```
**********************************************************
   15323 Wed May 22 03:21:43 2019
new/usr/src/cmd/sgs/libld/common/entry.c
11057 hidden undefined weak symbols should not leave relocations
11058 libld entrance descriptor assertions get NDEBUG check backwards
**********************************************************
______unchanged_portion_omitted_

 348 /*
 349  * Initialize new entrance and segment descriptors and add them as lists to
 350  * the output file descriptor.
 351  */
 352 uintptr_t
 353 ld_ent_setup(Ofl_desc *ofl, Xword segalign)
 354 {
 355         Ent_desc        *enp;
 356         predef_seg_t    *psegs;
 357         Sg_desc         *sgp;
 358         size_t          idx;

 360         /*
 361          * Initialize the elf library.
 362          */
 363         if (elf_version(EV_CURRENT) == EV_NONE) {
 364                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ELF_LIBELF),
 365                     EV_CURRENT);
 366                 return (S_ERROR);
 367         }

 369         /*
 370          * Initialize internal Global Symbol Table AVL tree
 371          */
 372         avl_create(&ofl->ofl_symavl, &ld_sym_avl_comp, sizeof (Sym_avlnode),
 373             SGSOFFSETOF(Sym_avlnode, sav_node));

 375         /* Initialize segment AVL tree */
 376         avl_create(&ofl->ofl_segs_avl, ofl_segs_avl_cmp,
 377             sizeof (Sg_desc), SGSOFFSETOF(Sg_desc, sg_avlnode));

 379         /* Initialize entrance criteria AVL tree */
 380         avl_create(&ofl->ofl_ents_avl, ofl_ents_avl_cmp, sizeof (Ent_desc),
 381             SGSOFFSETOF(Ent_desc, ec_avlnode));


 384         /*
 385          * Allocate and initialize writable copies of both the entrance and
 386          * segment descriptors.
 387          *
 388          * Note that on non-amd64 targets, this allocates a few more
 389          * elements than are needed. For now, we are willing to overallocate
 390          * a small amount to simplify the code.
 391          */
 392         if ((psegs = libld_malloc(sizeof (sg_desc))) == NULL)
 393                 return (S_ERROR);
 394         (void) memcpy(psegs, &sg_desc, sizeof (sg_desc));
 395         sgp = (Sg_desc *) psegs;

 397         /*
 398          * The data segment and stack permissions can differ:
 399          *
 400          *      - Architectural/ABI per-platform differences
 401          *      - Whether the object is built statically or dynamically
 402          *
 403          * Those segments so affected have their program header flags
 404          * set here at runtime, rather than in the sg_desc templates above.
 405          */
```



```
 406         psegs->psg_data.sg_phdr.p_flags = ld_targ.t_m.m_dataseg_perm;
 407         psegs->psg_bss.sg_phdr.p_flags = ld_targ.t_m.m_dataseg_perm;
 408         psegs->psg_dynamic.sg_phdr.p_flags = ld_targ.t_m.m_dataseg_perm;
 409         psegs->psg_sunwdtrace.sg_phdr.p_flags = ld_targ.t_m.m_dataseg_perm;
 410 #if     defined(_ELF64)
 411         psegs->psg_ldata.sg_phdr.p_flags = ld_targ.t_m.m_dataseg_perm;
 412         psegs->psg_sunwdtrace.sg_phdr.p_flags |= PF_X;
 413 #endif
 414         psegs->psg_sunwstack.sg_phdr.p_flags = ld_targ.t_m.m_stack_perm;
 415         if ((ofl->ofl_flags & FLG_OF_DYNAMIC) == 0)
 416                 psegs->psg_data.sg_phdr.p_flags |= PF_X;

 418         /*
 419          * Traverse the new entrance descriptor list converting the segment
 420          * pointer entries to the absolute address within the new segment
 421          * descriptor list.  Add each entrance descriptor to the output file
 422          * list.
 423          */
 424         if ((enp = libld_malloc(sizeof (ent_desc))) == NULL)
 425                 return (S_ERROR);
 426         (void) memcpy(enp, ent_desc, sizeof (ent_desc));
 427         for (idx = 0; idx < (sizeof (ent_desc) / sizeof (ent_desc[0])); idx++,
 428             enp++) {

 430 #if     defined(_ELF64)
 431                 /* Don't use the amd64 entry conditions for non-amd64 targets */
 432                 if ((enp->ec_attrmask & SHF_AMD64_LARGE) &&
 433                     (ld_targ.t_m.m_mach != EM_AMD64))
 434                         continue;
 435 #endif
 436                 if (aplist_append(&ofl->ofl_ents, enp,
 437                     AL_CNT_OFL_ENTRANCE) == NULL)
 438                         return (S_ERROR);

 440                 /*
 441                  * The segment pointer is currently pointing at a template
 442                  * segment descriptor in sg_desc. Compute its array index,
 443                  * and then use that index to compute the address of the
 444                  * corresponding descriptor in the writable copy.
 445                  */
 446                 enp->ec_segment =
 447                     &sgp[(enp->ec_segment - (Sg_desc *) &sg_desc)];
 448         }

 450         /*
 451          * Add each segment descriptor to the segment descriptor list. The
 452          * ones with non-NULL sg_name are also entered into the AVL tree.
 453          * For each loadable segment initialize a default alignment. Note
 454          * that ld(1) and ld.so.1 initialize this differently.
 455          */
 456         for (idx = 0; idx < predef_seg_nelts; idx++, sgp++) {
 457                 Phdr    *phdr = &(sgp->sg_phdr);

 459 #if     defined(_ELF64)
 460                 /* Ignore amd64 segment templates for non-amd64 targets */
 461                 switch (sgp->sg_id) {
 462                 case SGID_LRODATA:
 463                 case SGID_LDATA:
 464                         if ((ld_targ.t_m.m_mach != EM_AMD64))
 465                                 continue;
 466                 }
 467 #endif
 468                 if (phdr->p_type == PT_LOAD)
 469                         phdr->p_align = segalign;

 471                 if ((aplist_append(&ofl->ofl_segs, sgp,
```

```
472                     AL_CNT_SEGMENTS)) == NULL)
473                         return (S_ERROR);

475 #ifndef NDEBUG                    /* assert() is enabled */
475 #ifdef NDEBUG                     /* assert() is enabled */
476                 /*
477                  * Enforce the segment name rule: Any segment that can
478                  * be referenced by an entrance descriptor must have
479                  * a name. Any segment that cannot, must have a NULL
480                  * name pointer.
481                  */
482                 switch (phdr->p_type) {
483                 case PT_LOAD:
484                 case PT_NOTE:
485                 case PT_NULL:
486                         assert(sgp->sg_name != NULL);
487                         break;
488                 default:
489                         assert(sgp->sg_name == NULL);
490                         break;
491                 }
492 #endif

494                 /*
495                  * Add named segment descriptors to the AVL tree to
496                  * provide O(logN) lookups.
497                  */
498                 if (sgp->sg_name != NULL)
499                         avl_add(&ofl->ofl_segs_avl, sgp);
500         }

502         return (1);
503 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   46912 Wed May 22 03:21:44 2019
new/usr/src/cmd/sgs/libld/common/machrel.amd.c
11057 hidden undefined weak symbols should not leave relocations
11058 libld entrance descriptor assertions get NDEBUG check backwards
**********************************************************
______unchanged_portion_omitted_

 280 static uintptr_t
 281 ld_perform_outreloc(Rel_desc * orsp, Ofl_desc * ofl, Boolean *remain_seen)
 282 {
 283         Os_desc *       relosp, * osp = 0;
 284         Word            ndx;
 285         Xword           roffset, value;
 286         Sxword          raddend;
 287         Rela            rea;
 288         char            *relbits;
 289         Sym_desc *      sdp, * psym = (Sym_desc *)0;
 290         int             sectmoved = 0;

 292         raddend = orsp->rel_raddend;
 293         sdp = orsp->rel_sym;

 295         /*
 296          * If the section this relocation is against has been discarded
 297          * (-zignore), then also discard (skip) the relocation itself.
 298          */
 299         if (orsp->rel_isdesc && ((orsp->rel_flags &
 300             (FLG_REL_GOT | FLG_REL_BSS | FLG_REL_PLT | FLG_REL_NOINFO)) == 0) &&
 301             (orsp->rel_isdesc->is_flags & FLG_IS_DISCARD)) {
 302                 DBG_CALL(Dbg_reloc_discard(ofl->ofl_lml, M_MACH, orsp));
 303                 return (1);
 304         }

 306         /*
 307          * If this is a relocation against a move table, or expanded move
 308          * table, adjust the relocation entries.
 309          */
 310         if (RELAUX_GET_MOVE(orsp))
 311                 ld_adj_movereloc(ofl, orsp);

 313         /*
 314          * If this is a relocation against a section then we need to adjust the
 315          * raddend field to compensate for the new position of the input section
 316          * within the new output section.
 317          */
 318         if (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION) {
 319                 if (ofl->ofl_parsyms &&
 320                     (sdp->sd_isc->is_flags & FLG_IS_RELUPD) &&
 321                     /* LINTED */
 322                     (psym = ld_am_I_partial(orsp, orsp->rel_raddend))) {
 323                         DBG_CALL(Dbg_move_outsctadj(ofl->ofl_lml, psym));
 324                         sectmoved = 1;
 325                         if (ofl->ofl_flags & FLG_OF_RELOBJ)
 326                                 raddend = psym->sd_sym->st_value;
 327                         else
 328                                 raddend = psym->sd_sym->st_value -
 329                                     psym->sd_isc->is_osdesc->os_shdr->sh_addr;
 330                         /* LINTED */
 331                         raddend += (Off)_elf_getxoff(psym->sd_isc->is_indata);
 332                         if (psym->sd_isc->is_shdr->sh_flags & SHF_ALLOC)
 333                                 raddend +=
 334                                     psym->sd_isc->is_osdesc->os_shdr->sh_addr;
 335                 } else {
 336                         /* LINTED */
 337                         raddend += (Off)_elf_getxoff(sdp->sd_isc->is_indata);
```



```
 338                         if (sdp->sd_isc->is_shdr->sh_flags & SHF_ALLOC)
 339                                 raddend +=
 340                                     sdp->sd_isc->is_osdesc->os_shdr->sh_addr;
 341                 }
 342         }

 344         value = sdp->sd_sym->st_value;

 346         if (orsp->rel_flags & FLG_REL_GOT) {
 347                 /*
 348                  * Note: for GOT relative relocations on amd64
 349                  *       we discard the addend.  It was relevant
 350                  *       to the reference - not to the data item
 351                  *       being referenced (ie: that -4 thing).
 352                  */
 353                 raddend = 0;
 354                 osp = ofl->ofl_osgot;
 355                 roffset = ld_calc_got_offset(orsp, ofl);

 357         } else if (orsp->rel_flags & FLG_REL_PLT) {
 358                 /*
 359                  * Note that relocations for PLT's actually
 360                  * cause a relocation againt the GOT.
 361                  */
 362                 osp = ofl->ofl_osplt;
 363                 roffset = (ofl->ofl_osgot->os_shdr->sh_addr) +
 364                     sdp->sd_aux->sa_PLTGOTndx * M_GOT_ENTSIZE;
 365                 raddend = 0;
 366                 if (plt_entry(ofl, sdp) == S_ERROR)
 367                         return (S_ERROR);

 369         } else if (orsp->rel_flags & FLG_REL_BSS) {
 370                 /*
 371                  * This must be a R_AMD64_COPY.  For these set the roffset to
 372                  * point to the new symbols location.
 373                  */
 374                 osp = ofl->ofl_isbss->is_osdesc;
 375                 roffset = value;

 377                 /*
 378                  * The raddend doesn't mean anything in a R_SPARC_COPY
 379                  * relocation.  Null it out because it can confuse people.
 380                  */
 381                 raddend = 0;
 382         } else {
 383                 osp = RELAUX_GET_OSDESC(orsp);

 385                 /*
 386                  * Calculate virtual offset of reference point; equals offset
 387                  * into section + vaddr of section for loadable sections, or
 388                  * offset plus section displacement for nonloadable sections.
 389                  */
 390                 roffset = orsp->rel_roffset +
 391                     (Off)_elf_getxoff(orsp->rel_isdesc->is_indata);
 392                 if (!(ofl->ofl_flags & FLG_OF_RELOBJ))
 393                         roffset += orsp->rel_isdesc->is_osdesc->
 394                             os_shdr->sh_addr;
 395         }

 397         if ((osp == 0) || ((relosp = osp->os_relosdesc) == 0))
 398                 relosp = ofl->ofl_osrel;

 400         /*
 401          * Assign the symbols index for the output relocation.  If the
 402          * relocation refers to a SECTION symbol then it's index is based upon
 403          * the output sections symbols index.  Otherwise the index can be
```

```
404          * derived from the symbols index itself.
405          */
406         if (orsp->rel_rtype == R_AMD64_RELATIVE)
407                 ndx = STN_UNDEF;
408         else if ((orsp->rel_flags & FLG_REL_SCNNDX) ||
409             (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION)) {
410                 if (sectmoved == 0) {
411                         /*
412                          * Check for a null input section. This can
413                          * occur if this relocation references a symbol
414                          * generated by sym_add_sym().
415                          */
416                         if (sdp->sd_isc && sdp->sd_isc->is_osdesc)
417                                 ndx = sdp->sd_isc->is_osdesc->os_identndx;
418                         else
419                                 ndx = sdp->sd_shndx;
420                 } else
421                         ndx = ofl->ofl_parexpnndx;
422         } else
423                 ndx = sdp->sd_symndx;

425         /*
426          * Add the symbols 'value' to the addend field.
427          */
428         if (orsp->rel_flags & FLG_REL_ADVAL)
429                 raddend += value;

431         /*
432          * The addend field for R_AMD64_DTPMOD64 means nothing.  The addend
433          * is propagated in the corresponding R_AMD64_DTPOFF64 relocation.
434          */
435         if (orsp->rel_rtype == R_AMD64_DTPMOD64)
436                 raddend = 0;

438         if ((orsp->rel_rtype != M_R_NONE) &&
439             (orsp->rel_rtype != M_R_RELATIVE)) {
440                 if (ndx == 0) {
441                         Conv_inv_buf_t  inv_buf;
442                         Is_desc *isp = orsp->rel_isdesc;

444                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_NOSYMBOL),
445                             conv_reloc_type(ofl->ofl_nehdr->e_machine,
446                             orsp->rel_rtype, 0, &inv_buf),
447                             isp->is_file->ifl_name, EC_WORD(isp->is_scnndx),
448                             isp->is_name, EC_XWORD(roffset));
449                         return (S_ERROR);
450                 }
451         }
438         relbits = (char *)relosp->os_outdata->d_buf;

453         rea.r_info = ELF_R_INFO(ndx, orsp->rel_rtype);
454         rea.r_offset = roffset;
455         rea.r_addend = raddend;
456         DBG_CALL(Dbg_reloc_out(ofl, ELF_DBG_LD, SHT_RELA, &rea, relosp->os_name,
457             ld_reloc_sym_name(orsp)));

459         /*
460          * Assert we haven't walked off the end of our relocation table.
461          */
462         assert(relosp->os_szoutrels <= relosp->os_shdr->sh_size);

464         relbits = (char *)relosp->os_outdata->d_buf;

466 #endif /* ! codereview */
467         (void) memcpy((relbits + relosp->os_szoutrels),
468             (char *)&rea, sizeof (Rela));
```



```
469         relosp->os_szoutrels += (Xword)sizeof (Rela);

471         /*
472          * Determine if this relocation is against a non-writable, allocatable
473          * section.  If so we may need to provide a text relocation diagnostic.
474          * Note that relocations against the .plt (R_AMD64_JUMP_SLOT) actually
475          * result in modifications to the .got.
476          */
477         if (orsp->rel_rtype == R_AMD64_JUMP_SLOT)
478                 osp = ofl->ofl_osgot;

480         ld_reloc_remain_entry(orsp, osp, ofl, remain_seen);
481         return (1);
482 }

484 /*
485  * amd64 Instructions for TLS processing
486  */
487 static uchar_t tlsinstr_gd_ie[] = {
488         /*
489          *      0x00 movq %fs:0, %rax
490          */
491         0x64, 0x48, 0x8b, 0x04, 0x25,
492         0x00, 0x00, 0x00, 0x00,
493         /*
494          *      0x09 addq x@gottpoff(%rip), %rax
495          */
496         0x48, 0x03, 0x05, 0x00, 0x00,
497         0x00, 0x00
498 };

500 static uchar_t tlsinstr_gd_le[] = {
501         /*
502          *      0x00 movq %fs:0, %rax
503          */
504         0x64, 0x48, 0x8b, 0x04, 0x25,
505         0x00, 0x00, 0x00, 0x00,
506         /*
507          *      0x09 leaq x@gottpoff(%rip), %rax
508          */
509         0x48, 0x8d, 0x80, 0x00, 0x00,
510         0x00, 0x00
511 };

513 static uchar_t tlsinstr_ld_le[] = {
514         /*
515          * .byte 0x66
516          */
517         0x66,
518         /*
519          * .byte 0x66
520          */
521         0x66,
522         /*
523          * .byte 0x66
524          */
525         0x66,
526         /*
527          * movq %fs:0, %rax
528          */
529         0x64, 0x48, 0x8b, 0x04, 0x25,
530         0x00, 0x00, 0x00, 0x00
531 };

533 #define REX_B           0x1
534 #define REX_X           0x2
```

```
#define REX_R           0x4
#define REX_W           0x8
#define REX_PREFIX      0x40

#define REX_RW          (REX_PREFIX | REX_R | REX_W)
#define REX_BW          (REX_PREFIX | REX_B | REX_W)
#define REX_BRW         (REX_PREFIX | REX_B | REX_R | REX_W)

#define REG_ESP         0x4

#define INSN_ADDMR      0x03    /* addq mem,reg */
#define INSN_ADDIR      0x81    /* addq imm,reg */
#define INSN_MOVMR      0x8b    /* movq mem,reg */
#define INSN_MOVIR      0xc7    /* movq imm,reg */
#define INSN_LEA        0x8d    /* leaq mem,reg */

static Fixupret
tls_fixups(Ofl_desc *ofl, Rel_desc *arsp)
{
        Sym_desc        *sdp = arsp->rel_sym;
        Word            rtype = arsp->rel_rtype;
        uchar_t         *offset;

        offset = (uchar_t *)((uintptr_t)arsp->rel_roffset +
            (uintptr_t)_elf_getxoff(arsp->rel_isdesc->is_indata) +
            (uintptr_t)RELAUX_GET_OSDESC(arsp)->os_outdata->d_buf);

        /*
         * Note that in certain of the original insn sequences below, the
         * instructions are not necessarily adjacent
         */
        if (sdp->sd_ref == REF_DYN_NEED) {
                /*
                 * IE reference model
                 */
                switch (rtype) {
                case R_AMD64_TLSGD:
                        /*
                         *  GD -> IE
                         *
                         * Transition:
                         *      0x00 .byte 0x66
                         *      0x01 leaq x@tlsgd(%rip), %rdi
                         *      0x08 .word 0x6666
                         *      0x0a rex64
                         *      0x0b call __tls_get_addr@plt
                         *      0x10
                         * To:
                         *      0x00 movq %fs:0, %rax
                         *      0x09 addq x@gottpoff(%rip), %rax
                         *      0x10
                         */
                        DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                            R_AMD64_GOTTPOFF, arsp, ld_reloc_sym_name));
                        arsp->rel_rtype = R_AMD64_GOTTPOFF;
                        arsp->rel_roffset += 8;
                        arsp->rel_raddend = (Sxword)-4;

                        /*
                         * Adjust 'offset' to beginning of instruction
                         * sequence.
