These
1511  * excluded values should be removed before calculating any capabilities
1512  * sections size.
1513  */
1514 static void
1515 capmask_value(Lm_list *lml, Word type, Capmask *capmask, int *title)
1516 {
1517         /*
1518          * First determine whether any bits should be excluded.
1519          */
1520         if ((capmask->cm_val & capmask->cm_exc) == 0)
1521                 return;
```



```
1523         DBG_CALL(Dbg_cap_post_title(lml, title));

1525         DBG_CALL(Dbg_cap_val_entry(lml, DBG_STATE_CURRENT, type,
1526             capmask->cm_val, ld_targ.t_m.m_mach));
1527         DBG_CALL(Dbg_cap_val_entry(lml, DBG_STATE_EXCLUDE, type,
1528             capmask->cm_exc, ld_targ.t_m.m_mach));

1530         capmask->cm_val &= ~capmask->cm_exc;

1532         DBG_CALL(Dbg_cap_val_entry(lml, DBG_STATE_RESOLVED, type,
1533             capmask->cm_val, ld_targ.t_m.m_mach));
1534 }

1536 static void
1537 capstr_value(Lm_list *lml, Word type, Caplist *caplist, int *title)
1538 {
1539         Aliste  idx1, idx2;
1540         char    *estr;
1541         Capstr  *capstr;
1542         Boolean found = FALSE;

1544         /*
1545          * First determine whether any strings should be excluded.
1546          */
1547         for (APLIST_TRAVERSE(caplist->cl_exc, idx1, estr)) {
1548                 for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1549                         if (strcmp(estr, capstr->cs_str) == 0) {
1550                                 found = TRUE;
1551                                 break;
1552                         }
1553                 }
1554         }

1556         if (found == FALSE)
1557                 return;

1559         /*
1560          * Traverse the current strings, then delete the excluded strings,
1561          * and finally display the resolved strings.
1562          */
1563         if (DBG_ENABLED) {
1564                 Dbg_cap_post_title(lml, title);
1565                 for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1566                         Dbg_cap_ptr_entry(lml, DBG_STATE_CURRENT, type,
1567                             capstr->cs_str);
1568                 }
1569         }
1570         for (APLIST_TRAVERSE(caplist->cl_exc, idx1, estr)) {
1571                 for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1572                         if (strcmp(estr, capstr->cs_str) == 0) {
1573                                 DBG_CALL(Dbg_cap_ptr_entry(lml,
1574                                     DBG_STATE_EXCLUDE, type, capstr->cs_str));
1575                                 alist_delete(caplist->cl_val, &idx2);
1576                                 break;
1577                         }
1578                 }
1579         }
1580         if (DBG_ENABLED) {
1581                 for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1582                         Dbg_cap_ptr_entry(lml, DBG_STATE_RESOLVED, type,
1583                             capstr->cs_str);
1584                 }
1585         }
1586 }
```

```
1588 /*
1589  * Build a capabilities section.
1590  */
1591 #define CAP_UPDATE(cap, capndx, tag, val)       \
1592         cap->c_tag = tag; \
1593         cap->c_un.c_val = val; \
1594         cap++, capndx++;

1596 static uintptr_t
1597 make_cap(Ofl_desc *ofl, Word shtype, const char *shname, int ident)
1598 {
1599         Shdr            *shdr;
1600         Elf_Data        *data;
1601         Is_desc         *isec;
1602         Cap             *cap;
1603         size_t          size = 0;
1604         Word            capndx = 0;
1605         Str_tbl         *strtbl;
1606         Objcapset       *ocapset = &ofl->ofl_ocapset;
1607         Aliste          idx1;
1608         Capstr          *capstr;
1609         int             title = 0;

1611         /*
1612          * Determine which string table to use for any CA_SUNW_MACH,
1613          * CA_SUNW_PLAT, or CA_SUNW_ID strings.
1614          */
1615         if (OFL_IS_STATIC_OBJ(ofl))
1616                 strtbl = ofl->ofl_strtab;
1617         else
1618                 strtbl = ofl->ofl_dynstrtab;

1620         /*
1621          * If symbol capabilities have been requested, but none have been
1622          * created, warn the user.  This scenario can occur if none of the
1623          * input relocatable objects defined any object capabilities.
1624          */
1625         if ((ofl->ofl_flags & FLG_OF_OTOSCAP) && (ofl->ofl_capsymcnt == 0))
1626                 ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_CAP_NOSYMSFOUND));

1628         /*
1629          * If symbol capabilities have been collected, but no symbols are left
1630          * referencing these capabilities, promote the capability groups back
1631          * to an object capability definition.
1632          */
1633         if ((ofl->ofl_flags & FLG_OF_OTOSCAP) && ofl->ofl_capsymcnt &&
1634             (ofl->ofl_capfamilies == NULL)) {
1635                 ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_CAP_NOSYMSFOUND));
1636                 ld_cap_move_symtoobj(ofl);
1637                 ofl->ofl_capsymcnt = 0;
1638                 ofl->ofl_capgroups = NULL;
1639                 ofl->ofl_flags &= ~FLG_OF_OTOSCAP;
1640         }

1642         /*
1643          * Remove any excluded capabilities.
1644          */
1645         capstr_value(ofl->ofl_lml, CA_SUNW_PLAT, &ocapset->oc_plat, &title);
1646         capstr_value(ofl->ofl_lml, CA_SUNW_MACH, &ocapset->oc_mach, &title);
1647         capmask_value(ofl->ofl_lml, CA_SUNW_HW_2, &ocapset->oc_hw_2, &title);
1648         capmask_value(ofl->ofl_lml, CA_SUNW_HW_1, &ocapset->oc_hw_1, &title);
1649         capmask_value(ofl->ofl_lml, CA_SUNW_SF_1, &ocapset->oc_sf_1, &title);

1651         /*
1652          * Determine how many entries are required for any object capabilities.
1653          */
```



```
1654         size += alist_nitems(ocapset->oc_plat.cl_val);
1655         size += alist_nitems(ocapset->oc_mach.cl_val);
1656         if (ocapset->oc_hw_2.cm_val)
1657                 size++;
1658         if (ocapset->oc_hw_1.cm_val)
1659                 size++;
1660         if (ocapset->oc_sf_1.cm_val)
1661                 size++;

1663         /*
1664          * Only identify a capabilities group if the group has content.  If a
1665          * capabilities identifier exists, and no other capabilities have been
1666          * supplied, remove the identifier.  This scenario could exist if a
1667          * user mistakenly defined a lone identifier, or if an identified group
1668          * was overridden so as to clear the existing capabilities and the
1669          * identifier was not also cleared.
1670          */
1671         if (ocapset->oc_id.cs_str) {
1672                 if (size)
1673                         size++;
1674                 else
1675                         ocapset->oc_id.cs_str = NULL;
1676         }
1677         if (size)
1678                 size++;                 /* Add CA_SUNW_NULL */

1680         /*
1681          * Determine how many entries are required for any symbol capabilities.
1682          */
1683         if (ofl->ofl_capsymcnt) {
1684                 /*
1685                  * If there are no object capabilities, a CA_SUNW_NULL entry
1686                  * is required before any symbol capabilities.
1687                  */
1688                 if (size == 0)
1689                         size++;
1690                 size += ofl->ofl_capsymcnt;
1691         }

1693         if (size == 0)
1694                 return (NULL);

1696         if (new_section(ofl, shtype, shname, size, &isec,
1697             &shdr, &data) == S_ERROR)
1698                 return (S_ERROR);

1700         if ((data->d_buf = libld_malloc(shdr->sh_size)) == NULL)
1701                 return (S_ERROR);

1703         cap = (Cap *)data->d_buf;

1705         /*
1706          * Fill in any object capabilities.  If there is an identifier, then the
1707          * identifier comes first.  The remaining items follow in precedence
1708          * order, although the order isn't important for runtime verification.
1709          */
1710         if (ocapset->oc_id.cs_str) {
1711                 ofl->ofl_flags |= FLG_OF_CAPSTRS;
1712                 if (st_insert(strtbl, ocapset->oc_id.cs_str) == -1)
1713                         return (S_ERROR);
1714                 ocapset->oc_id.cs_ndx = capndx;
1715                 CAP_UPDATE(cap, capndx, CA_SUNW_ID, 0);
1716         }
1717         if (ocapset->oc_plat.cl_val) {
1718                 ofl->ofl_flags |= (FLG_OF_PTCAP | FLG_OF_CAPSTRS);
```

```
1720                /*
1721                 * Insert any platform name strings in the appropriate string
1722                 * table.  The capability value can't be filled in yet, as the
1723                 * final offset of the strings isn't known until later.
1724                 */
1725                for (ALIST_TRAVERSE(ocapset->oc_plat.cl_val, idx1, capstr)) {
1726                        if (st_insert(strtbl, capstr->cs_str) == -1)
1727                                return (S_ERROR);
1728                        capstr->cs_ndx = capndx;
1729                        CAP_UPDATE(cap, capndx, CA_SUNW_PLAT, 0);
1730                }
1731        }
1732        if (ocapset->oc_mach.cl_val) {
1733                ofl->ofl_flags |= (FLG_OF_PTCAP | FLG_OF_CAPSTRS);

1735                /*
1736                 * Insert the machine name strings in the appropriate string
1737                 * table.  The capability value can't be filled in yet, as the
1738                 * final offset of the strings isn't known until later.
1739                 */
1740                for (ALIST_TRAVERSE(ocapset->oc_mach.cl_val, idx1, capstr)) {
1741                        if (st_insert(strtbl, capstr->cs_str) == -1)
1742                                return (S_ERROR);
1743                        capstr->cs_ndx = capndx;
1744                        CAP_UPDATE(cap, capndx, CA_SUNW_MACH, 0);
1745                }
1746        }
1747        if (ocapset->oc_hw_2.cm_val) {
1748                ofl->ofl_flags |= FLG_OF_PTCAP;
1749                CAP_UPDATE(cap, capndx, CA_SUNW_HW_2, ocapset->oc_hw_2.cm_val);
1750        }
1751        if (ocapset->oc_hw_1.cm_val) {
1752                ofl->ofl_flags |= FLG_OF_PTCAP;
1753                CAP_UPDATE(cap, capndx, CA_SUNW_HW_1, ocapset->oc_hw_1.cm_val);
1754        }
1755        if (ocapset->oc_sf_1.cm_val) {
1756                ofl->ofl_flags |= FLG_OF_PTCAP;
1757                CAP_UPDATE(cap, capndx, CA_SUNW_SF_1, ocapset->oc_sf_1.cm_val);
1758        }
1759        CAP_UPDATE(cap, capndx, CA_SUNW_NULL, 0);

1761        /*
1762         * Fill in any symbol capabilities.
1763         */
1764        if (ofl->ofl_capgroups) {
1765                Cap_group       *cgp;

1767                for (APLIST_TRAVERSE(ofl->ofl_capgroups, idx1, cgp)) {
1768                        Objcapset       *scapset = &cgp->cg_set;
1769                        Aliste          idx2;
1770                        Is_desc         *isp;

1772                        cgp->cg_ndx = capndx;

1774                        if (scapset->oc_id.cs_str) {
1775                                ofl->ofl_flags |= FLG_OF_CAPSTRS;
1776                                /*
1777                                 * Insert the identifier string in the
1778                                 * appropriate string table.  The capability
1779                                 * value can't be filled in yet, as the final
1780                                 * offset of the string isn't known until later.
1781                                 */
1782                                if (st_insert(strtbl,
1783                                    scapset->oc_id.cs_str) == -1)
1784                                        return (S_ERROR);
1785                                scapset->oc_id.cs_ndx = capndx;
```



```
1786                                CAP_UPDATE(cap, capndx, CA_SUNW_ID, 0);
1787                        }

1789                        if (scapset->oc_plat.cl_val) {
1790                                ofl->ofl_flags |= FLG_OF_CAPSTRS;

1792                                /*
1793                                 * Insert the platform name string in the
1794                                 * appropriate string table.  The capability
1795                                 * value can't be filled in yet, as the final
1796                                 * offset of the string isn't known until later.
1797                                 */
1798                                for (ALIST_TRAVERSE(scapset->oc_plat.cl_val,
1799                                    idx2, capstr)) {
1800                                        if (st_insert(strtbl,
1801                                            capstr->cs_str) == -1)
1802                                                return (S_ERROR);
1803                                        capstr->cs_ndx = capndx;
1804                                        CAP_UPDATE(cap, capndx,
1805                                            CA_SUNW_PLAT, 0);
1806                                }
1807                        }
1808                        if (scapset->oc_mach.cl_val) {
1809                                ofl->ofl_flags |= FLG_OF_CAPSTRS;