                         */
                        offset -= 4;
                        (void) memcpy(offset, tlsinstr_gd_ie,
                            sizeof (tlsinstr_gd_ie));
                        return (FIX_RELOC);
```



```
                case R_AMD64_PLT32:
                        /*
                         * Fixup done via the TLS_GD relocation.
                         */
                        DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                            R_AMD64_NONE, arsp, ld_reloc_sym_name));
                        return (FIX_DONE);
                }
        }

        /*
         * LE reference model
         */
        switch (rtype) {
        case R_AMD64_TLSGD:
                /*
                 * GD -> LE
                 *
                 * Transition:
                 *      0x00 .byte 0x66
                 *      0x01 leaq x@tlsgd(%rip), %rdi
                 *      0x08 .word 0x6666
                 *      0x0a rex64
                 *      0x0b call __tls_get_addr@plt
                 *      0x10
                 * To:
                 *      0x00 movq %fs:0, %rax
                 *      0x09 leaq x@tpoff(%rax), %rax
                 *      0x10
                 */
                DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                    R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
                arsp->rel_rtype = R_AMD64_TPOFF32;
                arsp->rel_roffset += 8;
                arsp->rel_raddend = 0;

                /*
                 * Adjust 'offset' to beginning of instruction sequence.
                 */
                offset -= 4;
                (void) memcpy(offset, tlsinstr_gd_le, sizeof (tlsinstr_gd_le));
                return (FIX_RELOC);

        case R_AMD64_GOTTPOFF: {
                /*
                 * IE -> LE
                 *
                 * Transition 1:
                 *      movq %fs:0, %reg
                 *      addq x@gottpoff(%rip), %reg
                 * To:
                 *      movq %fs:0, %reg
                 *      leaq x@tpoff(%reg), %reg
                 *
                 * Transition (as a special case):
                 *      movq %fs:0, %r12/%rsp
                 *      addq x@gottpoff(%rip), %r12/%rsp
                 * To:
                 *      movq %fs:0, %r12/%rsp
                 *      addq x@tpoff(%rax), %r12/%rsp
                 *
                 * Transition 2:
                 *      movq x@gottpoff(%rip), %reg
                 *      movq %fs:(%reg), %reg
                 * To:
```

```
667                  *      movq x@tpoff(%reg), %reg
668                  *      movq %fs:(%reg), %reg
669                  */
670                 Conv_inv_buf_t  inv_buf;
671                 uint8_t reg;            /* Register */

673                 offset -= 3;

675                 reg = offset[2] >> 3; /* Encoded dest. reg. operand */

677                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
678                     R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
679                 arsp->rel_rtype = R_AMD64_TPOFF32;
680                 arsp->rel_raddend = 0;

682                 /*
683                  * This is transition 2, and the special case of form 1 where
684                  * a normal transition would index %rsp or %r12 and need a SIB
685                  * byte in the leaq for which we lack space
686                  */
687                 if ((offset[1] == INSN_MOVMR) ||
688                     ((offset[1] == INSN_ADDMR) && (reg == REG_ESP))) {
689                         /*
690                          * If we needed an extra bit of MOD.reg to refer to
691                          * this register as the dest of the original movq we
692                          * need an extra bit of MOD.rm to refer to it in the
693                          * dest of the replacement movq or addq.
694                          */
695                         if (offset[0] == REX_RW)
696                                 offset[0] = REX_BW;

698                         offset[1] = (offset[1] == INSN_MOVMR) ?
699                             INSN_MOVIR : INSN_ADDIR;
700                         offset[2] = 0xc0 | reg;

702                         return (FIX_RELOC);
703                 } else if (offset[1] == INSN_ADDMR) {
704                         /*
705                          * If we needed an extra bit of MOD.reg to refer to
706                          * this register in the dest of the addq we need an
707                          * extra bit of both MOD.reg and MOD.rm to refer to it
708                          * in the source and dest of the leaq
709                          */
710                         if (offset[0] == REX_RW)
711                                 offset[0] = REX_BRW;

713                         offset[1] = INSN_LEA;
714                         offset[2] = 0x80 | (reg << 3) | reg;

716                         return (FIX_RELOC);
717                 }

719                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
720                     conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
721                     arsp->rel_isdesc->is_file->ifl_name,
722                     ld_reloc_sym_name(arsp),
723                     arsp->rel_isdesc->is_name,
724                     EC_OFF(arsp->rel_roffset));
725                 return (FIX_ERROR);
726         }
727         case R_AMD64_TLSLD:
728                 /*
729                  * LD -> LE
730                  *
731                  * Transition
732                  *      0x00 leaq x1@tlsgd(%rip), %rdi
```



```
733                  *      0x07 call __tls_get_addr@plt
734                  *      0x0c
735                  * To:
736                  *      0x00 .byte 0x66
737                  *      0x01 .byte 0x66
738                  *      0x02 .byte 0x66
739                  *      0x03 movq %fs:0, %rax
740                  */
741                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
742                     R_AMD64_NONE, arsp, ld_reloc_sym_name));
743                 offset -= 3;
744                 (void) memcpy(offset, tlsinstr_ld_le, sizeof (tlsinstr_ld_le));
745                 return (FIX_DONE);

747         case R_AMD64_DTPOFF32:
748                 /*
749                  * LD->LE
750                  *
751                  * Transition:
752                  *      0x00 leaq x1@dtpoff(%rax), %rcx
753                  * To:
754                  *      0x00 leaq x1@tpoff(%rax), %rcx
755                  */
756                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
757                     R_AMD64_TPOFF32, arsp, ld_reloc_sym_name));
758                 arsp->rel_rtype = R_AMD64_TPOFF32;
759                 return (FIX_RELOC);
760         }

762         return (FIX_RELOC);
763 }

765 static uintptr_t
766 ld_do_activerelocs(Ofl_desc *ofl)
767 {
768         Rel_desc        *arsp;
769         Rel_cachebuf    *rcbp;
770         Aliste          idx;
771         uintptr_t       return_code = 1;
772         ofl_flag_t      flags = ofl->ofl_flags;

774         if (aplist_nitems(ofl->ofl_actrels.rc_list) != 0)
775                 DBG_CALL(Dbg_reloc_doact_title(ofl->ofl_lml));

777         /*
778          * Process active relocations.
779          */
780         REL_CACHE_TRAVERSE(&ofl->ofl_actrels, idx, rcbp, arsp) {
781                 uchar_t         *addr;
782                 Xword           value;
783                 Sym_desc        *sdp;
784                 const char      *ifl_name;
785                 Xword           refaddr;
786                 int             moved = 0;
787                 Gotref          gref;
788                 Os_desc         *osp;

790                 /*
791                  * If the section this relocation is against has been discarded
792                  * (-zignore), then discard (skip) the relocation itself.
793                  */
794                 if ((arsp->rel_isdesc->is_flags & FLG_IS_DISCARD) &&
795                     ((arsp->rel_flags & (FLG_REL_GOT | FLG_REL_BSS |
796                     FLG_REL_PLT | FLG_REL_NOINFO)) == 0)) {
797                         DBG_CALL(Dbg_reloc_discard(ofl->ofl_lml, M_MACH, arsp));
798                         continue;
```

```
799                 }
801                 /*
802                  * We determine what the 'got reference' model (if required)
803                  * is at this point.  This needs to be done before tls_fixup()
804                  * since it may 'transition' our instructions.
805                  *
806                  * The got table entries have already been assigned,
807                  * and we bind to those initial entries.
808                  */
809                 if (arsp->rel_flags & FLG_REL_DTLS)
810                         gref = GOT_REF_TLSGD;
811                 else if (arsp->rel_flags & FLG_REL_MTLS)
812                         gref = GOT_REF_TLSLD;
813                 else if (arsp->rel_flags & FLG_REL_STLS)
814                         gref = GOT_REF_TLSIE;
815                 else
816                         gref = GOT_REF_GENERIC;
818                 /*
819                  * Perform any required TLS fixups.
820                  */
821                 if (arsp->rel_flags & FLG_REL_TLSFIX) {
822                         Fixupret        ret;
824                         if ((ret = tls_fixups(ofl, arsp)) == FIX_ERROR)
825                                 return (S_ERROR);
826                         if (ret == FIX_DONE)
827                                 continue;
828                 }
830                 /*
831                  * If this is a relocation against a move table, or
832                  * expanded move table, adjust the relocation entries.
833                  */
834                 if (RELAUX_GET_MOVE(arsp))
835                         ld_adj_movereloc(ofl, arsp);
837                 sdp = arsp->rel_sym;
838                 refaddr = arsp->rel_roffset +
839                     (Off)_elf_getxoff(arsp->rel_isdesc->is_indata);
841                 if ((arsp->rel_flags & FLG_REL_CLVAL) ||
842                     (arsp->rel_flags & FLG_REL_GOTCL))
843                         value = 0;
844                 else if (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION) {
845                         Sym_desc        *sym;
847                         /*
848                          * The value for a symbol pointing to a SECTION
849                          * is based off of that sections position.
850                          */
851                         if ((sdp->sd_isc->is_flags & FLG_IS_RELUPD) &&
852                             /* LINTED */
853                             (sym = ld_am_I_partial(arsp, arsp->rel_raddend))) {
854                                 /*
855                                  * The symbol was moved, so adjust the value
856                                  * relative to the new section.
857                                  */
858                                 value = sym->sd_sym->st_value;
859                                 moved = 1;
861                                 /*
862                                  * The original raddend covers the displacement
863                                  * from the section start to the desired
864                                  * address. The value computed above gets us
```



```
865                                  * from the section start to the start of the
866                                  * symbol range. Adjust the old raddend to
867                                  * remove the offset from section start to
868                                  * symbol start, leaving the displacement
869                                  * within the range of the symbol.
870                                  */
871                                 arsp->rel_raddend -= sym->sd_osym->st_value;
872                         } else {
873                                 value = _elf_getxoff(sdp->sd_isc->is_indata);
874                                 if (sdp->sd_isc->is_shdr->sh_flags & SHF_ALLOC)
875                                         value += sdp->sd_isc->is_osdesc->
876                                             os_shdr->sh_addr;
877                         }
878                         if (sdp->sd_isc->is_shdr->sh_flags & SHF_TLS)
879                                 value -= ofl->ofl_tlsphdr->p_vaddr;
881                 } else if (IS_SIZE(arsp->rel_rtype)) {
882                         /*
883                          * Size relocations require the symbols size.
884                          */
885                         value = sdp->sd_sym->st_size;
887                 } else if ((sdp->sd_flags & FLG_SY_CAP) &&
888                     sdp->sd_aux && sdp->sd_aux->sa_PLTndx) {
889                         /*
890                          * If relocation is against a capabilities symbol, we
891                          * need to jump to an associated PLT, so that at runtime
892                          * ld.so.1 is involved to determine the best binding
893                          * choice. Otherwise, the value is the symbols value.
894                          */
895                         value = ld_calc_plt_addr(sdp, ofl);
896                 } else
897                         value = sdp->sd_sym->st_value;
899                 /*
900                  * Relocation against the GLOBAL_OFFSET_TABLE.
901                  */
902                 if ((arsp->rel_flags & FLG_REL_GOT) &&
903                     !ld_reloc_set_aux_osdesc(ofl, arsp, ofl->ofl_osgot))
904                         return (S_ERROR);
905                 osp = RELAUX_GET_OSDESC(arsp);
907                 /*
908                  * If loadable and not producing a relocatable object add the
909                  * sections virtual address to the reference address.
910                  */
911                 if ((arsp->rel_flags & FLG_REL_LOAD) &&
912                     ((flags & FLG_OF_RELOBJ) == 0))
913                         refaddr += arsp->rel_isdesc->is_osdesc->
914                             os_shdr->sh_addr;
916                 /*
917                  * If this entry has a PLT assigned to it, its value is actually
918                  * the address of the PLT (and not the address of the function).
919                  */
920                 if (IS_PLT(arsp->rel_rtype)) {
921                         if (sdp->sd_aux && sdp->sd_aux->sa_PLTndx)
922                                 value = ld_calc_plt_addr(sdp, ofl);
923                 }
925                 /*
926                  * Add relocations addend to value.  Add extra
927                  * relocation addend if needed.
928                  *
929                  * Note: For GOT relative relocations on amd64 we discard the
930                  * addend.  It was relevant to the reference - not to the
```

```
931                  * data item being referenced (ie: that -4 thing).
932                  */
933                 if ((arsp->rel_flags & FLG_REL_GOT) == 0)
934                         value += arsp->rel_raddend;
935
936                 /*
937                  * Determine whether the value needs further adjustment. Filter
938                  * through the attributes of the relocation to determine what
939                  * adjustment is required.  Note, many of the following cases
940                  * are only applicable when a .got is present.  As a .got is
941                  * not generated when a relocatable object is being built,
942                  * any adjustments that require a .got need to be skipped.
943                  */
944                 if ((arsp->rel_flags & FLG_REL_GOT) &&
945                     ((flags & FLG_OF_RELOBJ) == 0)) {
946                         Xword           R1addr;
947                         uintptr_t       R2addr;
948                         Word            gotndx;
949                         Gotndx          *gnp;
950
951                         /*
952                          * Perform relocation against GOT table. Since this
953                          * doesn't fit exactly into a relocation we place the
954                          * appropriate byte in the GOT directly
955                          *
956                          * Calculate offset into GOT at which to apply
957                          * the relocation.
958                          */
959                         gnp = ld_find_got_ndx(sdp->sd_GOTndxs, gref, ofl, arsp);
960                         assert(gnp);
961
962                         if (arsp->rel_rtype == R_AMD64_DTPOFF64)
963                                 gotndx = gnp->gn_gotndx + 1;
964                         else
965                                 gotndx = gnp->gn_gotndx;
966
967                         R1addr = (Xword)(gotndx * M_GOT_ENTSIZE);
968
969                         /*
970                          * Add the GOTs data's offset.
971                          */
972                         R2addr = R1addr + (uintptr_t)osp->os_outdata->d_buf;
973
974                         DBG_CALL(Dbg_reloc_doact(ofl->ofl_lml, ELF_DBG_LD_ACT,
975                             M_MACH, SHT_RELA, arsp, R1addr, value,
976                             ld_reloc_sym_name));
977
978                         /*
979                          * And do it.
980                          */
981                         if (ofl->ofl_flags1 & FLG_OF1_ENCDIFF)
982                                 *(Xword *)R2addr = ld_bswap_Xword(value);
983                         else
984                                 *(Xword *)R2addr = value;
985                         continue;
986
987                 } else if (IS_GOT_BASED(arsp->rel_rtype) &&
988                     ((flags & FLG_OF_RELOBJ) == 0)) {
989                         value -= ofl->ofl_osgot->os_shdr->sh_addr;
990
991                 } else if (IS_GOTPCREL(arsp->rel_rtype) &&
992                     ((flags & FLG_OF_RELOBJ) == 0)) {
993                         Gotndx *gnp;
994
995                         /*
996                          * Calculation:
```



```
997                          *      G + GOT + A - P
998                          */
999                         gnp = ld_find_got_ndx(sdp->sd_GOTndxs, gref, ofl, arsp);
1000                        assert(gnp);
1001                        value = (Xword)(ofl->ofl_osgot->os_shdr-> sh_addr) +
1002                            ((Xword)gnp->gn_gotndx * M_GOT_ENTSIZE) +
1003                            arsp->rel_raddend - refaddr;
1004
1005                } else if (IS_GOT_PC(arsp->rel_rtype) &&
1006                    ((flags & FLG_OF_RELOBJ) == 0)) {
1007                        value = (Xword)(ofl->ofl_osgot->os_shdr->
1008                            sh_addr) - refaddr + arsp->rel_raddend;
1009
1010                } else if ((IS_PC_RELATIVE(arsp->rel_rtype)) &&
1011                    (((flags & FLG_OF_RELOBJ) == 0) ||
1012                    (osp == sdp->sd_isc->is_osdesc))) {
1013                        value -= refaddr;
1014
1015                } else if (IS_TLS_INS(arsp->rel_rtype) &&
1016                    IS_GOT_RELATIVE(arsp->rel_rtype) &&
1017                    ((flags & FLG_OF_RELOBJ) == 0)) {
1018                        Gotndx  *gnp;
1019
1020                        gnp = ld_find_got_ndx(sdp->sd_GOTndxs, gref, ofl, arsp);
1021                        assert(gnp);
1022                        value = (Xword)gnp->gn_gotndx * M_GOT_ENTSIZE;
1023
1024                } else if (IS_GOT_RELATIVE(arsp->rel_rtype) &&
1025                    ((flags & FLG_OF_RELOBJ) == 0)) {
1026                        Gotndx *gnp;
1027
1028                        gnp = ld_find_got_ndx(sdp->sd_GOTndxs, gref, ofl, arsp);
1029                        assert(gnp);
1030                        value = (Xword)gnp->gn_gotndx * M_GOT_ENTSIZE;
1031
1032                } else if ((arsp->rel_flags & FLG_REL_STLS) &&
1033                    ((flags & FLG_OF_RELOBJ) == 0)) {
1034                        Xword   tlsstatsize;
1035
1036                        /*
1037                         * This is the LE TLS reference model.  Static
1038                         * offset is hard-coded.
1039                         */
1040                        tlsstatsize = S_ROUND(ofl->ofl_tlsphdr->p_memsz,
1041                            M_TLSSTATALIGN);
1042                        value = tlsstatsize - value;
1043
1044                        /*
1045                         * Since this code is fixed up, it assumes a negative
1046                         * offset that can be added to the thread pointer.
1047                         */
1048                        if (arsp->rel_rtype == R_AMD64_TPOFF32)
1049                                value = -value;
1050                }
1051
1052                if (arsp->rel_isdesc->is_file)
1053                        ifl_name = arsp->rel_isdesc->is_file->ifl_name;
1054                else
1055                        ifl_name = MSG_INTL(MSG_STR_NULL);
1056
1057                /*
1058                 * Make sure we have data to relocate.  Compiler and assembler
1059                 * developers have been known to generate relocations against
1060                 * invalid sections (normally .bss), so for their benefit give
1061                 * them sufficient information to help analyze the problem.
1062                 * End users should never see this.
```

```
1063                 */
1064                if (arsp->rel_isdesc->is_indata->d_buf == 0) {
1065                        Conv_inv_buf_t inv_buf;
1066
1067                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_EMPTYSEC),
1068                            conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
1069                            ifl_name, ld_reloc_sym_name(arsp),
1070                            EC_WORD(arsp->rel_isdesc->is_scnndx),
1071                            arsp->rel_isdesc->is_name);
1072                        return (S_ERROR);
1073                }
1074
1075                /*
1076                 * Get the address of the data item we need to modify.
1077                 */
1078                addr = (uchar_t *)((uintptr_t)arsp->rel_roffset +
1079                    (uintptr_t)_elf_getxoff(arsp->rel_isdesc->is_indata));
1080
1081                DBG_CALL(Dbg_reloc_doact(ofl->ofl_lml, ELF_DBG_LD_ACT,
1082                    M_MACH, SHT_RELA, arsp, EC_NATPTR(addr), value,
1083                    ld_reloc_sym_name));
1084                addr += (uintptr_t)osp->os_outdata->d_buf;
1085
1086                if ((((uintptr_t)addr - (uintptr_t)ofl->ofl_nehdr) >
1087                    ofl->ofl_size) || (arsp->rel_roffset >
1088                    osp->os_shdr->sh_size)) {
1089                        int             class;
1090                        Conv_inv_buf_t inv_buf;
1091
1092                        if (((uintptr_t)addr - (uintptr_t)ofl->ofl_nehdr) >
1093                            ofl->ofl_size)
1094                                class = ERR_FATAL;
1095                        else
1096                                class = ERR_WARNING;
1097
1098                        ld_eprintf(ofl, class, MSG_INTL(MSG_REL_INVALOFFSET),
1099                            conv_reloc_amd64_type(arsp->rel_rtype, 0, &inv_buf),
1100                            ifl_name, EC_WORD(arsp->rel_isdesc->is_scnndx),
1101                            arsp->rel_isdesc->is_name, ld_reloc_sym_name(arsp),
1102                            EC_ADDR((uintptr_t)addr -
1103                            (uintptr_t)ofl->ofl_nehdr));
1104
1105                        if (class == ERR_FATAL) {
1106                                return_code = S_ERROR;
1107                                continue;
1108                        }
1109                }
1110
1111                /*
1112                 * The relocation is additive.  Ignore the previous symbol
1113                 * value if this local partial symbol is expanded.
1114                 */
1115                if (moved)
1116                        value -= *addr;
1117
1118                /*
1119                 * If '-z noreloc' is specified - skip the do_reloc_ld stage.
1120                 */
1121                if (OFL_DO_RELOC(ofl)) {
1122                        /*
1123                         * If this is a PROGBITS section and the running linker
1124                         * has a different byte order than the target host,
1125                         * tell do_reloc_ld() to swap bytes.
1126                         */
1127                        if (do_reloc_ld(arsp, addr, &value, ld_reloc_sym_name,
1128                            ifl_name, OFL_SWAP_RELOC_DATA(ofl, arsp),
```



```
1129                            ofl->ofl_lml) == 0) {
1130                                ofl->ofl_flags |= FLG_OF_FATAL;
1131                                return_code = S_ERROR;
1132                        }
1133                }
1134        }
1135        return (return_code);
1136 }
1137
1138 static uintptr_t
1139 ld_add_outrel(Word flags, Rel_desc *rsp, Ofl_desc *ofl)
1140 {
1141        Rel_desc        *orsp;
1142        Sym_desc        *sdp = rsp->rel_sym;
1143
1144        /*
1145         * Static executables *do not* want any relocations against them.
1146         * Since our engine still creates relocations against a WEAK UNDEFINED
1147         * symbol in a static executable, it's best to disable them here
1148         * instead of through out the relocation code.
1149         */
1150        if (OFL_IS_STATIC_EXEC(ofl))
1151                return (1);
1152
1153        /*
1154         * If the symbol will be reduced, we can't leave outstanding
1155         * relocations against it, as nothing will ever be able to satisfy them
1156         * (and the symbol won't be in .dynsym
1157         */
1158        if ((sdp != NULL) &&
1159            (sdp->sd_sym->st_shndx == SHN_UNDEF) &&
1160            (rsp->rel_rtype != M_R_NONE) &&
1161            (rsp->rel_rtype != M_R_RELATIVE)) {
1162                if (ld_sym_reducable(ofl, sdp))
1163                        return (1);
1164        }
1165 #endif /* ! codereview */
1166
1167        /*
1168         * If we are adding a output relocation against a section
1169         * symbol (non-RELATIVE) then mark that section.  These sections
1170         * will be added to the .dynsym symbol table.
1171         */
1172        if (sdp && (rsp->rel_rtype != M_R_RELATIVE) &&
1173            ((flags & FLG_REL_SCNNDX) ||
1174            (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION))) {
1175
1176                /*
1177                 * If this is a COMMON symbol - no output section
1178                 * exists yet - (it's created as part of sym_validate()).
1179                 * So - we mark here that when it's created it should
1180                 * be tagged with the FLG_OS_OUTREL flag.
1181                 */
1182                if ((sdp->sd_flags & FLG_SY_SPECSEC) &&
1183                    (sdp->sd_sym->st_shndx == SHN_COMMON)) {
1184                        if (ELF_ST_TYPE(sdp->sd_sym->st_info) != STT_TLS)
1185                                ofl->ofl_flags1 |= FLG_OF1_BSSOREL;
1186                        else
1187                                ofl->ofl_flags1 |= FLG_OF1_TLSOREL;
1188                } else {
1189                        Os_desc *osp;
1190                        Is_desc *isp = sdp->sd_isc;
1191
1192                        if (isp && ((osp = isp->is_osdesc) != NULL) &&
1193                            ((osp->os_flags & FLG_OS_OUTREL) == 0)) {
1194                                ofl->ofl_dynshdrcnt++;
```

```
                                osp->os_flags |= FLG_OS_OUTREL;
                        }
                }
        }

        /* Enter it into the output relocation cache */
        if ((orsp = ld_reloc_enter(ofl, &ofl->ofl_outrels, rsp, flags)) == NULL)
                return (S_ERROR);

        if (flags & FLG_REL_GOT)
                ofl->ofl_relocgotsz += (Xword)sizeof (Rela);
        else if (flags & FLG_REL_PLT)
                ofl->ofl_relocpltsz += (Xword)sizeof (Rela);
        else if (flags & FLG_REL_BSS)
                ofl->ofl_relocbsssz += (Xword)sizeof (Rela);
        else if (flags & FLG_REL_NOINFO)
                ofl->ofl_relocrelsz += (Xword)sizeof (Rela);
        else
                RELAUX_GET_OSDESC(orsp)->os_szoutrels += (Xword)sizeof (Rela);

        if (orsp->rel_rtype == M_R_RELATIVE)
                ofl->ofl_relocrelcnt++;

        /*
         * We don't perform sorting on PLT relocations because
         * they have already been assigned a PLT index and if we
         * were to sort them we would have to re-assign the plt indexes.
         */
        if (!(flags & FLG_REL_PLT))
                ofl->ofl_reloccnt++;

        /*
         * Insure a GLOBAL_OFFSET_TABLE is generated if required.
         */
        if (IS_GOT_REQUIRED(orsp->rel_rtype))
                ofl->ofl_flags |= FLG_OF_BLDGOT;

        /*
         * Identify and possibly warn of a displacement relocation.
         */
        if (orsp->rel_flags & FLG_REL_DISP) {
                ofl->ofl_dtflags_1 |= DF_1_DISPRELPND;

                if (ofl->ofl_flags & FLG_OF_VERBOSE)
                        ld_disp_errmsg(MSG_INTL(MSG_REL_DISPREL4), orsp, ofl);
        }
        DBG_CALL(Dbg_reloc_ors_entry(ofl->ofl_lml, ELF_DBG_LD, SHT_RELA,
            M_MACH, orsp));
        return (1);
}

/*
 * process relocation for a LOCAL symbol
 */
static uintptr_t
ld_reloc_local(Rel_desc * rsp, Ofl_desc * ofl)
{
        ofl_flag_t      flags = ofl->ofl_flags;
        Sym_desc        *sdp = rsp->rel_sym;
        Word            shndx = sdp->sd_sym->st_shndx;
        Word            ortype = rsp->rel_rtype;

        /*
         * if ((shared object) and (not pc relative relocation) and
         *    (not against ABS symbol))
         * then
```



```
         *      build R_AMD64_RELATIVE
         * fi
         */
        if ((flags & FLG_OF_SHAROBJ) && (rsp->rel_flags & FLG_REL_LOAD) &&
            !(IS_PC_RELATIVE(rsp->rel_rtype)) && !(IS_SIZE(rsp->rel_rtype)) &&
            !(IS_GOT_BASED(rsp->rel_rtype)) &&
            !(rsp->rel_isdesc != NULL &&
            (rsp->rel_isdesc->is_shdr->sh_type == SHT_SUNW_dof)) &&
            (((sdp->sd_flags & FLG_SY_SPECSEC) == 0) ||
            (shndx != SHN_ABS) || (sdp->sd_aux && sdp->sd_aux->sa_symspec))) {

                /*
                 * R_AMD64_RELATIVE updates a 64bit address, if this
                 * relocation isn't a 64bit binding then we can not
                 * simplify it to a RELATIVE relocation.
                 */
                if (reloc_table[ortype].re_fsize != sizeof (Addr)) {
                        return (ld_add_outrel(0, rsp, ofl));
                }

                rsp->rel_rtype = R_AMD64_RELATIVE;
                if (ld_add_outrel(FLG_REL_ADVAL, rsp, ofl) == S_ERROR)
                        return (S_ERROR);
                rsp->rel_rtype = ortype;
                return (1);
        }

        /*
         * If the relocation is against a 'non-allocatable' section
         * and we can not resolve it now - then give a warning
         * message.
         *
         * We can not resolve the symbol if either:
         *      a) it's undefined
         *      b) it's defined in a shared library and a
         *         COPY relocation hasn't moved it to the executable
         *
         * Note: because we process all of the relocations against the
         *      text segment before any others - we know whether
         *      or not a copy relocation will be generated before
         *      we get here (see reloc_init()->reloc_segments()).
         */
        if (!(rsp->rel_flags & FLG_REL_LOAD) &&
            ((shndx == SHN_UNDEF) ||
            ((sdp->sd_ref == REF_DYN_NEED) &&
            ((sdp->sd_flags & FLG_SY_MVTOCOMM) == 0)))) {
                Conv_inv_buf_t  inv_buf;
                Os_desc         *osp = RELAUX_GET_OSDESC(rsp);

                /*
                 * If the relocation is against a SHT_SUNW_ANNOTATE
                 * section - then silently ignore that the relocation
                 * can not be resolved.
                 */
                if (osp && (osp->os_shdr->sh_type == SHT_SUNW_ANNOTATE))
                        return (0);
                ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_REL_EXTERNSYM),
                    conv_reloc_amd64_type(rsp->rel_rtype, 0, &inv_buf),
                    rsp->rel_isdesc->is_file->ifl_name,
                    ld_reloc_sym_name(rsp), osp->os_name);
                return (1);
        }

        /*
         * Perform relocation.
         */
```

```
1327         return (ld_add_actrel(NULL, rsp, ofl));
1328 }


1331 static uintptr_t
1332 ld_reloc_TLS(Boolean local, Rel_desc * rsp, Ofl_desc * ofl)
1333 {
1334         Word            rtype = rsp->rel_rtype;
1335         Sym_desc        *sdp = rsp->rel_sym;
1336         ofl_flag_t      flags = ofl->ofl_flags;
1337         Gotndx          *gnp;

1339         /*
1340          * If we're building an executable - use either the IE or LE access
1341          * model.  If we're building a shared object process any IE model.
1342          */
1343         if ((flags & FLG_OF_EXEC) || (IS_TLS_IE(rtype))) {
1344                 /*
1345                  * Set the DF_STATIC_TLS flag.
1346                  */
1347                 ofl->ofl_dtflags |= DF_STATIC_TLS;

1349                 if (!local || ((flags & FLG_OF_EXEC) == 0)) {
1350                         /*
1351                          * Assign a GOT entry for static TLS references.
1352                          */
1353                         if ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs,
1354                             GOT_REF_TLSIE, ofl, rsp)) == NULL) {

1356                                 if (ld_assign_got_TLS(local, rsp, ofl, sdp,
1357                                     gnp, GOT_REF_TLSIE, FLG_REL_STLS,
1358                                     rtype, R_AMD64_TPOFF64, 0) == S_ERROR)
1359                                         return (S_ERROR);
1360                         }

1362                         /*
1363                          * IE access model.
1364                          */
1365                         if (IS_TLS_IE(rtype))
1366                                 return (ld_add_actrel(FLG_REL_STLS, rsp, ofl));

1368                         /*
1369                          * Fixups are required for other executable models.
1370                          */
1371                         return (ld_add_actrel((FLG_REL_TLSFIX | FLG_REL_STLS),
1372                             rsp, ofl));
1373                 }

1375                 /*
1376                  * LE access model.
1377                  */
1378                 if (IS_TLS_LE(rtype))
1379                         return (ld_add_actrel(FLG_REL_STLS, rsp, ofl));

1381                 return (ld_add_actrel((FLG_REL_TLSFIX | FLG_REL_STLS),
1382                     rsp, ofl));
1383         }

1385         /*
1386          * Building a shared object.
1387          *
1388          * Assign a GOT entry for a dynamic TLS reference.
1389          */
1390         if (IS_TLS_LD(rtype) && ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs,
1391             GOT_REF_TLSLD, ofl, rsp)) == NULL)) {
```



```
1393                 if (ld_assign_got_TLS(local, rsp, ofl, sdp, gnp, GOT_REF_TLSLD,
1394                     FLG_REL_MTLS, rtype, R_AMD64_DTPMOD64, NULL) == S_ERROR)
1395                         return (S_ERROR);

1397         } else if (IS_TLS_GD(rtype) &&
1398             ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs, GOT_REF_TLSGD,
1399             ofl, rsp)) == NULL)) {

1401                 if (ld_assign_got_TLS(local, rsp, ofl, sdp, gnp, GOT_REF_TLSGD,
1402                     FLG_REL_DTLS, rtype, R_AMD64_DTPMOD64,
1403                     R_AMD64_DTPOFF64) == S_ERROR)
1404                         return (S_ERROR);
1405         }

1407         if (IS_TLS_LD(rtype))
1408                 return (ld_add_actrel(FLG_REL_MTLS, rsp, ofl));

1410         return (ld_add_actrel(FLG_REL_DTLS, rsp, ofl));
1411 }

1413 /* ARGSUSED5 */
1414 static uintptr_t
1415 ld_assign_got_ndx(Alist **alpp, Gotndx *pgnp, Gotref gref, Ofl_desc *ofl,
1416     Rel_desc *rsp, Sym_desc *sdp)
1417 {
1418         Xword           raddend;
1419         Gotndx          gn, *gnp;
1420         Aliste          idx;
1421         uint_t          gotents;

1423         raddend = rsp->rel_raddend;
1424         if (pgnp && (pgnp->gn_addend == raddend) && (pgnp->gn_gotref == gref))
1425                 return (1);

1427         if ((gref == GOT_REF_TLSGD) || (gref == GOT_REF_TLSLD))
1428                 gotents = 2;
1429         else
1430                 gotents = 1;

1432         gn.gn_addend = raddend;
1433         gn.gn_gotndx = ofl->ofl_gotcnt;
1434         gn.gn_gotref = gref;

1436         ofl->ofl_gotcnt += gotents;

1438         if (gref == GOT_REF_TLSLD) {
1439                 if (ofl->ofl_tlsldgotndx == NULL) {
1440                         if ((gnp = libld_malloc(sizeof (Gotndx))) == NULL)
1441                                 return (S_ERROR);
1442                         (void) memcpy(gnp, &gn, sizeof (Gotndx));
1443                         ofl->ofl_tlsldgotndx = gnp;
1444                 }
1445                 return (1);
1446         }

1448         idx = 0;
1449         for (ALIST_TRAVERSE(*alpp, idx, gnp)) {
1450                 if (gnp->gn_addend > raddend)
1451                         break;
1452         }

1454         /*
1455          * GOT indexes are maintained on an Alist, where there is typically
1456          * only one index.  The usage of this list is to scan the list to find
1457          * an index, and then apply that index immediately to a relocation.
1458          * Thus there are no external references to these GOT index structures
```

```
1459          * that can be compromised by the Alist being reallocated.
1460          */
1461         if (alist_insert(alpp, &gn, sizeof (Gotndx),
1462             AL_CNT_SDP_GOT, idx) == NULL)
1463                 return (S_ERROR);

1465         return (1);
1466 }

1468 static void
1469 ld_assign_plt_ndx(Sym_desc * sdp, Ofl_desc *ofl)
1470 {
1471         sdp->sd_aux->sa_PLTndx = 1 + ofl->ofl_pltcnt++;
1472         sdp->sd_aux->sa_PLTGOTndx = ofl->ofl_gotcnt++;
1473         ofl->ofl_flags |= FLG_OF_BLDGOT;
1474 }

1476 static uchar_t plt0_template[M_PLT_ENTSIZE] = {
1477 /* 0x00 PUSHQ GOT+8(%rip) */    0xff, 0x35, 0x00, 0x00, 0x00, 0x00,
1478 /* 0x06 JMP   *GOT+16(%rip) */  0xff, 0x25, 0x00, 0x00, 0x00, 0x00,
1479 /* 0x0c NOP */                  0x90,
1480 /* 0x0d NOP */                  0x90,
1481 /* 0x0e NOP */                  0x90,
1482 /* 0x0f NOP */                  0x90
1483 };

1485 /*
1486  * Initializes .got[0] with the _DYNAMIC symbol value.