1811                                /*
1812                                 * Insert the machine name string in the
1813                                 * appropriate string table.  The capability
1814                                 * value can't be filled in yet, as the final
1815                                 * offset of the string isn't known until later.
1816                                 */
1817                                for (ALIST_TRAVERSE(scapset->oc_mach.cl_val,
1818                                    idx2, capstr)) {
1819                                        if (st_insert(strtbl,
1820                                            capstr->cs_str) == -1)
1821                                                return (S_ERROR);
1822                                        capstr->cs_ndx = capndx;
1823                                        CAP_UPDATE(cap, capndx,
1824                                            CA_SUNW_MACH, 0);
1825                                }
1826                        }
1827                        if (scapset->oc_hw_2.cm_val) {
1828                                CAP_UPDATE(cap, capndx, CA_SUNW_HW_2,
1829                                    scapset->oc_hw_2.cm_val);
1830                        }
1831                        if (scapset->oc_hw_1.cm_val) {
1832                                CAP_UPDATE(cap, capndx, CA_SUNW_HW_1,
1833                                    scapset->oc_hw_1.cm_val);
1834                        }
1835                        if (scapset->oc_sf_1.cm_val) {
1836                                CAP_UPDATE(cap, capndx, CA_SUNW_SF_1,
1837                                    scapset->oc_sf_1.cm_val);
1838                        }
1839                        CAP_UPDATE(cap, capndx, CA_SUNW_NULL, 0);

1841                        /*
1842                         * If any object capabilities are available, determine
1843                         * whether these symbol capabilities are less
1844                         * restrictive, and hence redundant.
1845                         */
1846                        if (((ofl->ofl_flags & FLG_OF_PTCAP) == 0) ||
1847                            (is_cap_redundant(ocapset, scapset) == 0))
1848                                continue;

1850                        /*
1851                         * Indicate any files that provide redundant symbol
```

```
1852                         * capabilities.
1853                         */
1854                        for (APLIST_TRAVERSE(cgp->cg_secs, idx2, isp)) {
1855                                ld_eprintf(ofl, ERR_WARNING,
1856                                    MSG_INTL(MSG_CAP_REDUNDANT),
1857                                    isp->is_file->ifl_name,
1858                                    EC_WORD(isp->is_scnndx), isp->is_name);
1859                        }
1860                }
1861        }
1863        /*
1864         * If capabilities strings are required, the sh_info field of the
1865         * section header will be set to the associated string table.
1866         */
1867        if (ofl->ofl_flags & FLG_OF_CAPSTRS)
1868                shdr->sh_flags |= SHF_INFO_LINK;
1870        /*
1871         * Place these capabilities in the output file.
1872         */
1873        if ((ofl->ofl_oscap = ld_place_section(ofl, isec,
1874            NULL, ident, NULL)) == (Os_desc *)S_ERROR)
1875                return (S_ERROR);
1877        /*
1878         * If symbol capabilities are required, then a .SUNW_capinfo section is
1879         * also created.  This table will eventually be sized to match the
1880         * associated symbol table.
1881         */
1882        if (ofl->ofl_capfamilies) {
1883                if ((ofl->ofl_oscapinfo = make_sym_sec(ofl,
1884                    MSG_ORIG(MSG_SCN_SUNWCAPINFO), SHT_SUNW_capinfo,
1885                    ld_targ.t_id.id_capinfo)) == (Os_desc *)S_ERROR)
1886                        return (S_ERROR);
1888                /*
1889                 * If we're generating a dynamic object, capabilities family
1890                 * members are maintained in a .SUNW_capchain section.
1891                 */
1892                if (ofl->ofl_capchaincnt &&
1893                    ((ofl->ofl_flags & FLG_OF_RELOBJ) == 0)) {
1894                        if (new_section(ofl, SHT_SUNW_capchain,
1895                            MSG_ORIG(MSG_SCN_SUNWCAPCHAIN),
1896                            ofl->ofl_capchaincnt, &isec, &shdr,
1897                            &data) == S_ERROR)
1898                                return (S_ERROR);
1900                        ofl->ofl_oscapchain = ld_place_section(ofl, isec,
1901                            NULL, ld_targ.t_id.id_capchain, NULL);
1902                        if (ofl->ofl_oscapchain == (Os_desc *)S_ERROR)
1903                                return (S_ERROR);
1905                }
1906        }
1907        return (1);
1908 }
1909 #undef  CAP_UPDATE
1911 /*
1912  * Build the PLT section and its associated relocation entries.
1913  */
1914 static uintptr_t
1915 make_plt(Ofl_desc *ofl)
1916 {
1917        Shdr            *shdr;
```



```
1918        Elf_Data        *data;
1919        Is_desc         *isec;
1920        size_t          size = ld_targ.t_m.m_plt_reservsz +
1921            (((size_t)ofl->ofl_pltcnt + (size_t)ofl->ofl_pltpad) *
1922            ld_targ.t_m.m_plt_entsize);
1923        size_t          rsize = (size_t)ofl->ofl_relocpltsz;
1925        /*
1926         * On sparc, account for the NOP at the end of the plt.
1927         */
1928        if (ld_targ.t_m.m_mach == LD_TARG_BYCLASS(EM_SPARC, EM_SPARCV9))
1929                size += sizeof (Word);
1931        if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_PLT), 0,
1932            &isec, &shdr, &data) == S_ERROR)
1933                return (S_ERROR);
1935        data->d_size = size;
1936        data->d_align = ld_targ.t_m.m_plt_align;
1938        shdr->sh_flags = ld_targ.t_m.m_plt_shf_flags;
1939        shdr->sh_size = (Xword)size;
1940        shdr->sh_addralign = ld_targ.t_m.m_plt_align;
1941        shdr->sh_entsize = ld_targ.t_m.m_plt_entsize;
1943        ofl->ofl_osplt = ld_place_section(ofl, isec, NULL,
1944            ld_targ.t_id.id_plt, NULL);
1945        if (ofl->ofl_osplt == (Os_desc *)S_ERROR)
1946                return (S_ERROR);
1948        ofl->ofl_osplt->os_szoutrels = (Xword)rsize;
1950        return (1);
1951 }
1953 /*
1954  * Make the hash table.  Only built for dynamic executables and shared
1955  * libraries, and provides hashed lookup into the global symbol table
1956  * (.dynsym) for the run-time linker to resolve symbol lookups.
1957  */
1958 static uintptr_t
1959 make_hash(Ofl_desc *ofl)
1960 {
1961        Shdr            *shdr;
1962        Elf_Data        *data;
1963        Is_desc         *isec;
1964        size_t          size;
1965        Word            nsyms = ofl->ofl_globcnt;
1966        size_t          cnt;
1968        /*
1969         * Allocate section header structures. We set entcnt to 0
1970         * because it's going to change after we place this section.
1971         */
1972        if (new_section(ofl, SHT_HASH, MSG_ORIG(MSG_SCN_HASH), 0,
1973            &isec, &shdr, &data) == S_ERROR)
1974                return (S_ERROR);
1976        /*
1977         * Place the section first since it will affect the local symbol
1978         * count.
1979         */
1980        ofl->ofl_oshash =
1981            ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_hash, NULL);
1982        if (ofl->ofl_oshash == (Os_desc *)S_ERROR)
1983                return (S_ERROR);
```

```
1985         /*
1986          * Calculate the number of output hash buckets.
1987          */
1988         ofl->ofl_hashbkts = findprime(nsyms);

1990         /*
1991          * The size of the hash table is determined by
1992          *
1993          *      i.      the initial nbucket and nchain entries (2)
1994          *      ii.     the number of buckets (calculated above)
1995          *      iii.    the number of chains (this is based on the number of
1996          *              symbols in the .dynsym array).
1997          */
1998         cnt = 2 + ofl->ofl_hashbkts + DYNSYM_ALL_CNT(ofl);
1999         size = cnt * shdr->sh_entsize;

2001         /*
2002          * Finalize the section header and data buffer initialization.
2003          */
2004         if ((data->d_buf = libld_calloc(size, 1)) == NULL)
2005                 return (S_ERROR);
2006         data->d_size = size;
2007         shdr->sh_size = (Xword)size;

2009         return (1);
2010 }

2012 /*
2013  * Generate the standard symbol table.  Contains all locals and globals,
2014  * and resides in a non-allocatable section (ie. it can be stripped).
2015  */
2016 static uintptr_t
2017 make_symtab(Ofl_desc *ofl)
2018 {
2019         Shdr            *shdr;
2020         Elf_Data        *data;
2021         Is_desc         *isec;
2022         Is_desc         *xisec = 0;
2023         size_t          size;
2024         Word            symcnt;

2026         /*
2027          * Create the section headers. Note that we supply an ent_cnt
2028          * of 0. We won't know the count until the section has been placed.
2029          */
2030         if (new_section(ofl, SHT_SYMTAB, MSG_ORIG(MSG_SCN_SYMTAB), 0,
2031             &isec, &shdr, &data) == S_ERROR)
2032                 return (S_ERROR);

2034         /*
2035          * Place the section first since it will affect the local symbol
2036          * count.
2037          */
2038         if ((ofl->ofl_ossymtab = ld_place_section(ofl, isec, NULL,
2039             ld_targ.t_id.id_symtab, NULL)) == (Os_desc *)S_ERROR)
2040                 return (S_ERROR);

2042         /*
2043          * At this point we've created all but the 'shstrtab' section.
2044          * Determine if we have to use 'Extended Sections'.  If so - then
2045          * also create a SHT_SYMTAB_SHNDX section.
2046          */
2047         if ((ofl->ofl_shdrcnt + 1) >= SHN_LORESERVE) {
2048                 Shdr            *xshdr;
2049                 Elf_Data        *xdata;
```



```
2051                 if (new_section(ofl, SHT_SYMTAB_SHNDX,
2052                     MSG_ORIG(MSG_SCN_SYMTAB_SHNDX), 0, &xisec,
2053                     &xshdr, &xdata) == S_ERROR)
2054                         return (S_ERROR);

2056                 if ((ofl->ofl_ossymshndx = ld_place_section(ofl, xisec, NULL,
2057                     ld_targ.t_id.id_symtab_ndx, NULL)) == (Os_desc *)S_ERROR)
2058                         return (S_ERROR);
2059         }

2061         /*
2062          * Calculated number of symbols, which need to be augmented by
2063          * the (yet to be created) .shstrtab entry.
2064          */
2065         symcnt = (size_t)(1 + SYMTAB_ALL_CNT(ofl));
2066         size = symcnt * shdr->sh_entsize;

2068         /*
2069          * Finalize the section header and data buffer initialization.
2070          */
2071         data->d_size = size;
2072         shdr->sh_size = (Xword)size;

2074         /*
2075          * If we created a SHT_SYMTAB_SHNDX - then set it's sizes too.
2076          */
2077         if (xisec) {
2078                 size_t  xsize = symcnt * sizeof (Word);

2080                 xisec->is_indata->d_size = xsize;
2081                 xisec->is_shdr->sh_size = (Xword)xsize;
2082         }

2084         return (1);
2085 }

2087 /*
2088  * Build a dynamic symbol table. These tables reside in the text
2089  * segment of a dynamic executable or shared library.
2090  *
2091  *      .SUNW_ldynsym contains local function symbols
2092  *      .dynsym contains only globals symbols
2093  *
2094  * The two tables are created adjacent to each other, with .SUNW_ldynsym
2095  * coming first.
2096  */
2097 static uintptr_t
2098 make_dynsym(Ofl_desc *ofl)
2099 {
2100         Shdr            *shdr, *lshdr;
2101         Elf_Data        *data, *ldata;
2102         Is_desc         *isec, *lisec;
2103         size_t          size;
2104         Xword           cnt;
2105         int             allow_ldynsym;

2107         /*
2108          * Unless explicitly disabled, always produce a .SUNW_ldynsym section
2109          * when it is allowed by the file type, even if the resulting
2110          * table only ends up with a single STT_FILE in it. There are
2111          * two reasons: (1) It causes the generation of the DT_SUNW_SYMTAB
2112          * entry in the .dynamic section, which is something we would
2113          * like to encourage, and (2) Without it, we cannot generate
2114          * the associated .SUNW_dyn[sym|tls]sort sections, which are of
2115          * value to DTrace.
```

```
2116          *
2117          * In practice, it is extremely rare for an object not to have
2118          * local symbols for .SUNW_ldynsym, so 99% of the time, we'd be
2119          * doing it anyway.
2120          */
2121         allow_ldynsym = OFL_ALLOW_LDYNSYM(ofl);