1487  */
1488 static uintptr_t
1489 ld_fillin_gotplt(Ofl_desc *ofl)
1490 {
1491         int     bswap = (ofl->ofl_flags1 & FLG_OF1_ENCDIFF) != 0;

1493         if (ofl->ofl_osgot) {
1494                 Sym_desc        *sdp;

1496                 if ((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_DYNAMIC_U),
1497                     SYM_NOHASH, NULL, ofl)) != NULL) {
1498                         uchar_t *genptr;

1500                         genptr = ((uchar_t *)ofl->ofl_osgot->os_outdata->d_buf +
1501                             (M_GOT_XDYNAMIC * M_GOT_ENTSIZE));
1502                         /* LINTED */
1503                         *(Xword *)genptr = sdp->sd_sym->st_value;
1504                         if (bswap)
1505                                 /* LINTED */
1506                                 *(Xword *)genptr =
1507                                     /* LINTED */
1508                                     ld_bswap_Xword(*(Xword *)genptr);
1509                 }
1510         }

1512         /*
1513          * Fill in the reserved slot in the procedure linkage table the first
1514          * entry is:
1515          *      0x00 PUSHQ      GOT+8(%rip)         # GOT[1]
1516          *      0x06 JMP        *GOT+16(%rip)       # GOT[2]
1517          *      0x0c NOP
1518          *      0x0d NOP
1519          *      0x0e NOP
1520          *      0x0f NOP
1521          */
1522         if ((ofl->ofl_flags & FLG_OF_DYNAMIC) && ofl->ofl_osplt) {
1523                 uchar_t *pltent;
1524                 Xword   val1;
```



```
1526                 pltent = (uchar_t *)ofl->ofl_osplt->os_outdata->d_buf;
1527                 bcopy(plt0_template, pltent, sizeof (plt0_template));

1529                 /*
1530                  * If '-z noreloc' is specified - skip the do_reloc_ld
1531                  * stage.
1532                  */
1533                 if (!OFL_DO_RELOC(ofl))
1534                         return (1);

1536                 /*
1537                  * filin:
1538                  *      PUSHQ GOT + 8(%rip)
1539                  *
1540                  * Note: 0x06 below represents the offset to the
1541                  *       next instruction - which is what %rip will
1542                  *       be pointing at.
1543                  */
1544                 val1 = (ofl->ofl_osgot->os_shdr->sh_addr) +
1545                     (M_GOT_XLINKMAP * M_GOT_ENTSIZE) -
1546                     ofl->ofl_osplt->os_shdr->sh_addr - 0x06;

1548                 if (do_reloc_ld(&rdesc_r_amd64_gotpcrel, &pltent[0x02],
1549                     &val1, syn_rdesc_sym_name, MSG_ORIG(MSG_SPECFIL_PLTENT),
1550                     bswap, ofl->ofl_lml) == 0) {
1551                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_PLT_PLT0FAIL));
1552                         return (S_ERROR);
1553                 }

1555                 /*
1556                  * filin:
1557                  *  JMP *GOT+16(%rip)
1558                  */
1559                 val1 = (ofl->ofl_osgot->os_shdr->sh_addr) +
1560                     (M_GOT_XRTLD * M_GOT_ENTSIZE) -
1561                     ofl->ofl_osplt->os_shdr->sh_addr - 0x0c;

1563                 if (do_reloc_ld(&rdesc_r_amd64_gotpcrel, &pltent[0x08],
1564                     &val1, syn_rdesc_sym_name, MSG_ORIG(MSG_SPECFIL_PLTENT),
1565                     bswap, ofl->ofl_lml) == 0) {
1566                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_PLT_PLT0FAIL));
1567                         return (S_ERROR);
1568                 }
1569         }

1571         return (1);
1572 }



1576 /*
1577  * Template for generating "void (*)(void)" function
1578  */
1579 static const uchar_t nullfunc_tmpl[] = {        /* amd64 */
1580 /* 0x00 */      0x55,                           /* pushq  %rbp */
1581 /* 0x01 */      0x48, 0x8b, 0xec,               /* movq   %rsp,%rbp */
1582 /* 0x04 */      0x48, 0x8b, 0xe5,               /* movq   %rbp,%rsp */
1583 /* 0x07 */      0x5d,                           /* popq   %rbp */
1584 /* 0x08 */      0xc3                            /* ret */
1585 };


1588 /*
1589  * Function used to provide fill padding in SHF_EXECINSTR sections
1590  *
```

```
 * entry:
 *
 *      base - base address of section being filled
 *      offset - starting offset for fill within memory referenced by base
 *      cnt - # bytes to be filled
 *
 * exit:
 *      The fill has been completed.
 */
static void
execfill(void *base, off_t off, size_t cnt)
{
        /*
         * 0x90 is an X86 NOP instruction in both 32 and 64-bit worlds.
         * There are no alignment constraints.
         */
        (void) memset(off + (char *)base, 0x90, cnt);
}


/*
 * Return the ld_targ definition for this target.
 */
const Target *
ld_targ_init_x86(void)
{
        static const Target _ld_targ = {
                {                       /* Target_mach */
                        M_MACH,                 /* m_mach */
                        M_MACHPLUS,             /* m_machplus */
                        M_FLAGSPLUS,            /* m_flagsplus */
                        M_CLASS,                /* m_class */
                        M_DATA,                 /* m_data */

                        M_SEGM_ALIGN,           /* m_segm_align */
                        M_SEGM_ORIGIN,          /* m_segm_origin */
                        M_SEGM_AORIGIN,         /* m_segm_aorigin */
                        M_DATASEG_PERM,         /* m_dataseg_perm */
                        M_STACK_PERM,           /* m_stack_perm */
                        M_WORD_ALIGN,           /* m_word_align */
                        MSG_ORIG(MSG_PTH_RTLD_AMD64), /* m_def_interp */

                        /* Relocation type codes */
                        M_R_ARRAYADDR,          /* m_r_arrayaddr */
                        M_R_COPY,               /* m_r_copy */
                        M_R_GLOB_DAT,           /* m_r_glob_dat */
                        M_R_JMP_SLOT,           /* m_r_jmp_slot */
                        M_R_NUM,                /* m_r_num */
                        M_R_NONE,               /* m_r_none */
                        M_R_RELATIVE,           /* m_r_relative */
                        M_R_REGISTER,           /* m_r_register */

                        /* Relocation related constants */
                        M_REL_DT_COUNT,         /* m_rel_dt_count */
                        M_REL_DT_ENT,           /* m_rel_dt_ent */
                        M_REL_DT_SIZE,          /* m_rel_dt_size */
                        M_REL_DT_TYPE,          /* m_rel_dt_type */
                        M_REL_SHT_TYPE,         /* m_rel_sht_type */

                        /* GOT related constants */
                        M_GOT_ENTSIZE,          /* m_got_entsize */
                        M_GOT_XNumber,          /* m_got_xnumber */

                        /* PLT related constants */
                        M_PLT_ALIGN,            /* m_plt_align */
                        M_PLT_ENTSIZE,          /* m_plt_entsize */
```



```
                        M_PLT_RESERVSZ,         /* m_plt_reservsz */
                        M_PLT_SHF_FLAGS,        /* m_plt_shf_flags */

                        /* Section type of .eh_frame/.eh_frame_hdr sections */
                        SHT_AMD64_UNWIND,       /* m_sht_unwind */

                        M_DT_REGISTER,          /* m_dt_register */
                },
                {                       /* Target_machid */
                        M_ID_ARRAY,             /* id_array */
                        M_ID_BSS,               /* id_bss */
                        M_ID_CAP,               /* id_cap */
                        M_ID_CAPINFO,           /* id_capinfo */
                        M_ID_CAPCHAIN,          /* id_capchain */
                        M_ID_DATA,              /* id_data */
                        M_ID_DYNAMIC,           /* id_dynamic */
                        M_ID_DYNSORT,           /* id_dynsort */
                        M_ID_DYNSTR,            /* id_dynstr */
                        M_ID_DYNSYM,            /* id_dynsym */
                        M_ID_DYNSYM_NDX,        /* id_dynsym_ndx */
                        M_ID_GOT,               /* id_got */
                        M_ID_UNKNOWN,           /* id_gotdata (unused) */
                        M_ID_HASH,              /* id_hash */
                        M_ID_INTERP,            /* id_interp */
                        M_ID_LBSS,              /* id_lbss */
                        M_ID_LDYNSYM,           /* id_ldynsym */
                        M_ID_NOTE,              /* id_note */
                        M_ID_NULL,              /* id_null */
                        M_ID_PLT,               /* id_plt */
                        M_ID_REL,               /* id_rel */
                        M_ID_STRTAB,            /* id_strtab */
                        M_ID_SYMINFO,           /* id_syminfo */
                        M_ID_SYMTAB,            /* id_symtab */
                        M_ID_SYMTAB_NDX,        /* id_symtab_ndx */
                        M_ID_TEXT,              /* id_text */
                        M_ID_TLS,               /* id_tls */
                        M_ID_TLSBSS,            /* id_tlsbss */
                        M_ID_UNKNOWN,           /* id_unknown */
                        M_ID_UNWIND,            /* id_unwind */
                        M_ID_UNWINDHDR,         /* id_unwindhdr */
                        M_ID_USER,              /* id_user */
                        M_ID_VERSION,           /* id_version */
                },
                {                       /* Target_nullfunc */
                        nullfunc_tmpl,          /* nf_template */
                        sizeof (nullfunc_tmpl), /* nf_size */
                },
                {                       /* Target_fillfunc */
                        execfill                /* ff_execfill */
                },
                {                       /* Target_machrel */
                        reloc_table,

                        ld_init_rel,            /* mr_init_rel */
                        ld_mach_eflags,         /* mr_mach_eflags */
                        ld_mach_make_dynamic,   /* mr_mach_make_dynamic */
                        ld_mach_update_odynamic, /* mr_mach_update_odynamic */
                        ld_calc_plt_addr,       /* mr_calc_plt_addr */
                        ld_perform_outreloc,    /* mr_perform_outreloc */
                        ld_do_activerelocs,     /* mr_do_activerelocs */
                        ld_add_outrel,          /* mr_add_outrel */
                        NULL,                   /* mr_reloc_register */
                        ld_reloc_local,         /* mr_reloc_local */
                        NULL,                   /* mr_reloc_GOTOP */
                        ld_reloc_TLS,           /* mr_reloc_TLS */
                        NULL,                   /* mr_assign_got */
```

```
1723                        ld_find_got_ndx,        /* mr_find_got_ndx */
1724                        ld_calc_got_offset,     /* mr_calc_got_offset */
1725                        ld_assign_got_ndx,      /* mr_assign_got_ndx */
1726                        ld_assign_plt_ndx,      /* mr_assign_plt_ndx */
1727                        NULL,                   /* mr_allocate_got */
1728                        ld_fillin_gotplt,       /* mr_fillin_gotplt */
1729                },
1730                {                       /* Target_machsym */
1731                        NULL,                   /* ms_reg_check */
1732                        NULL,                   /* ms_mach_sym_typecheck */
1733                        NULL,                   /* ms_is_regsym */
1734                        NULL,                   /* ms_reg_find */
1735                        NULL                    /* ms_reg_enter */
1736                }
1737        };

1739        return (&_ld_targ);
1740 }
```

```
**********************************************************
   46468 Wed May 22 03:21:44 2019
new/usr/src/cmd/sgs/libld/common/machrel.intel.c
11057 hidden undefined weak symbols should not leave relocations
11058 libld entrance descriptor assertions get NDEBUG check backwards
**********************************************************
______unchanged_portion_omitted_

 224 static uintptr_t
 225 ld_perform_outreloc(Rel_desc * orsp, Ofl_desc * ofl, Boolean *remain_seen)
 226 {
 227         Os_desc *       relosp, * osp = 0;
 228         Word            ndx, roffset, value;
 229         Rel             rea;
 230         char            *relbits;
 231         Sym_desc *      sdp, * psym = (Sym_desc *)0;
 232         int             sectmoved = 0;

 234         sdp = orsp->rel_sym;

 236         /*
 237          * If the section this relocation is against has been discarded
 238          * (-zignore), then also discard (skip) the relocation itself.
 239          */
 240         if (orsp->rel_isdesc && ((orsp->rel_flags &
 241             (FLG_REL_GOT | FLG_REL_BSS | FLG_REL_PLT | FLG_REL_NOINFO)) == 0) &&
 242             (orsp->rel_isdesc->is_flags & FLG_IS_DISCARD)) {
 243                 DBG_CALL(Dbg_reloc_discard(ofl->ofl_lml, M_MACH, orsp));
 244                 return (1);
 245         }

 247         /*
 248          * If this is a relocation against a move table, or expanded move
 249          * table, adjust the relocation entries.
 250          */
 251         if (RELAUX_GET_MOVE(orsp))
 252                 ld_adj_movereloc(ofl, orsp);

 254         /*
 255          * If this is a relocation against a section using a partial initialized
 256          * symbol, adjust the embedded symbol info.
 257          *
 258          * The second argument of the am_I_partial() is the value stored at the
 259          * target address relocation is going to be applied.
 260          */
 261         if (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION) {
 262                 if (ofl->ofl_parsyms &&
 263                     (sdp->sd_isc->is_flags & FLG_IS_RELUPD) &&
 264                     /* LINTED */
 265                     (psym = ld_am_I_partial(orsp, *(Xword *)
 266                     ((uchar_t *)(orsp->rel_isdesc->is_indata->d_buf) +
 267                     orsp->rel_roffset)))) {
 268                         DBG_CALL(Dbg_move_outsctadj(ofl->ofl_lml, psym));
 269                         sectmoved = 1;
 270                 }
 271         }

 273         value = sdp->sd_sym->st_value;

 275         if (orsp->rel_flags & FLG_REL_GOT) {
 276                 osp = ofl->ofl_osgot;
 277                 roffset = (Word)ld_calc_got_offset(orsp, ofl);

 279         } else if (orsp->rel_flags & FLG_REL_PLT) {
 280                 /*
 281                  * Note that relocations for PLT's actually
```



```
 282                  * cause a relocation againt the GOT.
 283                  */
 284                 osp = ofl->ofl_osplt;
 285                 roffset = (Word) (ofl->ofl_osgot->os_shdr->sh_addr) +
 286                     sdp->sd_aux->sa_PLTGOTndx * M_GOT_ENTSIZE;

 288                 plt_entry(ofl, osp->os_relosdesc->os_szoutrels, sdp);

 290         } else if (orsp->rel_flags & FLG_REL_BSS) {
 291                 /*
 292                  * This must be a R_386_COPY.  For these set the roffset to
 293                  * point to the new symbols location.
 294                  */
 295                 osp = ofl->ofl_isbss->is_osdesc;
 296                 roffset = (Word)value;
 297         } else {
 298                 osp = RELAUX_GET_OSDESC(orsp);

 300                 /*
 301                  * Calculate virtual offset of reference point; equals offset
 302                  * into section + vaddr of section for loadable sections, or
 303                  * offset plus section displacement for nonloadable sections.
 304                  */
 305                 roffset = orsp->rel_roffset +
 306                     (Off)_elf_getxoff(orsp->rel_isdesc->is_indata);
 307                 if (!(ofl->ofl_flags & FLG_OF_RELOBJ))
 308                         roffset += orsp->rel_isdesc->is_osdesc->
 309                             os_shdr->sh_addr;
 310         }

 312         if ((osp == 0) || ((relosp = osp->os_relosdesc) == 0))
 313                 relosp = ofl->ofl_osrel;

 315         /*
 316          * Assign the symbols index for the output relocation.  If the
 317          * relocation refers to a SECTION symbol then it's index is based upon
 318          * the output sections symbols index.  Otherwise the index can be
 319          * derived from the symbols index itself.
 320          */
 321         if (orsp->rel_rtype == R_386_RELATIVE)
 322                 ndx = STN_UNDEF;
 323         else if ((orsp->rel_flags & FLG_REL_SCNNDX) ||
 324             (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION)) {
 325                 if (sectmoved == 0) {
 326                         /*
 327                          * Check for a null input section. This can
 328                          * occur if this relocation references a symbol
 329                          * generated by sym_add_sym().
 330                          */
 331                         if (sdp->sd_isc && sdp->sd_isc->is_osdesc)
 332                                 ndx = sdp->sd_isc->is_osdesc->os_identndx;
 333                         else
 334                                 ndx = sdp->sd_shndx;
 335                 } else
 336                         ndx = ofl->ofl_parexpnndx;
 337         } else
 338                 ndx = sdp->sd_symndx;

 340         /*
 341          * If we have a replacement value for the relocation
 342          * target, put it in place now.
 343          */
 344         if (orsp->rel_flags & FLG_REL_NADDEND) {
 345                 Xword   addend = orsp->rel_raddend;
 346                 uchar_t *addr;
```

```
348                 /*
349                  * Get the address of the data item we need to modify.
350                  */
351                 addr = (uchar_t *)((uintptr_t)orsp->rel_roffset +
352                     (uintptr_t)_elf_getxoff(orsp->rel_isdesc->is_indata));
353                 addr += (uintptr_t)RELAUX_GET_OSDESC(orsp)->os_outdata->d_buf;
354                 if (ld_reloc_targval_set(ofl, orsp, addr, addend) == 0)
355                         return (S_ERROR);
356         }

358         if ((orsp->rel_rtype != M_R_NONE) &&
359             (orsp->rel_rtype != M_R_RELATIVE)) {
360                 if (ndx == 0) {
361                         Conv_inv_buf_t  inv_buf;
362                         Is_desc *isp = orsp->rel_isdesc;

364                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_NOSYMBOL),
365                             conv_reloc_type(ofl->ofl_nehdr->e_machine,
366                             orsp->rel_rtype, 0, &inv_buf),
367                             isp->is_file->ifl_name, EC_WORD(isp->is_scnndx),
368                             isp->is_name, EC_XWORD(roffset));
369                         return (S_ERROR);
370                 }
371         }
358         relbits = (char *)relosp->os_outdata->d_buf;

373         rea.r_info = ELF_R_INFO(ndx, orsp->rel_rtype);
374         rea.r_offset = roffset;
375         DBG_CALL(Dbg_reloc_out(ofl, ELF_DBG_LD, SHT_REL, &rea, relosp->os_name,
376             ld_reloc_sym_name(orsp)));

378         /*
379          * Assert we haven't walked off the end of our relocation table.
380          */
381         assert(relosp->os_szoutrels <= relosp->os_shdr->sh_size);

383         relbits = (char *)relosp->os_outdata->d_buf;

385 #endif /* ! codereview */
386         (void) memcpy((relbits + relosp->os_szoutrels),
387             (char *)&rea, sizeof (Rel));
388         relosp->os_szoutrels += sizeof (Rel);

390         /*
391          * Determine if this relocation is against a non-writable, allocatable
392          * section.  If so we may need to provide a text relocation diagnostic.