2123         /*
2124          * Create the section headers. Note that we supply an ent_cnt
2125          * of 0. We won't know the count until the section has been placed.
2126          */
2127         if (allow_ldynsym && new_section(ofl, SHT_SUNW_LDYNSYM,
2128             MSG_ORIG(MSG_SCN_LDYNSYM), 0, &lisec, &lshdr, &ldata) == S_ERROR)
2129                 return (S_ERROR);

2131         if (new_section(ofl, SHT_DYNSYM, MSG_ORIG(MSG_SCN_DYNSYM), 0,
2132             &isec, &shdr, &data) == S_ERROR)
2133                 return (S_ERROR);

2135         /*
2136          * Place the section(s) first since it will affect the local symbol
2137          * count.
2138          */
2139         if (allow_ldynsym &&
2140             ((ofl->ofl_osldynsym = ld_place_section(ofl, lisec, NULL,
2141             ld_targ.t_id.id_ldynsym, NULL)) == (Os_desc *)S_ERROR))
2142                 return (S_ERROR);
2143         ofl->ofl_osdynsym =
2144             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_dynsym, NULL);
2145         if (ofl->ofl_osdynsym == (Os_desc *)S_ERROR)
2146                 return (S_ERROR);

2148         cnt = DYNSYM_ALL_CNT(ofl);
2149         size = (size_t)cnt * shdr->sh_entsize;

2151         /*
2152          * Finalize the section header and data buffer initialization.
2153          */
2154         data->d_size = size;
2155         shdr->sh_size = (Xword)size;

2157         /*
2158          * An ldynsym contains local function symbols. It is not
2159          * used for linking, but if present, serves to allow better
2160          * stack traces to be generated in contexts where the symtab
2161          * is not available. (dladdr(), or stripped executable/library files).
2162          */
2163         if (allow_ldynsym) {
2164                 cnt = 1 + ofl->ofl_dynlocscnt + ofl->ofl_dynscopecnt;
2165                 size = (size_t)cnt * shdr->sh_entsize;

2167                 ldata->d_size = size;
2168                 lshdr->sh_size = (Xword)size;
2169         }

2171         return (1);
2172 }

2174 /*
2175  * Build .SUNW_dynsymsort and/or .SUNW_dyntlssort sections. These are
2176  * index sections for the .SUNW_ldynsym/.dynsym pair that present data
2177  * and function symbols sorted by address.
2178  */
2179 static uintptr_t
2180 make_dynsort(Ofl_desc *ofl)
2181 {
```



```
2182         Shdr            *shdr;
2183         Elf_Data        *data;
2184         Is_desc         *isec;

2186         /* Only do it if the .SUNW_ldynsym section is present */
2187         if (!OFL_ALLOW_LDYNSYM(ofl))
2188                 return (1);

2190         /* .SUNW_dynsymsort */
2191         if (ofl->ofl_dynsymsortcnt > 0) {
2192                 if (new_section(ofl, SHT_SUNW_symsort,
2193                     MSG_ORIG(MSG_SCN_DYNSYMSORT), ofl->ofl_dynsymsortcnt,
2194                     &isec, &shdr, &data) == S_ERROR)
2195                 return (S_ERROR);

2197                 if ((ofl->ofl_osdynsymsort = ld_place_section(ofl, isec, NULL,
2198                     ld_targ.t_id.id_dynsort, NULL)) == (Os_desc *)S_ERROR)
2199                         return (S_ERROR);
2200         }

2202         /* .SUNW_dyntlssort */
2203         if (ofl->ofl_dyntlssortcnt > 0) {
2204                 if (new_section(ofl, SHT_SUNW_tlssort,
2205                     MSG_ORIG(MSG_SCN_DYNTLSSORT),
2206                     ofl->ofl_dyntlssortcnt, &isec, &shdr, &data) == S_ERROR)
2207                 return (S_ERROR);

2209                 if ((ofl->ofl_osdyntlssort = ld_place_section(ofl, isec, NULL,
2210                     ld_targ.t_id.id_dynsort, NULL)) == (Os_desc *)S_ERROR)
2211                         return (S_ERROR);
2212         }

2214         return (1);
2215 }

2217 /*
2218  * Helper routine for make_dynsym_shndx. Builds a
2219  * a SHT_SYMTAB_SHNDX for .dynsym or .SUNW_ldynsym, without knowing
2220  * which one it is.
2221  */
2222 static uintptr_t
2223 make_dyn_shndx(Ofl_desc *ofl, const char *shname, Os_desc *symtab,
2224     Os_desc **ret_os)
2225 {
2226         Is_desc         *isec;
2227         Is_desc         *dynsymisp;
2228         Shdr            *shdr, *dynshdr;
2229         Elf_Data        *data;

2231         dynsymisp = ld_os_first_isdesc(symtab);
2232         dynshdr = dynsymisp->is_shdr;

2234         if (new_section(ofl, SHT_SYMTAB_SHNDX, shname,
2235             (dynshdr->sh_size / dynshdr->sh_entsize),
2236             &isec, &shdr, &data) == S_ERROR)
2237                 return (S_ERROR);

2239         if ((*ret_os = ld_place_section(ofl, isec, NULL,
2240             ld_targ.t_id.id_dynsym_ndx, NULL)) == (Os_desc *)S_ERROR)
2241                 return (S_ERROR);

2243         assert(*ret_os);

2245         return (1);
2246 }
```

```
2248 /*
2249  * Build a SHT_SYMTAB_SHNDX for the .dynsym, and .SUNW_ldynsym
2250  */
2251 static uintptr_t
2252 make_dynsym_shndx(Ofl_desc *ofl)
2253 {
2254         /*
2255          * If there is a .SUNW_ldynsym, generate a section for its extended
2256          * index section as well.
2257          */
2258         if (OFL_ALLOW_LDYNSYM(ofl)) {
2259                 if (make_dyn_shndx(ofl, MSG_ORIG(MSG_SCN_LDYNSYM_SHNDX),
2260                     ofl->ofl_osldynsym, &ofl->ofl_osldynshndx) == S_ERROR)
2261                         return (S_ERROR);
2262         }

2264         /* The Generate a section for the dynsym */
2265         if (make_dyn_shndx(ofl, MSG_ORIG(MSG_SCN_DYNSYM_SHNDX),
2266             ofl->ofl_osdynsym, &ofl->ofl_osdynshndx) == S_ERROR)
2267                 return (S_ERROR);

2269         return (1);
2270 }


2273 /*
2274  * Build a string table for the section headers.
2275  */
2276 static uintptr_t
2277 make_shstrtab(Ofl_desc *ofl)
2278 {
2279         Shdr            *shdr;
2280         Elf_Data        *data;
2281         Is_desc         *isec;
2282         size_t          size;

2284         if (new_section(ofl, SHT_STRTAB, MSG_ORIG(MSG_SCN_SHSTRTAB),
2285             0, &isec, &shdr, &data) == S_ERROR)
2286                 return (S_ERROR);

2288         /*
2289          * Place the section first, as it may effect the number of section
2290          * headers to account for.
2291          */
2292         ofl->ofl_osshstrtab =
2293             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_note, NULL);
2294         if (ofl->ofl_osshstrtab == (Os_desc *)S_ERROR)
2295                 return (S_ERROR);

2297         size = st_getstrtab_sz(ofl->ofl_shdrsttab);
2298         assert(size > 0);

2300         data->d_size = size;
2301         shdr->sh_size = (Xword)size;

2303         return (1);
2304 }

2306 /*
2307  * Build a string section for the standard symbol table.
2308  */
2309 static uintptr_t
2310 make_strtab(Ofl_desc *ofl)
2311 {
2312         Shdr            *shdr;
2313         Elf_Data        *data;
```



```
2314         Is_desc         *isec;
2315         size_t          size;

2317         /*
2318          * This string table consists of all the global and local symbols.
2319          * Account for null bytes at end of the file name and the beginning
2320          * of section.
2321          */
2322         if (st_insert(ofl->ofl_strtab, ofl->ofl_name) == -1)
2323                 return (S_ERROR);

2325         size = st_getstrtab_sz(ofl->ofl_strtab);
2326         assert(size > 0);

2328         if (new_section(ofl, SHT_STRTAB, MSG_ORIG(MSG_SCN_STRTAB),
2329             0, &isec, &shdr, &data) == S_ERROR)
2330                 return (S_ERROR);

2332         /* Set the size of the data area */
2333         data->d_size = size;
2334         shdr->sh_size = (Xword)size;

2336         ofl->ofl_osstrtab =
2337             ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_strtab, NULL);
2338         return ((uintptr_t)ofl->ofl_osstrtab);
2339 }

2341 /*
2342  * Build a string table for the dynamic symbol table.
2343  */
2344 static uintptr_t
2345 make_dynstr(Ofl_desc *ofl)
2346 {
2347         Shdr            *shdr;
2348         Elf_Data        *data;
2349         Is_desc         *isec;
2350         size_t          size;

2352         /*
2353          * If producing a .SUNW_ldynsym, account for the initial STT_FILE
2354          * symbol that precedes the scope reduced global symbols.
2355          */
2356         if (OFL_ALLOW_LDYNSYM(ofl)) {
2357                 if (st_insert(ofl->ofl_dynstrtab, ofl->ofl_name) == -1)
2358                         return (S_ERROR);
2359                 ofl->ofl_dynscopecnt++;
2360         }

2362         /*
2363          * Account for any local, named register symbols.  These locals are
2364          * required for reference from DT_REGISTER .dynamic entries.
2365          */
2366         if (ofl->ofl_regsyms) {
2367                 int     ndx;

2369                 for (ndx = 0; ndx < ofl->ofl_regsymsno; ndx++) {
2370                         Sym_desc        *sdp;

2372                         if ((sdp = ofl->ofl_regsyms[ndx]) == NULL)
2373                                 continue;

2375                         if (!SYM_IS_HIDDEN(sdp) &&
2376                             (ELF_ST_BIND(sdp->sd_sym->st_info) != STB_LOCAL))
2377                                 continue;

2379                         if (sdp->sd_sym->st_name == NULL)
```

```
2380                                    continue;
2381
2382                            if (st_insert(ofl->ofl_dynstrtab, sdp->sd_name) == -1)
2383                                    return (S_ERROR);
2384                    }
2385            }
2386
2387            /*
2388             * Reserve entries for any per-symbol auxiliary/filter strings.
2389             */
2390            if (ofl->ofl_dtsfltrs != NULL) {
2391                    Dfltr_desc      *dftp;
2392                    Aliste          idx;
2393
2394                    for (ALIST_TRAVERSE(ofl->ofl_dtsfltrs, idx, dftp))
2395                            if (st_insert(ofl->ofl_dynstrtab, dftp->dft_str) == -1)
2396                                    return (S_ERROR);
2397            }
2398
2399            size = st_getstrtab_sz(ofl->ofl_dynstrtab);
2400            assert(size > 0);
2401
2402            if (new_section(ofl, SHT_STRTAB, MSG_ORIG(MSG_SCN_DYNSTR),
2403                0, &isec, &shdr, &data) == S_ERROR)
2404                    return (S_ERROR);
2405
2406            /* Make it allocable if necessary */
2407            if (!(ofl->ofl_flags & FLG_OF_RELOBJ))
2408                    shdr->sh_flags |= SHF_ALLOC;
2409
2410            /* Set the size of the data area */
2411            data->d_size = size + DYNSTR_EXTRA_PAD;
2412
2413            shdr->sh_size = (Xword)size;
2414
2415            ofl->ofl_osdynstr =
2416                ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_dynstr, NULL);
2417            return ((uintptr_t)ofl->ofl_osdynstr);
2418    }
2419
2420    /*
2421     * Generate an output relocation section which will contain the relocation
2422     * information to be applied to the 'osp' section.
2423     *
2424     * If (osp == NULL) then we are creating the coalesced relocation section
2425     * for an executable and/or a shared object.
2426     */
2427    static uintptr_t
2428    make_reloc(Ofl_desc *ofl, Os_desc *osp)
2429    {
2430            Shdr            *shdr;
2431            Elf_Data        *data;
2432            Is_desc         *isec;
2433            size_t          size;
2434            Xword           sh_flags;
2435            char            *sectname;
2436            Os_desc         *rosp;
2437            Word            relsize;
2438            const char      *rel_prefix;
2439
2440            /* LINTED */
2441            if (ld_targ.t_m.m_rel_sht_type == SHT_REL) {
2442                    /* REL */
2443                    relsize = sizeof (Rel);
2444                    rel_prefix = MSG_ORIG(MSG_SCN_REL);
2445            } else {
```



```
2446                    /* RELA */
2447                    relsize = sizeof (Rela);
2448                    rel_prefix = MSG_ORIG(MSG_SCN_RELA);
2449            }
2450
2451            if (osp) {
2452                    size = osp->os_szoutrels;
2453                    sh_flags = osp->os_shdr->sh_flags;
2454                    if ((sectname = libld_malloc(strlen(rel_prefix) +
2455                        strlen(osp->os_name) + 1)) == 0)
2456                            return (S_ERROR);
2457                    (void) strcpy(sectname, rel_prefix);
2458                    (void) strcat(sectname, osp->os_name);
2459            } else if (ofl->ofl_flags & FLG_OF_COMREL) {
2460                    size = (ofl->ofl_reloccnt - ofl->ofl_reloccntsub) * relsize;
2461                    sh_flags = SHF_ALLOC;
2462                    sectname = (char *)MSG_ORIG(MSG_SCN_SUNWRELOC);
2463            } else {
2464                    size = ofl->ofl_relocrelsz;
2465                    sh_flags = SHF_ALLOC;
2466                    sectname = (char *)rel_prefix;
2467            }
2468
2469            /*
2470             * Keep track of total size of 'output relocations' (to be stored
2471             * in .dynamic)
2472             */
2473            /* LINTED */
2474            ofl->ofl_relocsz += (Xword)size;
2475
2476            if (new_section(ofl, ld_targ.t_m.m_rel_sht_type, sectname, 0, &isec,
2477                &shdr, &data) == S_ERROR)
2478                    return (S_ERROR);
2479
2480            data->d_size = size;
2481
2482            shdr->sh_size = (Xword)size;
2483            if (OFL_ALLOW_DYNSYM(ofl) && (sh_flags & SHF_ALLOC))
2484                    shdr->sh_flags = SHF_ALLOC;
2485
2486            if (osp) {
2487                    /*
2488                     * The sh_info field of the SHT_REL* sections points to the
2489                     * section the relocations are to be applied to.
2490                     */
2491                    shdr->sh_flags |= SHF_INFO_LINK;
2492            }
2493
2494            rosp = ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_rel, NULL);
2495            if (rosp == (Os_desc *)S_ERROR)
2496                    return (S_ERROR);
2497
2498            /*
2499             * Associate this relocation section to the section its going to
2500             * relocate.
2501             */
2502            if (osp) {
2503                    Aliste  idx;
2504                    Is_desc *risp;
2505
2506                    /*
2507                     * This is used primarily so that we can update
2508                     * SHT_GROUP[sect_no] entries to point to the
2509                     * created output relocation sections.
2510                     */
2511                    for (APLIST_TRAVERSE(osp->os_relisdescs, idx, risp)) {
```

```
2512                        risp->is_osdesc = rosp;

2514                        /*
2515                         * If the input relocation section had the SHF_GROUP
2516                         * flag set - propagate it to the output relocation
2517                         * section.
2518                         */
2519                        if (risp->is_shdr->sh_flags & SHF_GROUP) {
2520                                rosp->os_shdr->sh_flags |= SHF_GROUP;
2521                                break;
2522                        }
2523                }
2524                osp->os_relosdesc = rosp;
2525        } else
2526                ofl->ofl_osrel = rosp;

2528        /*
2529         * If this is the first relocation section we've encountered save it
2530         * so that the .dynamic entry can be initialized accordingly.
2531         */
2532        if (ofl->ofl_osrelhead == (Os_desc *)0)
2533                ofl->ofl_osrelhead = rosp;

2535        return (1);
2536 }

2538 /*
2539  * Generate version needed section.
2540  */
2541 static uintptr_t
2542 make_verneed(Ofl_desc *ofl)
2543 {
2544        Shdr            *shdr;
2545        Elf_Data        *data;
2546        Is_desc         *isec;

2548        /*
2549         * verneed sections do not have a constant element size, so the
2550         * value of ent_cnt specified here (0) is meaningless.
2551         */
2552        if (new_section(ofl, SHT_SUNW_verneed, MSG_ORIG(MSG_SCN_SUNWVERSION),
2553            0, &isec, &shdr, &data) == S_ERROR)
2554                return (S_ERROR);

2556        /* During version processing we calculated the total size. */
2557        data->d_size = ofl->ofl_verneedsz;
2558        shdr->sh_size = (Xword)ofl->ofl_verneedsz;

2560        ofl->ofl_osverneed =
2561            ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_version, NULL);
2562        return ((uintptr_t)ofl->ofl_osverneed);
2563 }

2565 /*
2566  * Generate a version definition section.
2567  *
2568  *  o   the SHT_SUNW_verdef section defines the versions that exist within this
2569  *      image.
2570  */
2571 static uintptr_t
2572 make_verdef(Ofl_desc *ofl)
2573 {
2574        Shdr            *shdr;
2575        Elf_Data        *data;
2576        Is_desc         *isec;
2577        Ver_desc        *vdp;
```



```
2578        Str_tbl         *strtab;

2580        /*
2581         * Reserve a string table entry for the base version dependency (other
2582         * dependencies have symbol representations, which will already be
2583         * accounted for during symbol processing).
2584         */
2585        vdp = (Ver_desc *)ofl->ofl_verdesc->apl_data[0];

2587        if (OFL_IS_STATIC_OBJ(ofl))
2588                strtab = ofl->ofl_strtab;
2589        else
2590                strtab = ofl->ofl_dynstrtab;

2592        if (st_insert(strtab, vdp->vd_name) == -1)
2593                return (S_ERROR);

2595        /*
2596         * verdef sections do not have a constant element size, so the
2597         * value of ent_cnt specified here (0) is meaningless.
2598         */
2599        if (new_section(ofl, SHT_SUNW_verdef, MSG_ORIG(MSG_SCN_SUNWVERSION),
2600            0, &isec, &shdr, &data) == S_ERROR)
2601                return (S_ERROR);

2603        /* During version processing we calculated the total size. */
2604        data->d_size = ofl->ofl_verdefsz;
2605        shdr->sh_size = (Xword)ofl->ofl_verdefsz;

2607        ofl->ofl_osverdef =
2608            ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_version, NULL);
2609        return ((uintptr_t)ofl->ofl_osverdef);
2610 }

2612 /*
2613  * This routine is called when -z nopartial is in effect.
2614  */
2615 uintptr_t
2616 ld_make_parexpn_data(Ofl_desc *ofl, size_t size, Xword align)
2617 {
2618        Shdr            *shdr;
2619        Elf_Data        *data;
2620        Is_desc         *isec;
2621        Os_desc         *osp;

2623        if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_DATA), 0,
2624            &isec, &shdr, &data) == S_ERROR)
2625                return (S_ERROR);

2627        shdr->sh_flags |= SHF_WRITE;
2628        data->d_size = size;
2629        shdr->sh_size = (Xword)size;
2630        if (align != 0) {
2631                data->d_align = align;
2632                shdr->sh_addralign = align;
2633        }

2635        if ((data->d_buf = libld_calloc(size, 1)) == NULL)
2636                return (S_ERROR);

2638        /*
2639         * Retain handle to this .data input section. Variables using move
2640         * sections (partial initialization) will be redirected here when
2641         * such global references are added and '-z nopartial' is in effect.
2642         */
2643        ofl->ofl_isparexpn = isec;
```

```
2644         osp = ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_data, NULL);
2645         if (osp == (Os_desc *)S_ERROR)
2646                 return (S_ERROR);

2648         if (!(osp->os_flags & FLG_OS_OUTREL)) {
2649                 ofl->ofl_dynshdrcnt++;
2650                 osp->os_flags |= FLG_OS_OUTREL;
2651         }
2652         return (1);
2653 }

2655 /*
2656  * Make .sunwmove section
2657  */
2658 uintptr_t
2659 ld_make_sunwmove(Ofl_desc *ofl, int mv_nums)
2660 {
2661         Shdr            *shdr;
2662         Elf_Data        *data;
2663         Is_desc         *isec;
2664         Aliste          idx;
2665         Sym_desc        *sdp;
2666         int             cnt = 1;


2669         if (new_section(ofl, SHT_SUNW_move, MSG_ORIG(MSG_SCN_SUNWMOVE),
2670             mv_nums, &isec, &shdr, &data) == S_ERROR)
2671                 return (S_ERROR);

2673         if ((data->d_buf = libld_calloc(data->d_size, 1)) == NULL)
2674                 return (S_ERROR);

2676         /*
2677          * Copy move entries
2678          */
2679         for (APLIST_TRAVERSE(ofl->ofl_parsyms, idx, sdp)) {
2680                 Aliste          idx2;
2681                 Mv_desc         *mdp;

2683                 if (sdp->sd_flags & FLG_SY_PAREXPN)
2684                         continue;

2686                 for (ALIST_TRAVERSE(sdp->sd_move, idx2, mdp))
2687                         mdp->md_oidx = cnt++;
2688         }

2690         if ((ofl->ofl_osmove = ld_place_section(ofl, isec, NULL, 0, NULL)) ==
2691             (Os_desc *)S_ERROR)
2692                 return (S_ERROR);

2694         return (1);
2695 }

2697 /*
2698  * Given a relocation descriptor that references a string table
2699  * input section, locate the string referenced and return a pointer
2700  * to it.
2701  */
2702 static const char *
2703 strmerge_get_reloc_str(Ofl_desc *ofl, Rel_desc *rsp)
2704 {
2705         Sym_desc *sdp = rsp->rel_sym;
2706         Xword    str_off;

2708         /*
2709          * In the case of an STT_SECTION symbol, the addend of the
```



```
2710          * relocation gives the offset into the string section. For
2711          * other symbol types, the symbol value is the offset.
2712          */

2714         if (ELF_ST_TYPE(sdp->sd_sym->st_info) != STT_SECTION) {
2715                 str_off = sdp->sd_sym->st_value;
2716         } else if ((rsp->rel_flags & FLG_REL_RELA) == FLG_REL_RELA) {
2717                 /*
2718                  * For SHT_RELA, the addend value is found in the
2719                  * rel_raddend field of the relocation.
2720                  */
2721                 str_off = rsp->rel_raddend;
2722         } else {        /* REL and STT_SECTION */
2723                 /*
2724                  * For SHT_REL, the "addend" is not part of the relocation
2725                  * record. Instead, it is found at the relocation target
2726                  * address.
2727                  */
2728                 uchar_t *addr = (uchar_t *)((uintptr_t)rsp->rel_roffset +
2729                     (uintptr_t)rsp->rel_isdesc->is_indata->d_buf);

2731                 if (ld_reloc_targval_get(ofl, rsp, addr, &str_off) == 0)
2732                         return (0);
2733         }

2735         return (str_off + (char *)sdp->sd_isc->is_indata->d_buf);
2736 }

2738 /*
2739  * First pass over the relocation records for string table merging.
2740  * Build lists of relocations and symbols that will need modification,
2741  * and insert the strings they reference into the mstrtab string table.
2742  *
2743  * entry:
2744  *      ofl, osp - As passed to ld_make_strmerge().
2745  *      mstrtab - String table to receive input strings. This table
2746  *              must be in its first (initialization) pass and not
2747  *              yet cooked (st_getstrtab_sz() not yet called).
2748  *      rel_alpp - APlist to receive pointer to any relocation
2749  *              descriptors with STT_SECTION symbols that reference
2750  *              one of the input sections being merged.
2751  *      sym_alpp - APlist to receive pointer to any symbols that reference
2752  *              one of the input sections being merged.
2753  *      rcp - Pointer to cache of relocation descriptors to examine.
2754  *              Either &ofl->ofl_actrels (active relocations)
2755  *              or &ofl->ofl_outrels (output relocations).
2756  *
2757  * exit:
2758  *      On success, rel_alpp and sym_alpp are updated, and
2759  *      any strings in the mergeable input sections referenced by
2760  *      a relocation has been entered into mstrtab. True (1) is returned.
2761  *
2762  *      On failure, False (0) is returned.
2763  */
2764 static int
2765 strmerge_pass1(Ofl_desc *ofl, Os_desc *osp, Str_tbl *mstrtab,
2766     APlist **rel_alpp, APlist **sym_alpp, Rel_cache *rcp)
2767 {
2768         Aliste          idx;
2769         Rel_cachebuf    *rcbp;
2770         Sym_desc        *sdp;
2771         Sym_desc        *last_sdp = NULL;
2772         Rel_desc        *rsp;
2773         const char      *name;

2775         REL_CACHE_TRAVERSE(rcp, idx, rcbp, rsp) {
```

```
2776                sdp = rsp->rel_sym;
2777                if ((sdp->sd_isc == NULL) || ((sdp->sd_isc->is_flags &
2778                    (FLG_IS_DISCARD | FLG_IS_INSTRMRG)) != FLG_IS_INSTRMRG) ||
2779                    (sdp->sd_isc->is_osdesc != osp))
2780                        continue;