393          * Note that relocations against the .plt (R_386_JMP_SLOT) actually
394          * result in modifications to the .got.
395          */
396         if (orsp->rel_rtype == R_386_JMP_SLOT)
397                 osp = ofl->ofl_osgot;

399         ld_reloc_remain_entry(orsp, osp, ofl, remain_seen);
400         return (1);
401 }

403 /*
404  * i386 Instructions for TLS processing
405  */
406 static uchar_t tlsinstr_gd_ie[] = {
407         /*
408          * 0x00 movl %gs:0x0, %eax
409          */
410         0x65, 0xa1, 0x00, 0x00, 0x00, 0x00,
411         /*
412          * 0x06 addl x(%eax), %eax
```



```
413          * 0x0c ...
414          */
415         0x03, 0x80, 0x00, 0x00, 0x00, 0x00
416 };

418 static uchar_t tlsinstr_gd_le[] = {
419         /*
420          * 0x00 movl %gs:0x0, %eax
421          */
422         0x65, 0xa1, 0x00, 0x00, 0x00, 0x00,
423         /*
424          * 0x06 addl $0x0, %eax
425          */
426         0x05, 0x00, 0x00, 0x00, 0x00,
427         /*
428          * 0x0b nop
429          * 0x0c
430          */
431         0x90
432 };

434 static uchar_t tlsinstr_ld_le_movgs[] = {
435         /*
436          * 0x00 movl %gs:0x0,%eax
437          */
438         0x65, 0xa1, 0x00, 0x00, 0x00, 0x00,
439 };

441 /*
442  * 0x00 nopl 0(%eax,%eax) -- the intel recommended 5-byte nop
443  * See Intel´fi 64 and IA-32 Architectures Software Developerâ  s Manual
444  *    Volume 2B: Instruction Set Reference, M-U
445  *    Table 4-12, Recommended Multi-Byte Sequence of NOP Instruction
446  */
447 static uchar_t tlsinstr_nop5[] = {

449         0x0f, 0x1f, 0x44, 0x00, 0x00
450 };

452 #define TLS_GD_IE_MOV   0x8b    /* movl opcode */
453 #define TLS_GD_IE_POP   0x58    /* popl + reg */

455 #define TLS_GD_LE_MOVL  0xb8    /* movl + reg */

457 #define TLS_NOP         0x90    /* NOP instruction */

459 #define MODRM_MSK_MOD   0xc0
460 #define MODRM_MSK_RO    0x38
461 #define MODRM_MSK_RM    0x07

463 #define SIB_MSK_SS      0xc0
464 #define SIB_MSK_IND     0x38
465 #define SIB_MSK_BS      0x07

467 static Fixupret
468 tls_fixups(Ofl_desc *ofl, Rel_desc *arsp)
469 {
470         Sym_desc        *sdp = arsp->rel_sym;
471         Word            rtype = arsp->rel_rtype;
472         uchar_t         *offset, r1, r2;

474         offset = (uchar_t *)((uintptr_t)arsp->rel_roffset +
475             (uintptr_t)_elf_getxoff(arsp->rel_isdesc->is_indata) +
476             (uintptr_t)RELAUX_GET_OSDESC(arsp)->os_outdata->d_buf);

478         if (sdp->sd_ref == REF_DYN_NEED) {
```

```
479                 /*
480                  * IE reference model
481                  */
482                 switch (rtype) {
483                 case R_386_TLS_GD:
484                         /*
485                          * Transition:
486                          *      0x0 leal x@tlsgd(,r1,1), %eax
487                          *      0x7 call ___tls_get_addr
488                          *      0xc
489                          * To:
490                          *      0x0 movl %gs:0, %eax
491                          *      0x6 addl x@gotntpoff(r1), %eax
492                          */
493                         DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
494                             R_386_TLS_GOTIE, arsp, ld_reloc_sym_name));
495                         arsp->rel_rtype = R_386_TLS_GOTIE;
496                         arsp->rel_roffset += 5;

498                         /*
499                          * Adjust 'offset' to beginning of instruction
500                          * sequence.
501                          */
502                         offset -= 3;
503                         r1 = (offset[2] & SIB_MSK_IND) >> 3;
504                         (void) memcpy(offset, tlsinstr_gd_ie,
505                             sizeof (tlsinstr_gd_ie));

507                         /*
508                          * set register %r1 into the addl
509                          * instruction.
510                          */
511                         offset[0x7] |= r1;
512                         return (FIX_RELOC);

514                 case R_386_TLS_GD_PLT:
515                         /*
516                          * Fixup done via the TLS_GD relocation
517                          */
518                         DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
519                             R_386_NONE, arsp, ld_reloc_sym_name));
520                         return (FIX_DONE);
521                 }
522         }

524         /*
525          * LE reference model
526          */
527         switch (rtype) {
528         case R_386_TLS_GD:
529                 /*
530                  * Transition:
531                  *      0x0 leal x@tlsgd(,r1,1), %eax
532                  *      0x7 call ___tls_get_addr
533                  *      0xc
534                  * To:
535                  *      0x0 movl %gs:0, %eax
536                  *      0x6 addl $x@ntpoff, %eax
537                  *      0xb nop
538                  *      0xc
539                  */
540                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
541                     R_386_TLS_LE, arsp, ld_reloc_sym_name));

543                 arsp->rel_rtype = R_386_TLS_LE;
544                 arsp->rel_roffset += 4;
```



```
546                 /*
547                  * Adjust 'offset' to beginning of instruction
548                  * sequence.
549                  */
550                 offset -= 3;
551                 (void) memcpy(offset, tlsinstr_gd_le,
552                     sizeof (tlsinstr_gd_le));
553                 return (FIX_RELOC);

555         case R_386_TLS_GD_PLT:
556         case R_386_PLT32:
557                 /*
558                  * Fixup done via the TLS_GD/TLS_LDM relocation processing
559                  * and ld_reloc_plt() handling __tls_get_addr().
560                  */
561                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
562                     R_386_NONE, arsp, ld_reloc_sym_name));
563                 return (FIX_DONE);

565         case R_386_TLS_LDM_PLT:
566                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
567                     R_386_NONE, arsp, ld_reloc_sym_name));

569                 /*
570                  * Transition:
571                  *      call __tls_get_addr()
572                  * to:
573                  *      nopl 0x0(%eax,%eax)
574                  */
575                 (void) memcpy(offset - 1, tlsinstr_nop5,
576                     sizeof (tlsinstr_nop5));
577                 return (FIX_DONE);

579         case R_386_TLS_LDM:
580                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
581                     R_386_NONE, arsp, ld_reloc_sym_name));

583                 /*
584                  * Transition:
585                  *
586                  *  0x00 leal x1@tlsldm(%ebx), %eax
587                  *  0x06 call ___tls_get_addr
588                  *
589                  * to:
590                  *
591                  *  0x00 movl %gs:0, %eax
592                  */
593                 (void) memcpy(offset - 2, tlsinstr_ld_le_movgs,
594                     sizeof (tlsinstr_ld_le_movgs));

596                 /*
597                  *  We implicitly treat this as if a R_386_TLS_LDM_PLT for the
598                  *  __tls_get_addr call followed it as the GNU compiler
599                  *  doesn't generate one.  This is safe, because if one _does_
600                  *  exist we'll just write the nop again.
601                  */
602                 (void) memcpy(offset + 4, tlsinstr_nop5,
603                     sizeof (tlsinstr_nop5));
604                 return (FIX_DONE);

606         case R_386_TLS_LDO_32:
607                 /*
608                  *  Instructions:
609                  *
610                  *  0x10 leal x1@dtpoff(%eax), %edx     R_386_TLS_LDO_32
```

```
611                  *              to
612                  *  0x10 leal x1@ntpoff(%eax), %edx     R_386_TLS_LE
613                  *
614                  */
615                 offset -= 2;

617                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
618                     R_386_TLS_LE, arsp, ld_reloc_sym_name));
619                 arsp->rel_rtype = R_386_TLS_LE;
620                 return (FIX_RELOC);

622         case R_386_TLS_GOTIE:
623                 /*
624                  * These transitions are a little different than the
625                  * others, in that we could have multiple instructions
626                  * pointed to by a single relocation.  Depending upon the
627                  * instruction, we perform a different code transition.
628                  *
629                  * Here's the known transitions:
630                  *
631                  *  1) movl foo@gotntpoff(%reg1), %reg2
632                  *      0x8b, 0x80 | (reg2 << 3) | reg1, foo@gotntpoff
633                  *
634                  *  2) addl foo@gotntpoff(%reg1), %reg2
635                  *      0x03, 0x80 | (reg2 << 3) | reg1, foo@gotntpoff
636                  *
637                  *  Transitions IE -> LE
638                  *
639                  *  1) movl $foo@ntpoff, %reg2
640                  *      0xc7, 0xc0 | reg2, foo@ntpoff
641                  *
642                  *  2) addl $foo@ntpoff, %reg2
643                  *      0x81, 0xc0 | reg2, foo@ntpoff
644                  *
645                  * Note: reg1 != 4 (%esp)
646                  */
647                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
648                     R_386_TLS_LE, arsp, ld_reloc_sym_name));
649                 arsp->rel_rtype = R_386_TLS_LE;

651                 offset -= 2;
652                 r2 = (offset[1] & MODRM_MSK_RO) >> 3;
653                 if (offset[0] == 0x8b) {
654                         /* case 1 above */
655                         offset[0] = 0xc7;       /* movl */
656                         offset[1] = 0xc0 | r2;
657                         return (FIX_RELOC);
658                 }

660                 if (offset[0] == 0x03) {
661                         /* case 2 above */
662                         assert(offset[0] == 0x03);
663                         offset[0] = 0x81;       /* addl */
664                         offset[1] = 0xc0 | r2;
665                         return (FIX_RELOC);
666                 }

668                 /*
669                  * Unexpected instruction sequence - fatal error.
670                  */
671                 {
672                         Conv_inv_buf_t  inv_buf;

674                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
675                             conv_reloc_386_type(arsp->rel_rtype, 0, &inv_buf),
676                             arsp->rel_isdesc->is_file->ifl_name,
```



```
677                             ld_reloc_sym_name(arsp),
678                             arsp->rel_isdesc->is_name,
679                             EC_OFF(arsp->rel_roffset));
680                 }
681                 return (FIX_ERROR);

683         case R_386_TLS_IE:
684                 /*
685                  * These transitions are a little different than the
686                  * others, in that we could have multiple instructions
687                  * pointed to by a single relocation.  Depending upon the
688                  * instruction, we perform a different code transition.
689                  *
690                  * Here's the known transitions:
691                  *  1) movl foo@indntpoff, %eax
692                  *      0xa1, foo@indntpoff
693                  *
694                  *  2) movl foo@indntpoff, %eax
695                  *      0x8b, 0x05 | (reg << 3), foo@gotntpoff
696                  *
697                  *  3) addl foo@indntpoff, %eax
698                  *      0x03, 0x05 | (reg << 3), foo@gotntpoff
699                  *
700                  *  Transitions IE -> LE
701                  *
702                  *  1) movl $foo@ntpoff, %eax
703                  *      0xb8, foo@ntpoff
704                  *
705                  *  2) movl $foo@ntpoff, %reg
706                  *      0xc7, 0xc0 | reg, foo@ntpoff
707                  *
708                  *  3) addl $foo@ntpoff, %reg
709                  *      0x81, 0xc0 | reg, foo@ntpoff
710                  */
711                 arsp->rel_rtype = R_386_TLS_LE;
712                 offset--;
713                 if (offset[0] == 0xa1) {
714                         /* case 1 above */
715                         offset[0] = 0xb8;       /*  movl */
716                         return (FIX_RELOC);
717                 }

719                 offset--;
720                 if (offset[0] == 0x8b) {
721                         /* case 2 above */
722                         r2 = (offset[1] & MODRM_MSK_RO) >> 3;
723                         offset[0] = 0xc7;       /* movl */
724                         offset[1] = 0xc0 | r2;
725                         return (FIX_RELOC);
726                 }
727                 if (offset[0] == 0x03) {
728                         /* case 3 above */
729                         r2 = (offset[1] & MODRM_MSK_RO) >> 3;
730                         offset[0] = 0x81;       /* addl */
731                         offset[1] = 0xc0 | r2;
732                         return (FIX_RELOC);
733                 }
734                 /*
735                  * Unexpected instruction sequence - fatal error.
736                  */
737                 {
738                         Conv_inv_buf_t  inv_buf;

740                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADTLSINS),
741                             conv_reloc_386_type(arsp->rel_rtype, 0, &inv_buf),
742                             arsp->rel_isdesc->is_file->ifl_name,
```

```
743                             ld_reloc_sym_name(arsp),
744                             arsp->rel_isdesc->is_name,
745                             EC_OFF(arsp->rel_roffset));
746                 }
747                 return (FIX_ERROR);
748         }
749         return (FIX_RELOC);
750 }

752 static uintptr_t
753 ld_do_activerelocs(Ofl_desc *ofl)
754 {
755         Rel_desc        *arsp;
756         Rel_cachebuf    *rcbp;
757         Aliste          idx;
758         uintptr_t       return_code = 1;
759         ofl_flag_t      flags = ofl->ofl_flags;

761         if (aplist_nitems(ofl->ofl_actrels.rc_list) != 0)
762                 DBG_CALL(Dbg_reloc_doact_title(ofl->ofl_lml));

764         /*
765          * Process active relocations.
766          */
767         REL_CACHE_TRAVERSE(&ofl->ofl_actrels, idx, rcbp, arsp) {
768                 uchar_t         *addr;
769                 Xword           value;
770                 Sym_desc        *sdp;
771                 const char      *ifl_name;
772                 Xword           refaddr;
773                 int             moved = 0;
774                 Gotref          gref;
775                 Os_desc         *osp;

777                 /*
778                  * If the section this relocation is against has been discarded
779                  * (-zignore), then discard (skip) the relocation itself.
780                  */
781                 if ((arsp->rel_isdesc->is_flags & FLG_IS_DISCARD) &&
782                     ((arsp->rel_flags & (FLG_REL_GOT | FLG_REL_BSS |
783                     FLG_REL_PLT | FLG_REL_NOINFO)) == 0)) {
784                         DBG_CALL(Dbg_reloc_discard(ofl->ofl_lml, M_MACH, arsp));
785                         continue;
786                 }

788                 /*
789                  * We determine what the 'got reference' model (if required)
790                  * is at this point.  This needs to be done before tls_fixup()
791                  * since it may 'transition' our instructions.
792                  *
793                  * The got table entries have already been assigned,
794                  * and we bind to those initial entries.
795                  */
796                 if (arsp->rel_flags & FLG_REL_DTLS)
797                         gref = GOT_REF_TLSGD;
798                 else if (arsp->rel_flags & FLG_REL_MTLS)
799                         gref = GOT_REF_TLSLD;
800                 else if (arsp->rel_flags & FLG_REL_STLS)
801                         gref = GOT_REF_TLSIE;
802                 else
803                         gref = GOT_REF_GENERIC;

805                 /*
806                  * Perform any required TLS fixups.
807                  */
808                 if (arsp->rel_flags & FLG_REL_TLSFIX) {
```



```
809                         Fixupret        ret;

811                         if ((ret = tls_fixups(ofl, arsp)) == FIX_ERROR)
812                                 return (S_ERROR);
813                         if (ret == FIX_DONE)
814                                 continue;
815                 }

817                 /*
818                  * If this is a relocation against a move table, or
819                  * expanded move table, adjust the relocation entries.
820                  */
821                 if (RELAUX_GET_MOVE(arsp))
822                         ld_adj_movereloc(ofl, arsp);

824                 sdp = arsp->rel_sym;
825                 refaddr = arsp->rel_roffset +
826                     (Off)_elf_getxoff(arsp->rel_isdesc->is_indata);

828                 if (arsp->rel_flags & FLG_REL_CLVAL)
829                         value = 0;
830                 else if (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION) {
831                         /*
832                          * The value for a symbol pointing to a SECTION
833                          * is based off of that sections position.
834                          */
835                         if (sdp->sd_isc->is_flags & FLG_IS_RELUPD) {
836                                 Sym_desc        *sym;
837                                 Xword           radd;
838                                 uchar_t         *raddr = (uchar_t *)
839                                     arsp->rel_isdesc->is_indata->d_buf +
840                                     arsp->rel_roffset;

842                                 /*
843                                  * This is a REL platform. Hence, the second
844                                  * argument of ld_am_I_partial() is the value
845                                  * stored at the target address where the
846                                  * relocation is going to be applied.
847                                  */
848                                 if (ld_reloc_targval_get(ofl, arsp, raddr,
849                                     &radd) == 0)
850                                         return (S_ERROR);
851                                 sym = ld_am_I_partial(arsp, radd);
852                                 if (sym) {
853                                         Sym     *osym = sym->sd_osym;

855                                         /*
856                                          * The symbol was moved, so adjust the
857                                          * value relative to the new section.
858                                          */
859                                         value = sym->sd_sym->st_value;
860                                         moved = 1;

862                                         /*
863                                          * The original raddend covers the
864                                          * displacement from the section start
865                                          * to the desired address. The value
866                                          * computed above gets us from the
867                                          * section start to the start of the
868                                          * symbol range. Adjust the old raddend
869                                          * to remove the offset from section
870                                          * start to symbol start, leaving the
871                                          * displacement within the range of
872                                          * the symbol.
873                                          */
874                                         if (osym->st_value != 0) {
```

```
875                                             radd -= osym->st_value;
876                                             if (ld_reloc_targval_set(ofl,
877                                                 arsp, raddr, radd) == 0)
878                                                     return (S_ERROR);
879                                     }
880                             }
881                     }
882                     if (!moved) {
883                             value = _elf_getxoff(sdp->sd_isc->is_indata);
884                             if (sdp->sd_isc->is_shdr->sh_flags & SHF_ALLOC)
885                                     value += sdp->sd_isc->
886                                         is_osdesc->os_shdr->sh_addr;
887                     }
888                     if (sdp->sd_isc->is_shdr->sh_flags & SHF_TLS)
889                             value -= ofl->ofl_tlsphdr->p_vaddr;
890
891             } else if (IS_SIZE(arsp->rel_rtype)) {
892                     /*
893                      * Size relocations require the symbols size.