2782                /*
2783                 * Remember symbol for use in the third pass. There is no
2784                 * reason to save a given symbol more than once, so we take
2785                 * advantage of the fact that relocations to a given symbol
2786                 * tend to cluster in the list. If this is the same symbol
2787                 * we saved last time, don't bother.
2788                 */
2789                if (last_sdp != sdp) {
2790                        if (aplist_append(sym_alpp, sdp, AL_CNT_STRMRGSYM) ==
2791                            NULL)
2792                                return (0);
2793                        last_sdp = sdp;
2794                }

2796                /* Enter the string into our new string table */
2797                name = strmerge_get_reloc_str(ofl, rsp);
2798                if (st_insert(mstrtab, name) == -1)
2799                        return (0);

2801                /*
2802                 * If this is an STT_SECTION symbol, then the second pass
2803                 * will need to modify this relocation, so hang on to it.
2804                 */
2805                if ((ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION) &&
2806                    (aplist_append(rel_alpp, rsp, AL_CNT_STRMRGREL) == NULL))
2807                        return (0);
2808        }

2810        return (1);
2811 }

2813 /*
2814  * If the output section has any SHF_MERGE|SHF_STRINGS input sections,
2815  * replace them with a single merged/compressed input section.
2816  *
2817  * entry:
2818  *      ofl - Output file descriptor
2819  *      osp - Output section descriptor
2820  *      rel_alpp, sym_alpp, - Address of 2 APlists, to be used
2821  *              for internal processing. On the initial call to
2822  *              ld_make_strmerge, these list pointers must be NULL.
2823  *              The caller is encouraged to pass the same lists back for
2824  *              successive calls to this function without freeing
2825  *              them in between calls. This causes a single pair of
2826  *              memory allocations to be reused multiple times.
2827  *
2828  * exit:
2829  *      If section merging is possible, it is done. If no errors are
2830  *      encountered, True (1) is returned. On error, S_ERROR.
2831  *
2832  *      The contents of rel_alpp and sym_alpp on exit are
2833  *      undefined. The caller can free them, or pass them back to a subsequent
2834  *      call to this routine, but should not examine their contents.
2835  */
2836 static uintptr_t
2837 ld_make_strmerge(Ofl_desc *ofl, Os_desc *osp, APlist **rel_alpp,
2838     APlist **sym_alpp)
2839 {
2840        Str_tbl         *mstrtab;       /* string table for string merge secs */
2841        Is_desc         *mstrsec;       /* Generated string merge section */
```



```
2842        Is_desc         *isp;
2843        Shdr            *mstr_shdr;
2844        Elf_Data        *mstr_data;
2845        Sym_desc        *sdp;
2846        Rel_desc        *rsp;
2847        Aliste          idx;
2848        size_t          data_size;
2849        int             st_setstring_status;
2850        size_t          stoff;

2852        /* If string table compression is disabled, there's nothing to do */
2853        if ((ofl->ofl_flags1 & FLG_OF1_NCSTTAB) != 0)
2854                return (1);

2856        /*
2857         * Pass over the mergeable input sections, and if they haven't
2858         * all been discarded, create a string table.
2859         */
2860        mstrtab = NULL;
2861        for (APLIST_TRAVERSE(osp->os_mstrisdescs, idx, isp)) {
2862                if (isdesc_discarded(isp))
2863                        continue;

2865                /*
2866                 * Input sections of 0 size are dubiously valid since they do
2867                 * not even contain the NUL string.  Ignore them.
2868                 */
2869                if (isp->is_shdr->sh_size == 0)
 205                *if (isp->is_flags & FLG_IS_DISCARD)*
2870                        continue;

2872                /*
2873                 * We have at least one non-discarded section.
2874                 * Create a string table descriptor.
2875                 */
2876                if ((mstrtab = st_new(FLG_STNEW_COMPRESS)) == NULL)
2877                        return (S_ERROR);
2878                break;
2879        }

2881        /* If no string table was created, we have no mergeable sections */
2882        if (mstrtab == NULL)
2883                return (1);

2885        /*
2886         * This routine has to make 3 passes:
2887         *
2888         *      1) Examine all relocations, insert strings from relocations
2889         *              to the mergeable input sections into the string table.
2890         *      2) Modify the relocation values to be correct for the
2891         *              new merged section.
2892         *      3) Modify the symbols used by the relocations to reference
2893         *              the new section.
2894         *
2895         * These passes cannot be combined:
2896         *      - The string table code works in two passes, and all
2897         *              strings have to be loaded in pass one before the
2898         *              offset of any strings can be determined.
2899         *      - Multiple relocations reference a single symbol, so the
2900         *              symbol cannot be modified until all relocations are
2901         *              fixed.
2902         *
2903         * The number of relocations related to section merging is usually
2904         * a mere fraction of the overall active and output relocation lists,
2905         * and the number of symbols is usually a fraction of the number
2906         * of related relocations. We therefore build APlists for the
```

```
2907          * relocations and symbols in the first pass, and then use those
2908          * lists to accelerate the operation of pass 2 and 3.
2909          *
2910          * Reinitialize the lists to a completely empty state.
2911          */
2912         aplist_reset(*rel_alpp);
2913         aplist_reset(*sym_alpp);

2915         /*
2916          * Pass 1:
2917          *
2918          * Every relocation related to this output section (and the input
2919          * sections that make it up) is found in either the active, or the
2920          * output relocation list, depending on whether the relocation is to
2921          * be processed by this invocation of the linker, or inserted into the
2922          * output object.
2923          *
2924          * Build lists of relocations and symbols that will need modification,
2925          * and insert the strings they reference into the mstrtab string table.
2926          */
2927         if (strmerge_pass1(ofl, osp, mstrtab, rel_alpp, sym_alpp,
2928             &ofl->ofl_actrels) == 0)
2929                 goto return_s_error;
2930         if (strmerge_pass1(ofl, osp, mstrtab, rel_alpp, sym_alpp,
2931             &ofl->ofl_outrels) == 0)
2932                 goto return_s_error;

2934         /*
2935          * Get the size of the new input section. Requesting the
2936          * string table size "cooks" the table, and finalizes its contents.
2937          */
2938         data_size = st_getstrtab_sz(mstrtab);

2940         /* Create a new input section to hold the merged strings */
2941         if (new_section_from_template(ofl, isp, data_size,
2942             &mstrsec, &mstr_shdr, &mstr_data) == S_ERROR)
2943                 goto return_s_error;
2944         mstrsec->is_flags |= FLG_IS_GNSTRMRG;

2946         /*
2947          * Allocate a data buffer for the new input section.
2948          * Then, associate the buffer with the string table descriptor.
2949          */
2950         if ((mstr_data->d_buf = libld_malloc(data_size)) == NULL)
2951                 goto return_s_error;
2952         if (st_setstrbuf(mstrtab, mstr_data->d_buf, data_size) == -1)
2953                 goto return_s_error;

2955         /* Add the new section to the output image */
2956         if (ld_place_section(ofl, mstrsec, NULL, osp->os_identndx, NULL) ==
2957             (Os_desc *)S_ERROR)
2958                 goto return_s_error;

2960         /*
2961          * Pass 2:
2962          *
2963          * Revisit the relocation descriptors with STT_SECTION symbols
2964          * that were saved by the first pass. Update each relocation
2965          * record so that the offset it contains is for the new section
2966          * instead of the original.
2967          */
2968         for (APLIST_TRAVERSE(*rel_alpp, idx, rsp)) {
2969                 const char      *name;

2971                 /* Put the string into the merged string table */
2972                 name = strmerge_get_reloc_str(ofl, rsp);
```



```
2973                 st_setstring_status = st_setstring(mstrtab, name, &stoff);
2974                 if (st_setstring_status == -1) {
2975                         /*
2976                          * A failure to insert at this point means that
2977                          * something is corrupt. This isn't a resource issue.
2978                          */
2979                         assert(st_setstring_status != -1);
2980                         goto return_s_error;
2981                 }

2983                 /*
2984                  * Alter the relocation to access the string at the
2985                  * new offset in our new string table.
2986                  *
2987                  * For SHT_RELA platforms, it suffices to simply
2988                  * update the rel_raddend field of the relocation.
2989                  *
2990                  * For SHT_REL platforms, the new "addend" value
2991                  * needs to be written at the address being relocated.
2992                  * However, we can't alter the input sections which
2993                  * are mapped readonly, and the output image has not
2994                  * been created yet. So, we defer this operation,
2995                  * using the rel_raddend field of the relocation
2996                  * which is normally 0 on a REL platform, to pass the
2997                  * new "addend" value to ld_perform_outreloc() or
2998                  * ld_do_activerelocs(). The FLG_REL_NADDEND flag
2999                  * tells them that this is the case.
3000                  */
3001                 if ((rsp->rel_flags & FLG_REL_RELA) == 0)   /* REL */
3002                         rsp->rel_flags |= FLG_REL_NADDEND;
3003                 rsp->rel_raddend = (Sxword)stoff;

3005                 /*
3006                  * Generate a symbol name string for STT_SECTION symbols
3007                  * that might reference our merged section. This shows up
3008                  * in debug output and helps show how the relocation has
3009                  * changed from its original input section to our merged one.
3010                  */
3011                 if (ld_stt_section_sym_name(mstrsec) == NULL)
3012                         goto return_s_error;
3013         }

3015         /*
3016          * Pass 3:
3017          *
3018          * Modify the symbols referenced by the relocation descriptors
3019          * so that they reference the new input section containing the
3020          * merged strings instead of the original input sections.
3021          */
3022         for (APLIST_TRAVERSE(*sym_alpp, idx, sdp)) {
3023                 /*
3024                  * If we've already processed this symbol, don't do it
3025                  * twice. strmerge_pass1() uses a heuristic (relocations to
3026                  * the same symbol clump together) to avoid inserting a
3027                  * given symbol more than once, but repeat symbols in
3028                  * the list can occur.
3029                  */
3030                 if ((sdp->sd_isc->is_flags & FLG_IS_INSTRMRG) == 0)
3031                         continue;

3033                 if (ELF_ST_TYPE(sdp->sd_sym->st_info) != STT_SECTION) {
3034                         /*
3035                          * This is not an STT_SECTION symbol, so its
3036                          * value is the offset of the string within the
3037                          * input section. Update the address to reflect
3038                          * the address in our new merged section.
```

```
3039                          */
3040                         const char *name = sdp->sd_sym->st_value +
3041                             (char *)sdp->sd_isc->is_indata->d_buf;

3043                         st_setstring_status =
3044                             st_setstring(mstrtab, name, &stoff);
3045                         if (st_setstring_status == -1) {
3046                                 /*
3047                                  * A failure to insert at this point means
3048                                  * something is corrupt. This isn't a
3049                                  * resource issue.
3050                                  */
3051                                 assert(st_setstring_status != -1);
3052                                 goto return_s_error;
3053                         }

3055                         if (ld_sym_copy(sdp) == S_ERROR)
3056                                 goto return_s_error;
3057                         sdp->sd_sym->st_value = (Word)stoff;
3058                 }

3060                 /* Redirect the symbol to our new merged section */
3061                 sdp->sd_isc = mstrsec;
3062         }

3064         /*
3065          * There are no references left to the original input string sections.
3066          * Mark them as discarded so they don't go into the output image.
3067          * At the same time, add up the sizes of the replaced sections.
3068          */
3069         data_size = 0;
3070         for (APLIST_TRAVERSE(osp->os_mstrisdescs, idx, isp)) {
3071                 if (isp->is_flags & (FLG_IS_DISCARD | FLG_IS_GNSTRMRG))
3072                         continue;

3074                 data_size += isp->is_indata->d_size;

3076                 isp->is_flags |= FLG_IS_DISCARD;
3077                 DBG_CALL(Dbg_sec_discarded(ofl->ofl_lml, isp, mstrsec));
3078         }

3080         /* Report how much space we saved in the output section */
3081         DBG_CALL(Dbg_sec_genstr_compress(ofl->ofl_lml, osp->os_name, data_size,
3082             mstr_data->d_size));

3084         st_destroy(mstrtab);
3085         return (1);

3087 return_s_error:
3088         st_destroy(mstrtab);
3089         return (S_ERROR);
3090 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   88183 Mon Jul  7 11:53:57 2014
new/usr/src/cmd/sgs/packages/common/SUNWonld-README
4959 completely discarded merged string sections will corrupt output objects
**********************************************************
#
# 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
```