894                      */
895                     value = sdp->sd_sym->st_size;
896
897             } else if ((sdp->sd_flags & FLG_SY_CAP) &&
898                 sdp->sd_aux && sdp->sd_aux->sa_PLTndx) {
899                     /*
900                      * If relocation is against a capabilities symbol, we
901                      * need to jump to an associated PLT, so that at runtime
902                      * ld.so.1 is involved to determine the best binding
903                      * choice. Otherwise, the value is the symbols value.
904                      */
905                     value = ld_calc_plt_addr(sdp, ofl);
906
907             } else
908                     value = sdp->sd_sym->st_value;
909
910             /*
911              * Relocation against the GLOBAL_OFFSET_TABLE.
912              */
913             if ((arsp->rel_flags & FLG_REL_GOT) &&
914                 !ld_reloc_set_aux_osdesc(ofl, arsp, ofl->ofl_osgot))
915                     return (S_ERROR);
916             osp = RELAUX_GET_OSDESC(arsp);
917
918             /*
919              * If loadable and not producing a relocatable object add the
920              * sections virtual address to the reference address.
921              */
922             if ((arsp->rel_flags & FLG_REL_LOAD) &&
923                 ((flags & FLG_OF_RELOBJ) == 0))
924                     refaddr +=
925                         arsp->rel_isdesc->is_osdesc->os_shdr->sh_addr;
926
927             /*
928              * If this entry has a PLT assigned to it, its value is actually
929              * the address of the PLT (and not the address of the function).
930              */
931             if (IS_PLT(arsp->rel_rtype)) {
932                     if (sdp->sd_aux && sdp->sd_aux->sa_PLTndx)
933                             value = ld_calc_plt_addr(sdp, ofl);
934             }
935
936             /*
937              * Determine whether the value needs further adjustment. Filter
938              * through the attributes of the relocation to determine what
939              * adjustment is required.  Note, many of the following cases
940              * are only applicable when a .got is present.  As a .got is
```



```
941              * not generated when a relocatable object is being built,
942              * any adjustments that require a .got need to be skipped.
943              */
944             if ((arsp->rel_flags & FLG_REL_GOT) &&
945                 ((flags & FLG_OF_RELOBJ) == 0)) {
946                     Xword           R1addr;
947                     uintptr_t       R2addr;
948                     Word            gotndx;
949                     Gotndx          *gnp;
950
951                     /*
952                      * Perform relocation against GOT table.  Since this
953                      * doesn't fit exactly into a relocation we place the
954                      * appropriate byte in the GOT directly
955                      *
956                      * Calculate offset into GOT at which to apply
957                      * the relocation.
958                      */
959                     gnp = ld_find_got_ndx(sdp->sd_GOTndxs, gref, ofl, NULL);
960                     assert(gnp);
961
962                     if (arsp->rel_rtype == R_386_TLS_DTPOFF32)
963                             gotndx = gnp->gn_gotndx + 1;
964                     else
965                             gotndx = gnp->gn_gotndx;
966
967                     R1addr = (Xword)(gotndx * M_GOT_ENTSIZE);
968
969                     /*
970                      * Add the GOTs data's offset.
971                      */
972                     R2addr = R1addr + (uintptr_t)osp->os_outdata->d_buf;
973
974                     DBG_CALL(Dbg_reloc_doact(ofl->ofl_lml, ELF_DBG_LD_ACT,
975                         M_MACH, SHT_REL, arsp, R1addr, value,
976                         ld_reloc_sym_name));
977
978                     /*
979                      * And do it.
980                      */
981                     if (ofl->ofl_flags1 & FLG_OF1_ENCDIFF)
982                             *(Xword *)R2addr = ld_bswap_Xword(value);
983                     else
984                             *(Xword *)R2addr = value;
985                     continue;
986
987             } else if (IS_GOT_BASED(arsp->rel_rtype) &&
988                 ((flags & FLG_OF_RELOBJ) == 0)) {
989                     value -= ofl->ofl_osgot->os_shdr->sh_addr;
990
991             } else if (IS_GOT_PC(arsp->rel_rtype) &&
992                 ((flags & FLG_OF_RELOBJ) == 0)) {
993                     value = (Xword)(ofl->ofl_osgot->os_shdr->sh_addr) -
994                         refaddr;
995
996             } else if ((IS_PC_RELATIVE(arsp->rel_rtype)) &&
997                 (((flags & FLG_OF_RELOBJ) == 0) ||
998                 (osp == sdp->sd_isc->is_osdesc))) {
999                     value -= refaddr;
1000
1001            } else if (IS_TLS_INS(arsp->rel_rtype) &&
1002                IS_GOT_RELATIVE(arsp->rel_rtype) &&
1003                ((flags & FLG_OF_RELOBJ) == 0)) {
1004                    Gotndx  *gnp;
1005
1006                    gnp = ld_find_got_ndx(sdp->sd_GOTndxs, gref, ofl, NULL);
```

```
1007                        assert(gnp);
1008                        value = (Xword)gnp->gn_gotndx * M_GOT_ENTSIZE;
1009                        if (arsp->rel_rtype == R_386_TLS_IE) {
1010                                value += ofl->ofl_osgot->os_shdr->sh_addr;
1011                        }
1012
1013                } else if (IS_GOT_RELATIVE(arsp->rel_rtype) &&
1014                    ((flags & FLG_OF_RELOBJ) == 0)) {
1015                        Gotndx *gnp;
1016
1017                        gnp = ld_find_got_ndx(sdp->sd_GOTndxs,
1018                            GOT_REF_GENERIC, ofl, NULL);
1019                        assert(gnp);
1020                        value = (Xword)gnp->gn_gotndx * M_GOT_ENTSIZE;
1021
1022                } else if ((arsp->rel_flags & FLG_REL_STLS) &&
1023                    ((flags & FLG_OF_RELOBJ) == 0)) {
1024                        Xword   tlsstatsize;
1025
1026                        /*
1027                         * This is the LE TLS reference model.  Static
1028                         * offset is hard-coded.
1029                         */
1030                        tlsstatsize = S_ROUND(ofl->ofl_tlsphdr->p_memsz,
1031                            M_TLSSTATALIGN);
1032                        value = tlsstatsize - value;
1033
1034                        /*
1035                         * Since this code is fixed up, it assumes a
1036                         * negative offset that can be added to the
1037                         * thread pointer.
1038                         */
1039                        if ((arsp->rel_rtype == R_386_TLS_LDO_32) ||
1040                            (arsp->rel_rtype == R_386_TLS_LE))
1041                                value = -value;
1042                }
1043
1044                if (arsp->rel_isdesc->is_file)
1045                        ifl_name = arsp->rel_isdesc->is_file->ifl_name;
1046                else
1047                        ifl_name = MSG_INTL(MSG_STR_NULL);
1048
1049                /*
1050                 * Make sure we have data to relocate.  Compiler and assembler
1051                 * developers have been known to generate relocations against
1052                 * invalid sections (normally .bss), so for their benefit give
1053                 * them sufficient information to help analyze the problem.
1054                 * End users should never see this.
1055                 */
1056                if (arsp->rel_isdesc->is_indata->d_buf == 0) {
1057                        Conv_inv_buf_t  inv_buf;
1058
1059                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_EMPTYSEC),
1060                            conv_reloc_386_type(arsp->rel_rtype, 0, &inv_buf),
1061                            ifl_name, ld_reloc_sym_name(arsp),
1062                            EC_WORD(arsp->rel_isdesc->is_scnndx),
1063                            arsp->rel_isdesc->is_name);
1064                        return (S_ERROR);
1065                }
1066
1067                /*
1068                 * Get the address of the data item we need to modify.
1069                 */
1070                addr = (uchar_t *)((uintptr_t)arsp->rel_roffset +
1071                    (uintptr_t)_elf_getxoff(arsp->rel_isdesc->is_indata));
```



```
1073                DBG_CALL(Dbg_reloc_doact(ofl->ofl_lml, ELF_DBG_LD_ACT,
1074                    M_MACH, SHT_REL, arsp, EC_NATPTR(addr), value,
1075                    ld_reloc_sym_name));
1076                addr += (uintptr_t)osp->os_outdata->d_buf;
1077
1078                if ((((uintptr_t)addr - (uintptr_t)ofl->ofl_nehdr) >
1079                    ofl->ofl_size) || (arsp->rel_roffset >
1080                    osp->os_shdr->sh_size)) {
1081                        Conv_inv_buf_t  inv_buf;
1082                        int             class;
1083
1084                        if (((uintptr_t)addr - (uintptr_t)ofl->ofl_nehdr) >
1085                            ofl->ofl_size)
1086                                class = ERR_FATAL;
1087                        else
1088                                class = ERR_WARNING;
1089
1090                        ld_eprintf(ofl, class, MSG_INTL(MSG_REL_INVALOFFSET),
1091                            conv_reloc_386_type(arsp->rel_rtype, 0, &inv_buf),
1092                            ifl_name, EC_WORD(arsp->rel_isdesc->is_scnndx),
1093                            arsp->rel_isdesc->is_name, ld_reloc_sym_name(arsp),
1094                            EC_ADDR((uintptr_t)addr -
1095                            (uintptr_t)ofl->ofl_nehdr));
1096
1097                        if (class == ERR_FATAL) {
1098                                return_code = S_ERROR;
1099                                continue;
1100                        }
1101                }
1102
1103                /*
1104                 * The relocation is additive.  Ignore the previous symbol
1105                 * value if this local partial symbol is expanded.
1106                 */
1107                if (moved)
1108                        value -= *addr;
1109
1110                /*
1111                 * If we have a replacement value for the relocation
1112                 * target, put it in place now.
1113                 */
1114                if (arsp->rel_flags & FLG_REL_NADDEND) {
1115                        Xword addend = arsp->rel_raddend;
1116
1117                        if (ld_reloc_targval_set(ofl, arsp, addr, addend) == 0)
1118                                return (S_ERROR);
1119                }
1120
1121                /*
1122                 * If '-z noreloc' is specified - skip the do_reloc_ld stage.
1123                 */
1124                if (OFL_DO_RELOC(ofl)) {
1125                        if (do_reloc_ld(arsp, addr, &value, ld_reloc_sym_name,
1126                            ifl_name, OFL_SWAP_RELOC_DATA(ofl, arsp),
1127                            ofl->ofl_lml) == 0) {
1128                                ofl->ofl_flags |= FLG_OF_FATAL;
1129                                return_code = S_ERROR;
1130                        }
1131                }
1132        }
1133        return (return_code);
1134 }
1135
1136 /*
1137  * Add an output relocation record.
1138  */
```

```
1139 static uintptr_t
1140 ld_add_outrel(Word flags, Rel_desc *rsp, Ofl_desc *ofl)
1141 {
1142         Rel_desc        *orsp;
1143         Sym_desc        *sdp = rsp->rel_sym;
1144
1145         /*
1146          * Static executables *do not* want any relocations against them.
1147          * Since our engine still creates relocations against a WEAK UNDEFINED
1148          * symbol in a static executable, it's best to disable them here
1149          * instead of through out the relocation code.
1150          */
1151         if (OFL_IS_STATIC_EXEC(ofl))
1152                 return (1);
1153
1154         /*
1155          * If the symbol will be reduced, we can't leave outstanding
1156          * relocations against it, as nothing will ever be able to satisfy them
1157          * (and the symbol won't be in .dynsym
1158          */
1159         if ((sdp != NULL) &&
1160             (sdp->sd_sym->st_shndx == SHN_UNDEF) &&
1161             (rsp->rel_rtype != M_R_NONE) &&
1162             (rsp->rel_rtype != M_R_RELATIVE)) {
1163                 if (ld_sym_reducable(ofl, sdp))
1164                         return (1);
1165         }
1166 #endif /* ! codereview */
1167
1168         /*
1169          * If we are adding a output relocation against a section
1170          * symbol (non-RELATIVE) then mark that section.  These sections
1171          * will be added to the .dynsym symbol table.
1172          */
1173         if (sdp && (rsp->rel_rtype != M_R_RELATIVE) &&
1174             ((flags & FLG_REL_SCNNDX) ||
1175             (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION))) {
1176
1177                 /*
1178                  * If this is a COMMON symbol - no output section
1179                  * exists yet - (it's created as part of sym_validate()).
1180                  * So - we mark here that when it's created it should
1181                  * be tagged with the FLG_OS_OUTREL flag.
1182                  */
1183                 if ((sdp->sd_flags & FLG_SY_SPECSEC) &&
1184                     (sdp->sd_sym->st_shndx == SHN_COMMON)) {
1185                         if (ELF_ST_TYPE(sdp->sd_sym->st_info) != STT_TLS)
1186                                 ofl->ofl_flags1 |= FLG_OF1_BSSOREL;
1187                         else
1188                                 ofl->ofl_flags1 |= FLG_OF1_TLSOREL;
1189                 } else {
1190                         Os_desc *osp;
1191                         Is_desc *isp = sdp->sd_isc;
1192
1193                         if (isp && ((osp = isp->is_osdesc) != NULL) &&
1194                             ((osp->os_flags & FLG_OS_OUTREL) == 0)) {
1195                                 ofl->ofl_dynshdrcnt++;
1196                                 osp->os_flags |= FLG_OS_OUTREL;
1197                         }
1198                 }
1199         }
1200
1201         /* Enter it into the output relocation cache */
1202         if ((orsp = ld_reloc_enter(ofl, &ofl->ofl_outrels, rsp, flags)) == NULL)
1203                 return (S_ERROR);
```



```
1205         if (flags & FLG_REL_GOT)
1206                 ofl->ofl_relocgotsz += (Xword)sizeof (Rel);
1207         else if (flags & FLG_REL_PLT)
1208                 ofl->ofl_relocpltsz += (Xword)sizeof (Rel);
1209         else if (flags & FLG_REL_BSS)
1210                 ofl->ofl_relocbsssz += (Xword)sizeof (Rel);
1211         else if (flags & FLG_REL_NOINFO)
1212                 ofl->ofl_relocrelsz += (Xword)sizeof (Rel);
1213         else
1214                 RELAUX_GET_OSDESC(orsp)->os_szoutrels += (Xword)sizeof (Rel);
1215
1216         if (orsp->rel_rtype == M_R_RELATIVE)
1217                 ofl->ofl_relocrelcnt++;
1218
1219         /*
1220          * We don't perform sorting on PLT relocations because
1221          * they have already been assigned a PLT index and if we
1222          * were to sort them we would have to re-assign the plt indexes.
1223          */
1224         if (!(flags & FLG_REL_PLT))
1225                 ofl->ofl_reloccnt++;
1226
1227         /*
1228          * Insure a GLOBAL_OFFSET_TABLE is generated if required.
1229          */
1230         if (IS_GOT_REQUIRED(orsp->rel_rtype))
1231                 ofl->ofl_flags |= FLG_OF_BLDGOT;
1232
1233         /*
1234          * Identify and possibly warn of a displacement relocation.
1235          */
1236         if (orsp->rel_flags & FLG_REL_DISP) {
1237                 ofl->ofl_dtflags_1 |= DF_1_DISPRELPND;
1238
1239                 if (ofl->ofl_flags & FLG_OF_VERBOSE)
1240                         ld_disp_errmsg(MSG_INTL(MSG_REL_DISPREL4), orsp, ofl);
1241         }
1242         DBG_CALL(Dbg_reloc_ors_entry(ofl->ofl_lml, ELF_DBG_LD, SHT_REL,
1243             M_MACH, orsp));
1244         return (1);
1245 }
1246
1247 /*
1248  * process relocation for a LOCAL symbol
1249  */
1250 static uintptr_t
1251 ld_reloc_local(Rel_desc * rsp, Ofl_desc * ofl)
1252 {
1253         ofl_flag_t      flags = ofl->ofl_flags;
1254         Sym_desc        *sdp = rsp->rel_sym;
1255         Word            shndx = sdp->sd_sym->st_shndx;
1256
1257         /*
1258          * if ((shared object) and (not pc relative relocation) and
1259          *    (not against ABS symbol))
1260          * then
1261          *      build R_386_RELATIVE
1262          * fi
1263          */
1264         if ((flags & FLG_OF_SHAROBJ) && (rsp->rel_flags & FLG_REL_LOAD) &&
1265             !(IS_PC_RELATIVE(rsp->rel_rtype)) && !(IS_SIZE(rsp->rel_rtype)) &&
1266             !(IS_GOT_BASED(rsp->rel_rtype)) &&
1267             !(rsp->rel_isdesc != NULL &&
1268             (rsp->rel_isdesc->is_shdr->sh_type == SHT_SUNW_dof)) &&
1269             (((sdp->sd_flags & FLG_SY_SPECSEC) == 0) ||
1270             (shndx != SHN_ABS) || (sdp->sd_aux && sdp->sd_aux->sa_symspec))) {
```

```
1271                Word    ortype = rsp->rel_rtype;

1273                rsp->rel_rtype = R_386_RELATIVE;
1274                if (ld_add_outrel(NULL, rsp, ofl) == S_ERROR)
1275                        return (S_ERROR);
1276                rsp->rel_rtype = ortype;
1277        }

1279        /*
1280         * If the relocation is against a 'non-allocatable' section
1281         * and we can not resolve it now - then give a warning
1282         * message.
1283         *
1284         * We can not resolve the symbol if either:
1285         *      a) it's undefined
1286         *      b) it's defined in a shared library and a
1287         *         COPY relocation hasn't moved it to the executable
1288         *
1289         * Note: because we process all of the relocations against the
1290         *      text segment before any others - we know whether
1291         *      or not a copy relocation will be generated before
1292         *      we get here (see reloc_init()->reloc_segments()).
1293         */
1294        if (!(rsp->rel_flags & FLG_REL_LOAD) &&
1295            ((shndx == SHN_UNDEF) ||
1296            ((sdp->sd_ref == REF_DYN_NEED) &&
1297            ((sdp->sd_flags & FLG_SY_MVTOCOMM) == 0)))) {
1298                Conv_inv_buf_t  inv_buf;
1299                Os_desc         *osp = RELAUX_GET_OSDESC(rsp);

1301                /*
1302                 * If the relocation is against a SHT_SUNW_ANNOTATE
1303                 * section - then silently ignore that the relocation
1304                 * can not be resolved.
1305                 */
1306                if (osp && (osp->os_shdr->sh_type == SHT_SUNW_ANNOTATE))
1307                        return (0);
1308                ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_REL_EXTERNSYM),
1309                    conv_reloc_386_type(rsp->rel_rtype, 0, &inv_buf),
1310                    rsp->rel_isdesc->is_file->ifl_name,
1311                    ld_reloc_sym_name(rsp), osp->os_name);
1312                return (1);
1313        }

1315        /*
1316         * Perform relocation.