```
128 ------------------------------------------------------------------------------
129 4256518 miscalculated calloc() during dlclose/tsorting can result in segv
130 4254171 DT_SPARC_REGISTER has invalid value associated with it.
131 ------------------------------------------------------------------------------
132 All the above changes are incorporated in the following patches:
133         Solaris/SunOS 5.7_sparc         patch 106950-07
134         Solaris/SunOS 5.7_x86           patch 106951-07
135         Solaris/SunOS 5.6_sparc         patch 107733-04  (never released)
136         Solaris/SunOS 5.6_x86           patch 107734-04
137 ------------------------------------------------------------------------------
138 4293159 ld needs to combine sections with and without SHF_ORDERED flag(comdat)
139 4292238 linking a library which has a static char ptr invokes mprotect() call
140 ------------------------------------------------------------------------------
141 All the above changes except for:
142     4256518 miscalculated calloc() during dlclose/tsorting can result in segv
143     4254171 DT_SPARC_REGISTER has invalid value associated with it.
144 plus:
145     4238973 fix for 4121152 affects linking of Ada objects
146     4158744 patch 103627-02 causes core when RPATH has blank entry and
147             dlopen/dlclose is used
148 are incorporated in the following patches:
149         Solaris/SunOS 5.5.1_sparc       patch 103627-13
150         Solaris/SunOS 5.5.1_x86         patch 103628-12
151 ------------------------------------------------------------------------------
152 All the above changes are incorporated in the following patches:
153         Solaris/SunOS 5.7_sparc         patch 106950-08
154         Solaris/SunOS 5.7_x86           patch 106951-08
155         Solaris/SunOS 5.6_sparc         patch 107733-05
156         Solaris/SunOS 5.6_x86           patch 107734-05
157 ------------------------------------------------------------------------------
158 4295613 COMMON symbol resolution can be incorrect
159 ------------------------------------------------------------------------------
160 All the above changes plus:
161     4238973 fix for 4121152 affects linking of Ada objects
162     4158744 patch 103627-02 causes core when RPATH has blank entry and
163             dlopen/dlclose is used
164 are incorporated in the following patches:
165         Solaris/SunOS 5.5.1_sparc       patch 103627-14
166         Solaris/SunOS 5.5.1_x86         patch 103628-13
167 ------------------------------------------------------------------------------
168 All the above changes plus:
169     4351197 nfs performance problem by 103627-13
170 are incorporated in the following patches:
171         Solaris/SunOS 5.5.1_sparc       patch 103627-15
172         Solaris/SunOS 5.5.1_x86         patch 103628-14
173 ------------------------------------------------------------------------------
174 All the above changes are incorporated in the following patches:
175         Solaris/SunOS 5.7_sparc         patch 106950-09
176         Solaris/SunOS 5.7_x86           patch 106951-09
177         Solaris/SunOS 5.6_sparc         patch 107733-06
178         Solaris/SunOS 5.6_x86           patch 107734-06
179 ------------------------------------------------------------------------------
180 4158971 increase the default segment alignment for i386 to 64k
181 4064994 Add an $ISALIST token to those understood by the dynamic linker
182 xxxxxxx ia64 common code putback
183 4239308 LD_DEBUG busted for sparc machines
184 4239008 Support MAP_ANON
185 4238494 link-auditing extensions required
186 4232239 R_SPARC_LOX10 truncates field
187 4231722 R_SPARC_UA* relocations are busted
188 4235514 R_SPARC_OLO10 relocation fails
189 4244025 sgsmsg update
190 4239281 need to support SECREL relocations for ia64
191 4253751 ia64 linker must support PT_IA_64_UNWIND tables
192 4259254 dlmopen mistakenly closes fd 0 (stdin) under certain error conditions
193 4260872 libelf hangs when libthread present
```



```
194 4224569 linker core dumping when profiling specified
195 4270937 need mechanism to suppress ld.so.1's use of a default search path.
196 1050476 ld.so to permit configuration of search path
197 4273654 filtee processing using $ISALIST could be optimized
198 4271860 get MERCED cruft out of elf.h
199 4248991 Dynamic loader (via PLT) corrupts register G4
200 4275754 cannot mmap file: Resource temporarily unavailable
201 4277689 The linker can not handle relocation against MOVE tabl
202 4270766 atexit processing required on dlclose().
203 4279229 Add a "release" token to those understood by the dynamic linker
204 4215433 ld can bus error when insufficient disc space exists for output file
205 4285571 Pssst, want some free disk space?  ld's miscalculating.
206 4286236 ar gives confusing "bad format" error with a null .stab section
207 4286838 ld.so.1 can't handle a no-bits segment
208 4287364 ld.so.1 runtime configuration cleanup
209 4289573 disable linking of ia64 binaries for Solaris8
210 4293966 crle(1)'s default directories should be supplied
211 ------------------------------------------------------------------------------

213 ----------------------------------
214 Solaris 8 600 (1st Q-update - s28u1)
215 ----------------------------------
216 Bugid   Risk Synopsis
217 ==============================================================================
218 4309212 dlsym can't find symbol
219 4311226 rejection of preloading in secure apps is inconsistent
220 4312449 dlclose: invalid deletion of dependency can occur using RTLD_GLOBAL
221 ------------------------------------------------------------------------------
222 All the above changes are incorporated in the following patches:
223         Solaris/SunOS 5.8_sparc         patch 109147-01
224         Solaris/SunOS 5.8_x86           patch 109148-01
225         Solaris/SunOS 5.7_sparc         patch 106950-10
226         Solaris/SunOS 5.7_x86           patch 106951-10
227         Solaris/SunOS 5.6_sparc         patch 107733-07
228         Solaris/SunOS 5.6_x86           patch 107734-07
229 ------------------------------------------------------------------------------

231 ----------------------------------
232 Solaris 8 900 (2nd Q-update - s28u2)
233 ----------------------------------
234 Bugid   Risk Synopsis
235 ==============================================================================
236 4324775 non-PIC code & -zcombreloc don't mix very well...
237 4327653 run-time linker should preload tables it will process (madvise)
238 4324324 shared object code can be referenced before .init has fired
239 4321634 .init firing of multiple INITFIRST objects can fail
240 ------------------------------------------------------------------------------
241 All the above changes are incorporated in the following patches:
242         Solaris/SunOS 5.8_sparc         patch 109147-03
243         Solaris/SunOS 5.8_x86           patch 109148-03
244         Solaris/SunOS 5.7_sparc         patch 106950-11
245         Solaris/SunOS 5.7_x86           patch 106951-11
246         Solaris/SunOS 5.6_sparc         patch 107733-08
247         Solaris/SunOS 5.6_x86           patch 107734-08
248 ------------------------------------------------------------------------------
249 4338812 crle(1) omits entries in the directory cache
250 4341496 RFE: provide a static version of /usr/bin/crle
251 4340878 rtld should treat $ORIGIN like LD_LIBRARY_PATH in security issues
252 ------------------------------------------------------------------------------
253 All the above changes are incorporated in the following patches:
254         Solaris/SunOS 5.8_sparc         patch 109147-04
255         Solaris/SunOS 5.8_x86           patch 109148-04
256         Solaris/SunOS 5.7_sparc         patch 106950-12
257         Solaris/SunOS 5.7_x86           patch 106951-12
258 ------------------------------------------------------------------------------
259 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
--------------------------------------------------------------------------------
```

```
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-11
        Solaris/SunOS 5.8_x86           patch 109148-11
-------------------------------------------------------------------------------

-----------------------------------
Solaris 8 202 (7th Q-update - s28u7)
-----------------------------------
Bugid   Risk Synopsis
===============================================================================
4488954 ld.so.1 reuses same buffer to send ummapping range to
        _preexec_exit_handlers()
-------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-12
        Solaris/SunOS 5.8_x86           patch 109148-12
-------------------------------------------------------------------------------

---------
Solaris 9
---------
Bugid   Risk Synopsis
===============================================================================
4505289 incorrect handling of _START_ and _END_
4506164 mcs does not recognize #linkbefore or #linkafter qualifiers
4447560 strip is creating unexecutable files...
4513842 library names not in ld.so string pool cause corefile bugs
-------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-13
        Solaris/SunOS 5.8_x86           patch 109148-13
        Solaris/SunOS 5.7_sparc         patch 106950-16
        Solaris/SunOS 5.7_x86           patch 106951-16
-------------------------------------------------------------------------------
4291384 ld -M with a mapfile does not properly align Fortran REAL*8 data
4413322 SunOS 5.9 librtld_db doesn't show dlopened ".o" files anymore?
4429371 librtld_db busted on ia32 with SC6.x compilers...
4418274 elfdump dumps core on invalid input
4432224 libelf xlate routines are out of date
4433643 Memory leak using dlopen()/dlclose() in Solaris 8
4446564 ldd/lddstub - core dump conditions
4446115 translating SUNW_move sections is broken
4450225 The rdb command can fall into an infinite loop
4448531 Linker Causes Segmentation Fault
4453241 Regression in 4291384 can result in empty symbol table.
4453398 invalid runpath token can cause ld to spin.
4460230 ld (for OS 5.8 and 5.9) loses error message
4462245 ld.so.1 core dumps when executed directly...
4455802 need more flexibility in establishing a support library for ld
4467068 dyn_plt_entsize not properly initialized in ld.so.1
4468779 elf_plt_trace_write() broken on i386 (link-auditing)
4465871 -zld32 and -zld64 does not work the way it should
4461890 bad shared object created with -zredlocsym
4469400 ld.so.1: is_so_loaded isn't as efficient as we thought...
4469566 lazy loading fallback can reference un-relocated objects
4470493 libelf incorectly translates NOTE sections accross architectures...
4469684 rtld leaks dl_handles and permits on dlopen/dlclose
4475174 ld.so.1 prematurly reports the failure to load a object...
4475514 ld.so.1 can core dump in memory allocation fails (no swap)
4481851 Setting ld.so.1 environment variables globally would be useful
4482035 setting LD_PROFILE & LD_AUDIT causes ping command to issue warnings
        on 5.8
4377735 segment reservations cause sbrk() to fail
4491434 ld.so.1 can leak file-descriptors when loading same named objects
4289232 some of warning/error/debugging messages from libld.so can be revised
4462748 Linker Portion of TLS Support
```



```
4496718 run-time linkers mutex_locks not working with ld_libc interface
4497270 The -zredlocsym option should not eliminate partially initialized local
        symbols
4496963 dumping an object with crle(1) that uses $ORIGIN can loose its
        dependencies
4499413 Sun linker orders of magnitude slower than gnu linker
4461760 lazy loading libXm and libXt can fail.
4469031 The partial initialized (local) symbols for intel platform is not
        working.
4492883 Add link-editor option to multi-pass archives to resolve unsatisfied
        symbols
4503731 linker-related commands misspell "argument"
4503768 whocalls(1) should output messages to stderr, not stdout
4503748 whocalls(1) usage message and manpage could be improved
4503625 nm should be taught about TLS symbols - that they aren't allowed that is
4300120 segment address validation is too simplistic to handle segment
        reservations
4404547 krtld/reloc.h could have better error message, has typos
4270931 R_SPARC_HIX22 relocation is not handled properly
4485320 ld needs to support more the 32768 PLTs
4516434 sotruss can not watch libc_psr.so.1
4213100 sotruss could use more flexible pattern matching
4503457 ld seg fault with comdat
4510264 sections with SHF_TLS can come in different orders...
4518079 link-editor support library unable to modify section header flags
4515913 ld.so.1 can incorrectly decrement external reference counts on dlclose()
4519569 ld -V does not return a interesting value...
4524512 ld.so.1 should allow alternate termination signals
4524767 elfdump dies on bogus sh_name fields...
4524735 ld getopt processing of '-' changed
4521931 subroutine in a shared object as LOCL instead of GLOB
-------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-14
        Solaris/SunOS 5.8_x86           patch 109148-14
        Solaris/SunOS 5.7_sparc         patch 106950-17
        Solaris/SunOS 5.7_x86           patch 106951-17
-------------------------------------------------------------------------------
4532729 tentative definition of TLS variable causes linker to dump core
4526745 fixup ld error message about duplicate dependencies/needed names
4522999 Solaris linker one order of magnitude slower than GNU linker
4518966 dldump undoes existing relocations with no thought of alignment or size.
4587441 Certain libraries have race conditions when setting error codes
4523798 linker option to align bss to large pagesize alignments.
4524008 ld can improperly set st_size of symbols named "_init" or "_fini"
4619282 ld cannot link a program with the option -sb
4620846 Perl Configure probing broken by ld changes
4621122 multiple ld '-zinitarray=' on a commandline fails
-------------------------------------------------------------------------------
        Solaris/SunOS 5.8_sparc         patch 109147-15
        Solaris/SunOS 5.8_x86           patch 109148-15
        Solaris/SunOS 5.7_sparc         patch 106950-18
        Solaris/SunOS 5.7_x86           patch 106951-18
        Solaris/SunOS 5.6_sparc         patch 107733-10
        Solaris/SunOS 5.6_x86           patch 107734-10
-------------------------------------------------------------------------------
All the above changes plus:
        4616944 ar seg faults when order of object file is reversed.
are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-16
        Solaris/SunOS 5.8_x86           patch 109148-16
-------------------------------------------------------------------------------
All the above changes plus:
        4872634 Large LD_PRELOAD values can cause SEGV of process
are incorporated in the following patches:
        Solaris/SunOS 5.6_sparc         patch T107733-11
```

```
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
655 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
--------------------------------------------------------------------------------

-------------------------------------
```

```
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 --------------------------------------------------------------------------------
```

All the above changes are incorporated in the following patches:

Solaris/SunOS 5.9_sparc	patch T112963-24
Solaris/SunOS 5.9_x86	patch T113986-20
Solaris/SunOS 5.8_sparc	patch T109147-39
Solaris/SunOS 5.8_x86	patch T109148-39

6318306 a dlsym() from a filter should be redirected to an associated filtee
6318401 mis-aligned TLS variable
6324019 ld.so.1: malloc alignment is insufficient for new compilers
6324589 psh coredumps on x86 machines on snv_23
6236594 AMD64: Linker needs to handle the new .lbss section (D)
PSARC 2005/514 AMD64 - large section support
6314743 Linker: incorrect resolution for R_AMD64_GOTPC32
6311865 Linker: x86 medium model; invalid ELF program header

All the above changes are incorporated in the following patches:

Solaris/SunOS 5.10_sparc	patch T117461-07
Solaris/SunOS 5.10_x86	patch T118345-12

6309061 link_audit should use __asm__ with gcc
6310736 gcc and sgs/libld don’t get along on SPARC
6329796 Memory leak with iconv_open/iconv_close with patch 109147-33
6332983 s9 linker patches 112963-24/113986-20 causing cluster machines not
to boot

All the above changes are incorporated in the following patches:

Solaris/SunOS 5.10_sparc	patch T117461-08
Solaris/SunOS 5.10_x86	patch T121208-02
Solaris/SunOS 5.9_sparc	patch T112963-25
Solaris/SunOS 5.9_x86	patch T113986-21
Solaris/SunOS 5.8_sparc	patch T109147-40
Solaris/SunOS 5.8_x86	patch T109148-40

6445311 The sparc S8/S9/S10 linker patches which include the fix for the
CR6222525 are hit by the CR6439613.

All the above changes are incorporated in the following patches:

Solaris/SunOS 5.9_sparc	patch T112963-26
Solaris/SunOS 5.8_sparc	patch T109147-41

Solaris 10 807 (4th Q-update - s10u4)

Bugid Risk Synopsis

6487273 ld.so.1 may open arbitrary locale files when relative path is built
from locale environment vars
6487284 ld.so.1: buffer overflow in doprf() function

All the above changes are incorporated in the following patches:

Solaris/SunOS 5.10_sparc	patch T124922-01
Solaris/SunOS 5.10_x86	patch T124923-01
Solaris/SunOS 5.9_sparc	patch T112963-27
Solaris/SunOS 5.9_x86	patch T113986-22
Solaris/SunOS 5.8_sparc	patch T109147-42
Solaris/SunOS 5.8_x86	patch T109148-41

6477132 ld.so.1: memory leak when running set*id application

All the above changes are incorporated in the following patches:

Solaris/SunOS 5.10_sparc	patch T124922-02
Solaris/SunOS 5.10_x86	patch T124923-02
Solaris/SunOS 5.9_sparc	patch T112963-30
Solaris/SunOS 5.9_x86	patch T113986-24

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

All the above changes are incorporated in the following patches:

Solaris/SunOS 5.10_sparc	patch T124922-03
Solaris/SunOS 5.10_x86	patch T124923-03

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

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

---

Solaris 10 508 (5th Q-update - s10u5)

Bugid Risk Synopsis

```
6561987 data vac_conflict faults on lipthread libthread libs in s10.
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T127111-01
        Solaris/SunOS 5.10_x86          patch T127112-01
--------------------------------------------------------------------------------
6501793 GOTOP relocation transition (optimization) fails with offsets > 2^32
6532924 AMD64: Solaris 5.11 55b: SEGV after whocatches
6551627 OGL: SIGSEGV when trying to use OpenGL pipeline with splash screen,
        Solaris/Nvidia only
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T127111-04
        Solaris/SunOS 5.10_x86          patch T127112-04
--------------------------------------------------------------------------------
6479848 Enhancements to the linker support interface needed. (D)
        PSARC/2006/595 link-editor support library interface - ld_open()
6521608 assertion failure in runtime linker related to auditing
6494228 pclose() error when an audit library calls popen() and the main target
        is being run under ldd (D)
6568745 segfault when using LD_DEBUG with bit_audit library when instrumenting
        mozilla (D)
        PSARC/2007/413 Add -zglobalaudit option to ld
6602294 ps_pbrandname breaks apps linked directly against librtld_db
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T127111-07
        Solaris/SunOS 5.10_x86          patch T127112-07
--------------------------------------------------------------------------------

------------------------------------
Solaris 10 908 (6th Q-update - s10u6)
------------------------------------
Bugid   Risk Synopsis
================================================================================
6672544 elf_rtbndr must support non-ABI aligned stacks on amd64
6668050 First trip through PLT does not preserve args in xmm registers
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patch:
        Solaris/SunOS 5.10_x86          patch T137138-01
--------------------------------------------------------------------------------

------------------------------------
Solaris 10 409 (7th Q-update - s10u7)
------------------------------------
Bugid   Risk Synopsis
================================================================================
6629404 ld with -z ignore doesn't scale
6606203 link editor ought to allow creation of >2gb sized objects (P)
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T139574-01
        Solaris/SunOS 5.10_x86          patch T139575-01
--------------------------------------------------------------------------------
6746674 setuid applications do not find libraries any more because trusted
        directories behavior changed (D)
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T139574-02
        Solaris/SunOS 5.10_x86          patch T139575-02
--------------------------------------------------------------------------------
6703683 Can't build VirtualBox on Build 88 or 89
6737579 process_req_lib() in libld consumes file descriptors
6685125 ld/elfdump do not handle ZERO terminator .eh_frame amd64 unwind entry
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
```



```
        Solaris/SunOS 5.10_sparc        patch T139574-03
        Solaris/SunOS 5.10_x86          patch T139575-03
--------------------------------------------------------------------------------

------------------------------------
Solaris 10 1009 (8th Q-update - s10u8)
------------------------------------
Bugid   Risk Synopsis
================================================================================
6782597 32-bit ld.so.1 needs to accept objects with large inode number
6805502 The addition of "inline" keywords to sgs code broke the lint
        verification in S10
6807864 ld.so.1 is susceptible to a fatal dlsym()/setlocale() race
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T141692-01
        Solaris/SunOS 5.10_x86          patch T141693-01
NOTE: The fix for 6805502 is only applicable to s10.
--------------------------------------------------------------------------------
6826410 ld needs to sort sections using 32-bit sort keys
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T141771-01
        Solaris/SunOS 5.10_x86          patch T141772-01
NOTE: The fix for 6826410 is also available for s9 in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-33
        Solaris/SunOS 5.9_x86           patch T113986-27
--------------------------------------------------------------------------------
6568447 bcp is broken by 6551627
6599700 librtld_db needs better plugin support
6713830 mdb dumped core reading a gcore
6756048 rd_loadobj_iter() should always invoke brand plugin callback
6786744 32-bit dbx failed with unknown rtld_db.so error on snv_104
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T141444-06
        Solaris/SunOS 5.10_x86          patch T141445-06
--------------------------------------------------------------------------------

------------------------------------
Solaris 10 1005 (9th Q-update - s10u9)
------------------------------------
Bugid   Risk Synopsis
================================================================================
6850124 dlopen reports "No such file or directory" in spite of ENOMEM
        when mmap fails in anon_map()
6826513 ldd gets confused by a crle(1) LD_PRELOAD setting
6684577 ld should propagate SHF_LINK_ORDER flag to ET_REL objects
6524709 executables using /usr/lib/libc.so.1 as the ELF interpreter dump core
        (link-editor components only)
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch T143895-01
        Solaris/SunOS 5.10_x86          patch T143896-01
--------------------------------------------------------------------------------

------------------------------------
Solaris 10 XXXX (10th Q-update - s10u10)
------------------------------------
Bugid   Risk Synopsis
================================================================================
6478684 isainfo/cpuid reports pause instruction not supported on amd64
        PSARC/2010/089 Removal of AV_386_PAUSE and AV_386_MON
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.10_sparc        patch TXXXXXX-XX
```

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

----------------------------------------
Solaris Nevada (OpenSolaris 2008.05, snv_86)
----------------------------------------
Bugid   Risk Synopsis
================================================================================
6409350 BrandZ project integration into Solaris (link-editor components only)
6459189 UNIX03: *VSC* c99 compiler overwrites non-writable file
6423746 add an option to relax the resolution of COMDAT relocs (D)
4934427 runtime linker should load up static symbol names visible to
        dladdr() (D)
        PSARC 2006/526 SHT_SUNW_LDYNSYM - default local symbol addition
6448719 sys/elf.h could be updated with additional machine and ABI types
6336605 link-editors need to support R_*_SIZE relocations
        PSARC/2006/558 R_*_SIZE relocation support
6475375 symbol search optimization to reduce rescans
6475497 elfdump(1) is misreporting sh_link
6482058 lari(1) could be faster, and handle per-symbol filters better
6482974 defining virtual address of text segment can result in an invalid data
        segment
6476734 crle(1m) "-l" as described fails system, crle cores trying to fix
        /a/var/ld/ld.config in failsafe
6487499 link_audit "make clobber" creates and populates proto area
6488141 ld(1) should detect attempt to reference 0-length .bss section
6496718 restricted visibility symbol references should trigger archive
        extraction
6515970 HWCAP processing doesn't clean up fmap structure - browser fails to
        run java applet
6494214 Refinements to symbolic binding, symbol declarations and
        interposition (D)
        PSARC/2006/714 ld(1) mapfile: symbol interpose definition
6475344 DTrace needs ELF function and data symbols sorted by address (D)
        PSARC/2007/026 ELF symbol sort sections
6518480 ld -melf_i386 doesn't complain (D)
6519951 bfu is just another word for exit today (RPATH -> RUNPATH conversion
        bites us) (D)
6521504 ld: hardware capabilities processing from relocatables objects needs
        hardening.
6518322 Some ELF utilities need updating for .SUNW_ldynsym section (D)
        PSARC/2007/074 -L option for nm(1) to display SHT_SUNW_LDYNSYM symbols
6523787 dlopen() handle gets mistakenly orphaned - results in access to freed
        memory
6531189 SEGV in dladdr()
6527318 dlopen(name, RTLD_NOLOAD) returns handle for unloaded library
6518359 extern mapfiles references to _init/_fini can create INIT/FINI
        addresses of 0
6533587 ld.so.1: init/fini processing needs to compensate for interposer
        expectations
6516118 Reserved space needed in ELF dynamic section and string table (D)
        PSARC/2007/127 Reserved space for editing ELF dynamic sections
6535688 elfdump could be more robust in the face of Purify (D)
6516665 The link-editors should be more resilient against gcc's symbol
        versioning
6541004 hwcap filter processing can leak memory
5108874 elfdump SEGVs on bad object file
6547441 Uninitialized variable causes ld.so.1 to crash on object cleanup
6341667 elfdump should check alignments of ELF header elements
6387860 elfdump cores, when processing linux built ELF file
6198202 mcs -d dumps core
6246083 elfdump should allow section index specification
        (numeric -N equivalent) (D)
        PSARC/2007/247 Add -I option to elfdump
6556563 elfdump section overlap checking is too slow for large files
5006034 need ?E mapfile feature extension (D)
```



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

----------------------------------------
Solaris Nevada (OpenSolaris 2008.11, snv_101)
----------------------------------------
Bugid   Risk Synopsis
================================================================================
6671255 link-editor should support cross linking (D)
        PSARC/2008/179 cross link-editor
6674666 elfedit dyn:posflag1 needs option to locate element via NEEDED item
6675591 elfwrap - wrap data in an ELF file (D,P)
        PSARC/2008/198 elfwrap - wrap data in an ELF file
6678244 elfdump dynamic section sanity checking needs refinement
6679212 sgs use of SCCS id for versioning is obstacle to mercurial migration
```