1317         */
1318        return (ld_add_actrel(NULL, rsp, ofl));
1319 }

1321 static uintptr_t
1322 ld_reloc_TLS(Boolean local, Rel_desc * rsp, Ofl_desc * ofl)
1323 {
1324        Word            rtype = rsp->rel_rtype;
1325        Sym_desc        *sdp = rsp->rel_sym;
1326        ofl_flag_t      flags = ofl->ofl_flags;
1327        Gotndx          *gnp;

1329        /*
1330         * If we're building an executable - use either the IE or LE access
1331         * model.  If we're building a shared object process any IE model.
1332         */
1333        if ((flags & FLG_OF_EXEC) || (IS_TLS_IE(rtype))) {
1334                /*
1335                 * Set the DF_STATIC_TLS flag.
1336                 */
```



```
1337                ofl->ofl_dtflags |= DF_STATIC_TLS;

1339                if (!local || ((flags & FLG_OF_EXEC) == 0)) {
1340                        /*
1341                         * Assign a GOT entry for static TLS references.
1342                         */
1343                        if ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs,
1344                            GOT_REF_TLSIE, ofl, NULL)) == NULL) {

1346                                if (ld_assign_got_TLS(local, rsp, ofl, sdp,
1347                                    gnp, GOT_REF_TLSIE, FLG_REL_STLS,
1348                                    rtype, R_386_TLS_TPOFF, NULL) == S_ERROR)
1349                                        return (S_ERROR);
1350                        }

1352                        /*
1353                         * IE access model.
1354                         */
1355                        if (IS_TLS_IE(rtype)) {
1356                                if (ld_add_actrel(FLG_REL_STLS,
1357                                    rsp, ofl) == S_ERROR)
1358                                        return (S_ERROR);

1360                                /*
1361                                 * A non-pic shared object needs to adjust the
1362                                 * active relocation (indntpoff).
1363                                 */
1364                                if (((flags & FLG_OF_EXEC) == 0) &&
1365                                    (rtype == R_386_TLS_IE)) {
1366                                        rsp->rel_rtype = R_386_RELATIVE;
1367                                        return (ld_add_outrel(NULL, rsp, ofl));
1368                                }
1369                                return (1);
1370                        }

1372                        /*
1373                         * Fixups are required for other executable models.
1374                         */
1375                        return (ld_add_actrel((FLG_REL_TLSFIX | FLG_REL_STLS),
1376                            rsp, ofl));
1377                }

1379                /*
1380                 * LE access model.
1381                 */
1382                if (IS_TLS_LE(rtype) || (rtype == R_386_TLS_LDO_32))
1383                        return (ld_add_actrel(FLG_REL_STLS, rsp, ofl));

1385                return (ld_add_actrel((FLG_REL_TLSFIX | FLG_REL_STLS),
1386                    rsp, ofl));
1387        }

1389        /*
1390         * Building a shared object.
1391         *
1392         * Assign a GOT entry for a dynamic TLS reference.
1393         */
1394        if (IS_TLS_LD(rtype) && ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs,
1395            GOT_REF_TLSLD, ofl, NULL)) == NULL)) {

1397                if (ld_assign_got_TLS(local, rsp, ofl, sdp, gnp, GOT_REF_TLSLD,
1398                    FLG_REL_MTLS, rtype, R_386_TLS_DTPMOD32, NULL) == S_ERROR)
1399                        return (S_ERROR);

1401        } else if (IS_TLS_GD(rtype) && ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs,
1402            GOT_REF_TLSGD, ofl, NULL)) == NULL)) {
```

```
1404                 if (ld_assign_got_TLS(local, rsp, ofl, sdp, gnp, GOT_REF_TLSGD,
1405                     FLG_REL_DTLS, rtype, R_386_TLS_DTPMOD32,
1406                     R_386_TLS_DTPOFF32) == S_ERROR)
1407                         return (S_ERROR);
1408         }
1410         /*
1411          * For GD/LD TLS reference - TLS_{GD,LD}_CALL, this will eventually
1412          * cause a call to __tls_get_addr().  Convert this relocation to that
1413          * symbol now, and prepare for the PLT magic.
1414          */
1415         if ((rtype == R_386_TLS_GD_PLT) || (rtype == R_386_TLS_LDM_PLT)) {
1416                 Sym_desc        *tlsgetsym;
1418                 if ((tlsgetsym = ld_sym_add_u(MSG_ORIG(MSG_SYM_TLSGETADDR_UU),
1419                     ofl, MSG_STR_TLSREL)) == (Sym_desc *)S_ERROR)
1420                         return (S_ERROR);
1422                 rsp->rel_sym = tlsgetsym;
1423                 rsp->rel_rtype = R_386_PLT32;
1425                 if (ld_reloc_plt(rsp, ofl) == S_ERROR)
1426                         return (S_ERROR);
1428                 rsp->rel_sym = sdp;
1429                 rsp->rel_rtype = rtype;
1430                 return (1);
1431         }
1433         if (IS_TLS_LD(rtype))
1434                 return (ld_add_actrel(FLG_REL_MTLS, rsp, ofl));
1436         return (ld_add_actrel(FLG_REL_DTLS, rsp, ofl));
1437 }
1439 /* ARGSUSED4 */
1440 static uintptr_t
1441 ld_assign_got_ndx(Alist **alpp, Gotndx *pgnp, Gotref gref, Ofl_desc *ofl,
1442     Rel_desc *rsp, Sym_desc *sdp)
1443 {
1444         Gotndx  gn, *gnp;
1445         uint_t  gotents;
1447         if (pgnp)
1448                 return (1);
1450         if ((gref == GOT_REF_TLSGD) || (gref == GOT_REF_TLSLD))
1451                 gotents = 2;
1452         else
1453                 gotents = 1;
1455         gn.gn_addend = 0;
1456         gn.gn_gotndx = ofl->ofl_gotcnt;
1457         gn.gn_gotref = gref;
1459         ofl->ofl_gotcnt += gotents;
1461         if (gref == GOT_REF_TLSLD) {
1462                 if (ofl->ofl_tlsldgotndx == NULL) {
1463                         if ((gnp = libld_malloc(sizeof (Gotndx))) == NULL)
1464                                 return (S_ERROR);
1465                         (void) memcpy(gnp, &gn, sizeof (Gotndx));
1466                         ofl->ofl_tlsldgotndx = gnp;
1467                 }
1468                 return (1);
```



```
1469         }
1471         /*
1472          * GOT indexes are maintained on an Alist, where there is typically
1473          * only one index.  The usage of this list is to scan the list to find
1474          * an index, and then apply that index immediately to a relocation.
1475          * Thus there are no external references to these GOT index structures
1476          * that can be compromised by the Alist being reallocated.
1477          */
1478         if (alist_append(alpp, &gn, sizeof (Gotndx), AL_CNT_SDP_GOT) == NULL)
1479                 return (S_ERROR);
1481         return (1);
1482 }
1484 static void
1485 ld_assign_plt_ndx(Sym_desc * sdp, Ofl_desc *ofl)
1486 {
1487         sdp->sd_aux->sa_PLTndx = 1 + ofl->ofl_pltcnt++;
1488         sdp->sd_aux->sa_PLTGOTndx = ofl->ofl_gotcnt++;
1489         ofl->ofl_flags |= FLG_OF_BLDGOT;
1490 }
1492 /*
1493  * Initializes .got[0] with the _DYNAMIC symbol value.
1494  */
1495 static uintptr_t
1496 ld_fillin_gotplt(Ofl_desc *ofl)
1497 {
1498         ofl_flag_t      flags = ofl->ofl_flags;
1499         int             bswap = (ofl->ofl_flags1 & FLG_OF1_ENCDIFF) != 0;
1501         if (ofl->ofl_osgot) {
1502                 Sym_desc        *sdp;
1504                 if ((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_DYNAMIC_U),
1505                     SYM_NOHASH, NULL, ofl)) != NULL) {
1506                         uchar_t *genptr;
1508                         genptr = ((uchar_t *)ofl->ofl_osgot->os_outdata->d_buf +
1509                             (M_GOT_XDYNAMIC * M_GOT_ENTSIZE));
1510                         /* LINTED */
1511                         *(Word *)genptr = (Word)sdp->sd_sym->st_value;
1512                         if (bswap)
1513                                 /* LINTED */
1514                                 *(Word *)genptr =
1515                                     /* LINTED */
1516                                     ld_bswap_Word(*(Word *)genptr);
1517                 }
1518         }
1520         /*
1521          * Fill in the reserved slot in the procedure linkage table the first
1522          * entry is:
1523          *  if (building a.out) {
1524          *      PUSHL   got[1]              # the address of the link map entry
1525          *      JMP *   got[2]              # the address of rtbinder
1526          *  } else {
1527          *      PUSHL   got[1]@GOT(%ebx)    # the address of the link map entry
1528          *      JMP *   got[2]@GOT(%ebx)    # the address of rtbinder
1529          *  }
1530          */
1531         if ((flags & FLG_OF_DYNAMIC) && ofl->ofl_osplt) {
1532                 uchar_t *pltent;
1534                 pltent = (uchar_t *)ofl->ofl_osplt->os_outdata->d_buf;
```

```
1535                 if (!(flags & FLG_OF_SHAROBJ)) {
1536                         pltent[0] = M_SPECIAL_INST;
1537                         pltent[1] = M_PUSHL_DISP;
1538                         pltent += 2;
1539                         /* LINTED */
1540                         *(Word *)pltent = (Word)(ofl->ofl_osgot->os_shdr->
1541                             sh_addr + M_GOT_XLINKMAP * M_GOT_ENTSIZE);
1542                         if (bswap)
1543                                 /* LINTED */
1544                                 *(Word *)pltent =
1545                                     /* LINTED */
1546                                     ld_bswap_Word(*(Word *)pltent);
1547                         pltent += 4;
1548                         pltent[0] = M_SPECIAL_INST;
1549                         pltent[1] = M_JMP_DISP_IND;
1550                         pltent += 2;
1551                         /* LINTED */
1552                         *(Word *)pltent = (Word)(ofl->ofl_osgot->os_shdr->
1553                             sh_addr + M_GOT_XRTLD * M_GOT_ENTSIZE);
1554                         if (bswap)
1555                                 /* LINTED */
1556                                 *(Word *)pltent =
1557                                     /* LINTED */
1558                                     ld_bswap_Word(*(Word *)pltent);
1559                 } else {
1560                         pltent[0] = M_SPECIAL_INST;
1561                         pltent[1] = M_PUSHL_REG_DISP;
1562                         pltent += 2;
1563                         /* LINTED */
1564                         *(Word *)pltent = (Word)(M_GOT_XLINKMAP *
1565                             M_GOT_ENTSIZE);
1566                         if (bswap)
1567                                 /* LINTED */
1568                                 *(Word *)pltent =
1569                                     /* LINTED */
1570                                     ld_bswap_Word(*(Word *)pltent);
1571                         pltent += 4;
1572                         pltent[0] = M_SPECIAL_INST;
1573                         pltent[1] = M_JMP_REG_DISP_IND;
1574                         pltent += 2;
1575                         /* LINTED */
1576                         *(Word *)pltent = (Word)(M_GOT_XRTLD *
1577                             M_GOT_ENTSIZE);
1578                         if (bswap)
1579                                 /* LINTED */
1580                                 *(Word *)pltent =
1581                                     /* LINTED */
1582                                     ld_bswap_Word(*(Word *)pltent);
1583                 }
1584         }
1585         return (1);
1586 }



1590 /*
1591  * Template for generating "void (*)(void)" function
1592  */
1593 static const uchar_t nullfunc_tmpl[] = {        /* IA32 */
1594 /* 0x00 */      0xc3                            /* ret */
1595 };



1599 /*
1600  * Function used to provide fill padding in SHF_EXECINSTR sections
```



```
1601  *
1602  * entry:
1603  *
1604  *      base - base address of section being filled
1605  *      offset - starting offset for fill within memory referenced by base
1606  *      cnt - # bytes to be filled
1607  *
1608  * exit:
1609  *      The fill has been completed.
1610  */
1611 static void
1612 execfill(void *base, off_t off, size_t cnt)
1613 {
1614         /*
1615          * 0x90 is an X86 NOP instruction in both 32 and 64-bit worlds.
1616          * There are no alignment constraints.
1617          */
1618         (void) memset(off + (char *)base, 0x90, cnt);
1619 }


1622 /*
1623  * Return the ld_targ definition for this target.
1624  */
1625 const Target *
1626 ld_targ_init_x86(void)
1627 {
1628         static const Target _ld_targ = {
1629                 {                       /* Target_mach */
1630                         M_MACH,                 /* m_mach */
1631                         M_MACHPLUS,             /* m_machplus */
1632                         M_FLAGSPLUS,            /* m_flagsplus */
1633                         M_CLASS,                /* m_class */
1634                         M_DATA,                 /* m_data */

1636                         M_SEGM_ALIGN,           /* m_segm_align */
1637                         M_SEGM_ORIGIN,          /* m_segm_origin */
1638                         M_SEGM_AORIGIN,         /* m_segm_aorigin */
1639                         M_DATASEG_PERM,         /* m_dataseg_perm */
1640                         M_STACK_PERM,           /* m_stack_perm */
1641                         M_WORD_ALIGN,           /* m_word_align */
1642                         MSG_ORIG(MSG_PTH_RTLD), /* m_def_interp */

1644                         /* Relocation type codes */
1645                         M_R_ARRAYADDR,          /* m_r_arrayaddr */
1646                         M_R_COPY,               /* m_r_copy */
1647                         M_R_GLOB_DAT,           /* m_r_glob_dat */
1648                         M_R_JMP_SLOT,           /* m_r_jmp_slot */
1649                         M_R_NUM,                /* m_r_num */
1650                         M_R_NONE,               /* m_r_none */
1651                         M_R_RELATIVE,           /* m_r_relative */
1652                         M_R_REGISTER,           /* m_r_register */

1654                         /* Relocation related constants */
1655                         M_REL_DT_COUNT,         /* m_rel_dt_count */
1656                         M_REL_DT_ENT,           /* m_rel_dt_ent */
1657                         M_REL_DT_SIZE,          /* m_rel_dt_size */
1658                         M_REL_DT_TYPE,          /* m_rel_dt_type */
1659                         M_REL_SHT_TYPE,         /* m_rel_sht_type */

1661                         /* GOT related constants */
1662                         M_GOT_ENTSIZE,          /* m_got_entsize */
1663                         M_GOT_XNumber,          /* m_got_xnumber */

1665                         /* PLT related constants */
1666                         M_PLT_ALIGN,            /* m_plt_align */
```

```
1667                        M_PLT_ENTSIZE,          /* m_plt_entsize */
1668                        M_PLT_RESERVSZ,         /* m_plt_reservsz */
1669                        M_PLT_SHF_FLAGS,        /* m_plt_shf_flags */

1671                        /* Section type of .eh_frame/.eh_frame_hdr sections */
1672                        SHT_PROGBITS,           /* m_sht_unwind */

1674                        M_DT_REGISTER,          /* m_dt_register */
1675                },
1676                {                       /* Target_machid */
1677                        M_ID_ARRAY,             /* id_array */
1678                        M_ID_BSS,               /* id_bss */
1679                        M_ID_CAP,               /* id_cap */
1680                        M_ID_CAPINFO,           /* id_capinfo */
1681                        M_ID_CAPCHAIN,          /* id_capchain */
1682                        M_ID_DATA,              /* id_data */
1683                        M_ID_DYNAMIC,           /* id_dynamic */
1684                        M_ID_DYNSORT,           /* id_dynsort */
1685                        M_ID_DYNSTR,            /* id_dynstr */
1686                        M_ID_DYNSYM,            /* id_dynsym */
1687                        M_ID_DYNSYM_NDX,        /* id_dynsym_ndx */
1688                        M_ID_GOT,               /* id_got */
1689                        M_ID_UNKNOWN,           /* id_gotdata (unused) */
1690                        M_ID_HASH,              /* id_hash */
1691                        M_ID_INTERP,            /* id_interp */
1692                        M_ID_LBSS,              /* id_lbss */
1693                        M_ID_LDYNSYM,           /* id_ldynsym */
1694                        M_ID_NOTE,              /* id_note */
1695                        M_ID_NULL,              /* id_null */
1696                        M_ID_PLT,               /* id_plt */
1697                        M_ID_REL,               /* id_rel */
1698                        M_ID_STRTAB,            /* id_strtab */
1699                        M_ID_SYMINFO,           /* id_syminfo */
1700                        M_ID_SYMTAB,            /* id_symtab */
1701                        M_ID_SYMTAB_NDX,        /* id_symtab_ndx */
1702                        M_ID_TEXT,              /* id_text */
1703                        M_ID_TLS,               /* id_tls */
1704                        M_ID_TLSBSS,            /* id_tlsbss */
1705                        M_ID_UNKNOWN,           /* id_unknown */
1706                        M_ID_UNWIND,            /* id_unwind */
1707                        M_ID_UNWINDHDR,         /* id_unwindhdr */
1708                        M_ID_USER,              /* id_user */
1709                        M_ID_VERSION,           /* id_version */
1710                },
1711                {                       /* Target_nullfunc */
1712                        nullfunc_tmpl,          /* nf_template */
1713                        sizeof (nullfunc_tmpl), /* nf_size */
1714                },
1715                {                       /* Target_fillfunc */
1716                        execfill                /* ff_execfill */
1717                },
1718                {                       /* Target_machrel */
1719                        reloc_table,

1721                        ld_init_rel,            /* mr_init_rel */
1722                        ld_mach_eflags,         /* mr_mach_eflags */
1723                        ld_mach_make_dynamic,   /* mr_mach_make_dynamic */
1724                        ld_mach_update_odynamic, /* mr_mach_update_odynamic */
1725                        ld_calc_plt_addr,       /* mr_calc_plt_addr */
1726                        ld_perform_outreloc,    /* mr_perform_outreloc */
1727                        ld_do_activerelocs,     /* mr_do_activerelocs */
1728                        ld_add_outrel,          /* mr_add_outrel */
1729                        NULL,                   /* mr_reloc_register */
1730                        ld_reloc_local,         /* mr_reloc_local */
1731                        NULL,                   /* mr_reloc_GOTOP */
1732                        ld_reloc_TLS,           /* mr_reloc_TLS */
```



```
1733                        NULL,                   /* mr_assign_got */
1734                        ld_find_got_ndx,        /* mr_find_got_ndx */
1735                        ld_calc_got_offset,     /* mr_calc_got_offset */
1736                        ld_assign_got_ndx,      /* mr_assign_got_ndx */
1737                        ld_assign_plt_ndx,      /* mr_assign_plt_ndx */
1738                        NULL,                   /* mr_allocate_got */
1739                        ld_fillin_gotplt,       /* mr_fillin_gotplt */
1740                },
1741                {                       /* Target_machsym */
1742                        NULL,                   /* ms_reg_check */
1743                        NULL,                   /* ms_mach_sym_typecheck */
1744                        NULL,                   /* ms_is_regsym */
1745                        NULL,                   /* ms_reg_find */
1746                        NULL                    /* ms_reg_enter */
1747                }
1748        };

1750        return (&_ld_targ);
1751 }
```

```
**********************************************************
   64852 Wed May 22 03:21:45 2019
new/usr/src/cmd/sgs/libld/common/machrel.sparc.c
11057 hidden undefined weak symbols should not leave relocations
11058 libld entrance descriptor assertions get NDEBUG check backwards
**********************************************************
______unchanged_portion_omitted_

 581 #endif /* _ELF64 */

 583 static uintptr_t
 584 ld_perform_outreloc(Rel_desc *orsp, Ofl_desc *ofl, Boolean *remain_seen)
 585 {
 586         Os_desc         *relosp, *osp = NULL;
 587         Xword           ndx, roffset, value;
 588         Sxword          raddend;
 589         const Rel_entry *rep;
 590         Rela            rea;
 591         char            *relbits;
 592         Sym_desc        *sdp, *psym = NULL;
 593         int             sectmoved = 0;
 594         Word            dtflags1 = ofl->ofl_dtflags_1;
 595         ofl_flag_t      flags = ofl->ofl_flags;

 597         raddend = orsp->rel_raddend;
 598         sdp = orsp->rel_sym;

 600         /*
 601          * Special case, a regsiter symbol associated with symbol
 602          * index 0 is initialized (i.e. relocated) to a constant
 603          * in the r_addend field rather than to a symbol value.