```
1448 6681761 lies, darn lies, and linker README files
1449 6509323 Need to disable the Multiple Files loading - same name, different
1450         directories (or its stat() use)
1451 6686889 ld.so.1 regression - bad pointer created with 6509323 integration
1452 6695681 ldd(1) crashes when run from a chrooted environment
1453 6516212 usr/src/cmd/sgs/libelf warlock targets should be fixed or abandoned
1454 6678310 using LD_AUDIT, ld.so.1 calls shared library's .init before library is
1455         fully relocated (link-editor components only)
1456 6699594 The ld command has a problem handling 'protected' mapfile keyword.
1457 6699131 elfdump should display core file notes (D)
1458 6702260 single threading .init/.fini sections breaks staroffice
1459 6703919 boot hangs intermittently on x86 with onnv daily.0430 and on
1460 6701798 ld can enter infinite loop processing bad mapfile
1461 6706401 direct binding copy relocation fallback is insufficient for ild
1462         generated objects
1463 6705846 multithreaded C++ application seems to get deadlocked in the dynamic
1464         linker code
1465 6686343 ldd(1) - unused search path diagnosis should be enabled
1466 6712292 ld.so.1 should fall back to an interposer for failed direct bindings
1467 6716350 usr/src/cmd/sgs should be linted by nightly builds
1468 6720509 usr/src/cmd/sgs/sgsdemangler should be removed
1469 6617475 gas creates erroneous FILE symbols [was: ld.so.1 is reported as
1470         false positive by wsdiff]
1471 6724311 dldump() mishandles R_AMD64_JUMP_SLOT relocations
1472 6724774 elfdump -n doesn't print siginfo structure
1473 6728555 Fix for amd64 aw (6617475) breaks pure gcc builds
1474 6734598 ld(1) archive processing failure due to mismatched file descriptors (D)
1475 6735939 ld(1) discarded symbol relocations errors (Studio and GNU).
1476 6354160 Solaris linker includes more than one copy of code in binary when
1477         linking gnu object code
1478 6744003 ld(1) could provide better argument processing diagnostics (D)
1479         PSARC 2008/583 add gld options to ld(1)
1480 6749055 ld should generate GNU style VERSYM indexes for VERNEED records (D)
1481         PSARC/2008/603 ELF objects to adopt GNU-style Versym indexes
1482 6752728 link-editor can enter UNDEF symbols in symbol sort sections
1483 6756472 AOUT search path pruning (D)
1484 --------------------------------------------------------------------------------

1486 ------------------------------------------
1487 Solaris Nevada (OpenSolaris 2009.06, snv_111)
1488 ------------------------------------------
1489 Bugid   Risk Synopsis
1490 ================================================================================

1492 6754965 introduce the SF1_SUNW_ADDR32 bit in software capabilities (D)
1493         (link-editor components only)
1494         PSARC/2008/622 32-bit Address Restriction Software Capabilities Flag
1495 6756953 customer requests that DT_CONFIG strings be honored for secure apps (D)
1496 6765299 ld --version-script option not compatible with GNU ld (D)
1497 6748160 problem with -zrescan (D)
1498         PSARC/2008/651 New ld archive rescan options
1499 6763342 sloppy relocations need to get sloppier
1500 6736890 PT_SUNWBSS should be disabled (D)
1501         PSARC/2008/715 PT_SUNWBSS removal
1502 6772661 ldd/lddstub/ld.so.1 dump core in current nightly while processing
1503         libsoftcrypto_hwcap.so.1
1504 6765931 mcs generates unlink(NULL) system calls
1505 6775062 remove /usr/lib/libldstab.so (D)
1506 6782977 ld segfaults after support lib version error sends bad args to vprintf()
1507 6773695 ld -z nopartial can break non-pic objects
1508 6778453 RTLD_GROUP prevents use of application defined malloc
1509 6789925 64-bit applications with SF1_SUNW_ADDR32 require non-default starting
1510         address
1511 6792906 ld -z nopartial fix breaks TLS
1512 6686372 ld.so.1 should use mmapobj(2)
1513 6726108 dlopen() performance could be improved.
```



```
1514 6792836 ld is slow when processing GNU linkonce sections
1515 6797468 ld.so.1: orphaned handles aren't processed correctly
1516 6798676 ld.so.1: enters infinite loop with realloc/defragmentation logic
1517 6237063 request extension to dl* family to provide segment bounds
1518         information (D)
1519         PSARC/2009/054 dlinfo(3c) - segment mapping retrieval
1520 6800388 shstrtab can be sized incorrectly when -z ignore is used
1521 6805009 ld.so.1: link map control list tear down leaves dangling pointer -
1522         pfinstall does it again.
1523 6807050 GNU linkonce sections can create duplicate and incompatible
1524         eh_frame FDE entries
1525 --------------------------------------------------------------------------------

1527 --------------
1528 Solaris Nevada
1529 --------------
1530 Bugid   Risk Synopsis
1531 ================================================================================
1532 6813909 generalize eh_frame support to non-amd64 platforms
1533 6801536 ld: mapfile processing oddities unveiled through mmapobj(2) observations
1534 6802452 libelf shouldn't use MS_SYNC
1535 6818012 nm tries to modify readonly segment and dumps core
1536 6821646 xVM dom0 doesn't boot on daily.0324 and beyond
1537 6822828 librtld_db can return RD_ERR before RD_NOMAPS, which compromises dbx
1538         expectations.
1539 6821619 Solaris linkers need systematic approach to ELF OSABI (D)
1540         PSARC/2009/196 ELF objects to set OSABI / elfdump -O option
1541 6827468 6801536 breaks 'ld -s' if there are weak/strong symbol pairs
1542 6715578 AOUT (BCP) symbol lookup can be compromised with lazy loading.
1543 6752883 ld.so.1 error message should be buffered (not sent to stderr).
1544 6577982 ld.so.1 calls getpid() before it should when any LD_* are set
1545 6831285 linker LD_DEBUG support needs improvements (D)
1546 6806791 filter builds could be optimized (link-editor components only)
1547 6823371 calloc() uses suboptimal memset() causing 15% regression in SpecCPU2006
1548         gcc code (link-editor components only)
1549 6831308 ld.so.1: symbol rescanning does a little too much work
1550 6837777 ld ordered section code uses too much memory and works too hard
1551 6841199 Undo 10 year old workaround and use 64-bit ld on 32-bit objects
1552 6784790 ld should examine archives to determine output object class/machine (D)
1553         PSARC/2009/305 ld -32 option
1554 6849998 remove undocumented mapfile $SPECVERS and $NEED options
1555 6851224 elf_getshnum() and elf_getshstrndx() incompatible with 2002 ELF gABI
1556         agreement (D)
1557         PSARC/2009/363 replace elf_getphnum, elf_getshnum, and elf_getshstrndx
1558 6853809 ld.so.1: rescan fallback optimization is invalid
1559 6854158 ld.so.1: interposition can be skipped because of incorrect
1560         caller/destination validation
1561 6862967 rd_loadobj_iter() failing for core files
1562 6856173 streams core dumps when compiled in 64bit with a very large static
1563         array size
1564 6834197 ld pukes when given an empty plate
1565 6516644 per-symbol filtering shouldn't be allowed in executables
1566 6878605 ld should accept '%' syntax when matching input SHT_PROGBITS sections
1567 6850768 ld option to autogenerate wrappers/interposers similar to GNU ld
1568         --wrap (D)
1569         PSARC/2009/493 ld -z wrap option
1570 6888489 Null environment variables are not overriding crle(1) replaceable
1571         environment variables.
1572 6885456 Need to implement GNU-ld behavior in construction of .init/.fini
1573         sections
1574 6900241 ld should track SHT_GROUP sections by symbol name, not section name
1575 6901773 Special handling of STT_SECTION group signature symbol for GNU objects
1576 6901895 Failing asserts in ld update_osym() trying to build gcc 4.5 develpment
1577         head
1578 6909523 core dump when run "LD_DEBUG=help ls" in non-English locale
1579 6903688 mdb(1) can't resolve certain symbols in solaris10-branded processes
```

```
1580         from the global zone
1581 6923449 elfdump misinterprets _init/_fini symbols in dynamic section test
1582 6914728 Add dl_iterate_phdr() function to ld.so.1 (D)
1583         PSARC/2010/015 dl_iterate_phdr
1584 6916788 ld version 2 mapfile syntax (D)
1585         PSARC/2009/688 Human readable and extensible ld mapfile syntax
1586 6929607 ld generates incorrect VERDEF entries for ET_REL output objects
1587 6924224 linker should ignore SUNW_dof when calculating the elf checksum
1588 6918143 symbol capabilities (D)
1589         PSARC/2010/022 Linker-editors: Symbol Capabilities
1590 6910387 .tdata and .tbss separation invalidates TLS program header information
1591 6934123 elfdump -d coredumps on PA-RISC elf
1592 6931044 ld should not allow SHT_PROGBITS .eh_frame sections on amd64 (D)
1593 6931056 pvs -r output can include empty versions in output
1594 6938628 ld.so.1 should produce diagnostics for all dl*() entry points
1595 6938111 nm 'No symbol table data' message goes to stdout
1596 6941727 ld relocation cache memory use is excessive
1597 6932220 ld -z allextract skips objects that lack global symbols
1598 6943772 Testing for a symbols existence with RTLD_PROBE is compromised by
1599         RTLD_BIND_NOW
1600         PSARC/2010/XXX Deferred symbol references
1601 6943432 dlsym(RTLD_PROBE) should only bind to symbol definitions
1602 6668759 an external method for determining whether an ELF dependency is optional
1603 6954032 Support library with ld_open and -z allextract in snv_139 do not mix
1604 6949596 wrong section alignment generated in joint compilation with shared
1605         library
1606 6961755 ld.so.1's -e arguments should take precedence over environment
1607         variables. (D)
1608 6748925 moe returns wrong hwcap library in some circumstances
1609 6916796 OSnet mapfiles should use version 2 link-editor syntax
1610 6964517 OSnet mapfiles should use version 2 link-editor syntax (2nd pass)
1611 6948720 SHT_INIT_ARRAY etc. section names don't follow ELF gABI (D)
1612 6962343 sgsmsg should use mkstemp() for temporary file creation
1613 6965723 libsoftcrypto symbol capabilities rely on compiler generated
1614         capabilities - gcc failure (link-editor components only)
1615 6952219 ld support for archives larger than 2 GB (D, P)
1616         PSARC/2010/224 Support for archives larger than 2 GB
1617 6956152 dlclose() from an auditor can be fatal. Preinit/activity events should
1618         be more flexible. (D)
1619 6971440 moe can core dump while processing libc.
1620 6972234 sgs demo's could use some cleanup
1621 6935867 .dynamic could be readonly in sharable objects
1622 6975290 ld mishandles GOT relocation against local ABS symbol
1623 6972860 ld should provide user guidance to improve objects (D)
1624         PSARC/2010/312 Link-editor guidance
1625 ------------------------------------------------------------------------------

1627 --------------
1628 Illumos
1629 --------------
1630 Bugid   Risk Synopsis
1631 ==============================================================================

1633 308     ld may misalign sections only preceded by empty sections
1634 1301    ld crashes with '-z ignore' due to a null data descriptor
1635 1626    libld may accidentally return success while failing
1636 2413    %ymm* need to be preserved on way through PLT
1637 3210    ld should tolerate SHT_PROGBITS for .eh_frame sections on amd64
1638 3228    Want -zassert-deflib for ld
1639 3230    ld.so.1 should check default paths for DT_DEPAUDIT
1640 3260    linker is insufficiently careful with strtok
1641 3261    linker should ignore unknown hardware capabilities
1642 3265    link-editor builds bogus .eh_frame_hdr on ia32
1643 3453    GNU comdat redirection does exactly the wrong thing
1644 3439    discarded sections shouldn't end up on output lists
1645 3436    relocatable objects also need sloppy relocation
```



```
1646 3451    archive libraries with no symbols shouldn't require a string table
1647 3616    SHF_GROUP sections should not be discarded via other COMDAT mechanisms
1648 3709    need sloppy relocation for GNU .debug_macro
1649 3722    link-editor is over restrictive of R_AMD64_32 addends
1650 3926    multiple extern map file definitions corrupt symbol table entry
1651 3999    libld extended section handling is broken
1652 4003    dldump() can't deal with extended sections
1653 4227    ld --library-path is translated to -l-path, not -L
1654 4270    ld(1) argument error reporting is still pretty bad
1655 4383    libelf can't write extended sections when ELF_F_LAYOUT
1656 4959    completely discarded merged string sections will corrupt output objects
1657 #endif /* ! codereview */
```