 604          */
 605         if ((orsp->rel_rtype == M_R_REGISTER) && !sdp) {
 606                 relosp = ofl->ofl_osrel;
 607                 relbits = (char *)relosp->os_outdata->d_buf;

 609                 rea.r_info = ELF_R_INFO(0,
 610                     ELF_R_TYPE_INFO(RELAUX_GET_TYPEDATA(orsp),
 611                     orsp->rel_rtype));
 612                 rea.r_offset = orsp->rel_roffset;
 613                 rea.r_addend = raddend;
 614                 DBG_CALL(Dbg_reloc_out(ofl, ELF_DBG_LD, SHT_RELA, &rea,
 615                     relosp->os_name, ld_reloc_sym_name(orsp)));

 617                 assert(relosp->os_szoutrels <= relosp->os_shdr->sh_size);
 618                 (void) memcpy((relbits + relosp->os_szoutrels),
 619                     (char *)&rea, sizeof (Rela));
 620                 relosp->os_szoutrels += (Xword)sizeof (Rela);

 622                 return (1);
 623         }

 625         /*
 626          * If the section this relocation is against has been discarded
 627          * (-zignore), then also discard (skip) the relocation itself.
 628          */
 629         if (orsp->rel_isdesc && ((orsp->rel_flags &
 630             (FLG_REL_GOT | FLG_REL_BSS | FLG_REL_PLT | FLG_REL_NOINFO)) == 0) &&
 631             (orsp->rel_isdesc->is_flags & FLG_IS_DISCARD)) {
 632                 DBG_CALL(Dbg_reloc_discard(ofl->ofl_lml, M_MACH, orsp));
 633                 return (1);
 634         }

 636         /*
 637          * If this is a relocation against a move table, or expanded move
 638          * table, adjust the relocation entries.
```



```
 639          */
 640         if (RELAUX_GET_MOVE(orsp))
 641                 ld_adj_movereloc(ofl, orsp);

 643         /*
 644          * If this is a relocation against a section then we need to adjust the
 645          * raddend field to compensate for the new position of the input section
 646          * within the new output section.
 647          */
 648         if (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION) {
 649                 if (ofl->ofl_parsyms &&
 650                     (sdp->sd_isc->is_flags & FLG_IS_RELUPD) &&
 651                     (psym = ld_am_I_partial(orsp, orsp->rel_raddend))) {
 652                         /*
 653                          * If the symbol is moved, adjust the value
 654                          */
 655                         DBG_CALL(Dbg_move_outsctadj(ofl->ofl_lml, psym));
 656                         sectmoved = 1;
 657                         if (ofl->ofl_flags & FLG_OF_RELOBJ)
 658                                 raddend = psym->sd_sym->st_value;
 659                         else
 660                                 raddend = psym->sd_sym->st_value -
 661                                     psym->sd_isc->is_osdesc->os_shdr->sh_addr;
 662                         /* LINTED */
 663                         raddend += (Off)_elf_getxoff(psym->sd_isc->is_indata);
 664                         if (psym->sd_isc->is_shdr->sh_flags & SHF_ALLOC)
 665                                 raddend +=
 666                                     psym->sd_isc->is_osdesc->os_shdr->sh_addr;
 667                 } else {
 668                         /* LINTED */
 669                         raddend += (Off)_elf_getxoff(sdp->sd_isc->is_indata);
 670                         if (sdp->sd_isc->is_shdr->sh_flags & SHF_ALLOC)
 671                                 raddend +=
 672                                     sdp->sd_isc->is_osdesc->os_shdr->sh_addr;
 673                 }
 674         }

 676         value = sdp->sd_sym->st_value;

 678         if (orsp->rel_flags & FLG_REL_GOT) {
 679                 osp = ofl->ofl_osgot;
 680                 roffset = ld_calc_got_offset(orsp, ofl);

 682         } else if (orsp->rel_flags & FLG_REL_PLT) {
 683                 osp = ofl->ofl_osplt;
 684                 plt_entry(ofl, sdp->sd_aux->sa_PLTndx, &roffset, &raddend);
 685         } else if (orsp->rel_flags & FLG_REL_BSS) {
 686                 /*
 687                  * This must be a R_SPARC_COPY.  For these set the roffset to
 688                  * point to the new symbols location.
 689                  */
 690                 osp = ofl->ofl_isbss->is_osdesc;
 691                 roffset = (Xword)value;

 693                 /*
 694                  * The raddend doesn't mean anything in an R_SPARC_COPY
 695                  * relocation.  Null it out because it can confuse people.
 696                  */
 697                 raddend = 0;
 698         } else if (orsp->rel_flags & FLG_REL_REG) {
 699                 /*
 700                  * The offsets of relocations against register symbols
 701                  * identifiy the register directly - so the offset
 702                  * does not need to be adjusted.
 703                  */
 704                 roffset = orsp->rel_roffset;
```

```
705         } else {
706                 osp = RELAUX_GET_OSDESC(orsp);
707
708                 /*
709                  * Calculate virtual offset of reference point; equals offset
710                  * into section + vaddr of section for loadable sections, or
711                  * offset plus section displacement for nonloadable sections.
712                  */
713                 roffset = orsp->rel_roffset +
714                     (Off)_elf_getxoff(orsp->rel_isdesc->is_indata);
715                 if (!(ofl->ofl_flags & FLG_OF_RELOBJ))
716                         roffset += orsp->rel_isdesc->is_osdesc->
717                             os_shdr->sh_addr;
718         }
719
720         if ((osp == 0) || ((relosp = osp->os_relosdesc) == 0))
721                 relosp = ofl->ofl_osrel;
722
723         /*
724          * Verify that the output relocations offset meets the
725          * alignment requirements of the relocation being processed.
726          */
727         rep = &reloc_table[orsp->rel_rtype];
728         if (((flags & FLG_OF_RELOBJ) || !(dtflags1 & DF_1_NORELOC)) &&
729             !(rep->re_flags & FLG_RE_UNALIGN)) {
730                 if (((rep->re_fsize == 2) && (roffset & 0x1)) ||
731                     ((rep->re_fsize == 4) && (roffset & 0x3)) ||
732                     ((rep->re_fsize == 8) && (roffset & 0x7))) {
733                         Conv_inv_buf_t inv_buf;
734
735                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_NONALIGN),
736                             conv_reloc_SPARC_type(orsp->rel_rtype, 0, &inv_buf),
737                             orsp->rel_isdesc->is_file->ifl_name,
738                             ld_reloc_sym_name(orsp), EC_XWORD(roffset));
739                         return (S_ERROR);
740                 }
741         }
742
743         /*
744          * Assign the symbols index for the output relocation.  If the
745          * relocation refers to a SECTION symbol then it's index is based upon
746          * the output sections symbols index.  Otherwise the index can be
747          * derived from the symbols index itself.
748          */
749         if (orsp->rel_rtype == R_SPARC_RELATIVE)
750                 ndx = STN_UNDEF;
751         else if ((orsp->rel_flags & FLG_REL_SCNNDX) ||
752             (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION)) {
753                 if (sectmoved == 0) {
754                         /*
755                          * Check for a null input section. This can
756                          * occur if this relocation references a symbol
757                          * generated by sym_add_sym().
758                          */
759                         if (sdp->sd_isc && sdp->sd_isc->is_osdesc)
760                                 ndx = sdp->sd_isc->is_osdesc->os_identndx;
761                         else
762                                 ndx = sdp->sd_shndx;
763                 } else
764                         ndx = ofl->ofl_parexpnndx;
765         } else
766                 ndx = sdp->sd_symndx;
767
768         /*
769          * Add the symbols 'value' to the addend field.
770          */
```



```
771         if (orsp->rel_flags & FLG_REL_ADVAL)
772                 raddend += value;
773
774         /*
775          * The addend field for R_SPARC_TLS_DTPMOD32 and R_SPARC_TLS_DTPMOD64
776          * mean nothing.  The addend is propagated in the corresponding
777          * R_SPARC_TLS_DTPOFF* relocations.
778          */
779         if (orsp->rel_rtype == M_R_DTPMOD)
780                 raddend = 0;
781
782         /*
783          * Note that the other case which writes out the relocation, above, is
784          * M_R_REGISTER specific and so does not need this check.
785          */
786         if ((orsp->rel_rtype != M_R_NONE) &&
787             (orsp->rel_rtype != M_R_REGISTER) &&
788             (orsp->rel_rtype != M_R_RELATIVE)) {
789                 if (ndx == 0) {
790                         Conv_inv_buf_t  inv_buf;
791                         Is_desc *isp = orsp->rel_isdesc;
792
793                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_NOSYMBOL),
794                             conv_reloc_type(ofl->ofl_nehdr->e_machine,
795                             orsp->rel_rtype, 0, &inv_buf),
796                             isp->is_file->ifl_name, EC_WORD(isp->is_scnndx),
797                             isp->is_name, EC_XWORD(roffset));
798                         return (S_ERROR);
799                 }
800         }
801         relbits = (char *)relosp->os_outdata->d_buf;
802
803         rea.r_info = ELF_R_INFO(ndx,
804             ELF_R_TYPE_INFO(RELAUX_GET_TYPEDATA(orsp), orsp->rel_rtype));
805         rea.r_offset = roffset;
806         rea.r_addend = raddend;
807         DBG_CALL(Dbg_reloc_out(ofl, ELF_DBG_LD, SHT_RELA, &rea, relosp->os_name,
808             ld_reloc_sym_name(orsp)));
809
810         /*
811          * Assert we haven't walked off the end of our relocation table.
812          */
813         assert(relosp->os_szoutrels <= relosp->os_shdr->sh_size);
814
815         relbits = (char *)relosp->os_outdata->d_buf;
816 #endif /* ! codereview */
817         (void) memcpy((relbits + relosp->os_szoutrels),
818             (char *)&rea, sizeof (Rela));
819         relosp->os_szoutrels += (Xword)sizeof (Rela);
820
821         /*
822          * Determine if this relocation is against a non-writable, allocatable
823          * section.  If so we may need to provide a text relocation diagnostic.
824          */
825         ld_reloc_remain_entry(orsp, osp, ofl, remain_seen);
826         return (1);
827 }
828
829
830 /*
831  * Sparc Instructions for TLS processing
832  */
833 #if     defined(_ELF64)
834 #define TLS_GD_IE_LD    0xd0580000      /* ldx [%g0 + %g0], %o0 */
835 #else
```

```
#define TLS_GD_IE_LD    0xd0000000      /* ld [%g0 + %g0], %o0 */
#endif
#define TLS_GD_IE_ADD   0x9001c008      /* add %g7, %o0, %o0 */

#define TLS_GD_LE_XOR   0x80182000      /* xor %g0, 0, %g0 */
#define TLS_IE_LE_OR    0x80100000      /* or %g0, %o0, %o1 */
                                        /*  synthetic: mov %g0, %g0 */

#define TLS_LD_LE_CLRO0 0x90100000      /* clr  %o0 */

#define FM3_REG_MSK_RD  (0x1f << 25)    /* Formate (3) rd register mask */
                                        /*      bits 25->29 */
#define FM3_REG_MSK_RS1 (0x1f << 14)    /* Formate (3) rs1 register mask */
                                        /*      bits 14->18 */
#define FM3_REG_MSK_RS2 0x1f            /* Formate (3) rs2 register mask */
                                        /*      bits 0->4 */

#define REG_G7          7               /* %g7 register */

static Fixupret
tls_fixups(Ofl_desc *ofl, Rel_desc *arsp)
{
        Sym_desc        *sdp = arsp->rel_sym;
        Word            rtype = arsp->rel_rtype;
        Word            *offset, w;
        int             bswap = OFL_SWAP_RELOC_DATA(ofl, arsp);


        offset = (Word *)((uintptr_t)arsp->rel_roffset +
            (uintptr_t)_elf_getxoff(arsp->rel_isdesc->is_indata) +
            (uintptr_t)RELAUX_GET_OSDESC(arsp)->os_outdata->d_buf);

        if (sdp->sd_ref == REF_DYN_NEED) {
                /*
                 * IE reference model
                 */
                switch (rtype) {
                case R_SPARC_TLS_GD_HI22:
                        DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                            R_SPARC_TLS_IE_HI22, arsp,
                            ld_reloc_sym_name));
                        arsp->rel_rtype = R_SPARC_TLS_IE_HI22;
                        return (FIX_RELOC);

                case R_SPARC_TLS_GD_LO10:
                        DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                            R_SPARC_TLS_IE_LO10, arsp,
                            ld_reloc_sym_name));
                        arsp->rel_rtype = R_SPARC_TLS_IE_LO10;
                        return (FIX_RELOC);

                case R_SPARC_TLS_GD_ADD:
                        DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                            R_SPARC_NONE, arsp, ld_reloc_sym_name));
                        w = bswap ? ld_bswap_Word(*offset) : *offset;
                        w = (TLS_GD_IE_LD |
                            (w & (FM3_REG_MSK_RS1 | FM3_REG_MSK_RS2)));
                        *offset = bswap ? ld_bswap_Word(w) : w;
                        return (FIX_DONE);

                case R_SPARC_TLS_GD_CALL:
                        DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                            R_SPARC_NONE, arsp, ld_reloc_sym_name));
                        *offset = TLS_GD_IE_ADD;
                        if (bswap)
                                *offset = ld_bswap_Word(*offset);
```



```
                        return (FIX_DONE);
                }
                return (FIX_RELOC);
        }

        /*
         * LE reference model
         */
        switch (rtype) {
        case R_SPARC_TLS_IE_HI22:
        case R_SPARC_TLS_GD_HI22:
        case R_SPARC_TLS_LDO_HIX22:
                DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                    R_SPARC_TLS_LE_HIX22, arsp, ld_reloc_sym_name));
                arsp->rel_rtype = R_SPARC_TLS_LE_HIX22;
                return (FIX_RELOC);

        case R_SPARC_TLS_LDO_LOX10:
                DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                    R_SPARC_TLS_LE_LOX10, arsp, ld_reloc_sym_name));
                arsp->rel_rtype = R_SPARC_TLS_LE_LOX10;
                return (FIX_RELOC);

        case R_SPARC_TLS_IE_LO10:
        case R_SPARC_TLS_GD_LO10:
                /*
                 * Current instruction is:
                 *
                 *      or r1, %lo(x), r2
                 *           or
                 *      add r1, %lo(x), r2
                 *
                 * Need to udpate this to:
                 *
                 *      xor r1, %lox(x), r2
                 */
                DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                    R_SPARC_TLS_LE_LOX10, arsp, ld_reloc_sym_name));
                w = bswap ? ld_bswap_Word(*offset) : *offset;
                w = TLS_GD_LE_XOR |
                    (w & (FM3_REG_MSK_RS1 | FM3_REG_MSK_RD));
                *offset = bswap ? ld_bswap_Word(w) : w;
                arsp->rel_rtype = R_SPARC_TLS_LE_LOX10;
                return (FIX_RELOC);

        case R_SPARC_TLS_IE_LD:
        case R_SPARC_TLS_IE_LDX:
                /*
                 * Current instruction:
                 *      ld{x}   [r1 + r2], r3
                 *
                 * Need to update this to:
                 *
                 *      mov     r2, r3   (or  %g0, r2, r3)
                 */
                DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
                    R_SPARC_NONE, arsp, ld_reloc_sym_name));
                w = bswap ? ld_bswap_Word(*offset) : *offset;
                w = (w & (FM3_REG_MSK_RS2 | FM3_REG_MSK_RD)) | TLS_IE_LE_OR;
                *offset = bswap ? ld_bswap_Word(w) : w;
                return (FIX_DONE);

        case R_SPARC_TLS_LDO_ADD:
        case R_SPARC_TLS_GD_ADD:
                /*
                 * Current instruction is:
```

```
968                  *
969                  *      add gptr_reg, r2, r3
970                  *
971                  * Need to updated this to:
972                  *
973                  *      add %g7, r2, r3
974                  */
975                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
976                     R_SPARC_NONE, arsp, ld_reloc_sym_name));
977                 w = bswap ? ld_bswap_Word(*offset) : *offset;
978                 w = w & (~FM3_REG_MSK_RS1);
979                 w = w | (REG_G7 << 14);
980                 *offset = bswap ? ld_bswap_Word(w) : w;
981                 return (FIX_DONE);

983         case R_SPARC_TLS_LDM_CALL:
984                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
985                     R_SPARC_NONE, arsp, ld_reloc_sym_name));
986                 *offset = TLS_LD_LE_CLR0;
987                 if (bswap)
988                         *offset = ld_bswap_Word(*offset);
989                 return (FIX_DONE);

991         case R_SPARC_TLS_LDM_HI22:
992         case R_SPARC_TLS_LDM_LO10:
993         case R_SPARC_TLS_LDM_ADD:
994         case R_SPARC_TLS_IE_ADD:
995         case R_SPARC_TLS_GD_CALL:
996                 DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
997                     R_SPARC_NONE, arsp, ld_reloc_sym_name));
998                 *offset = M_NOP;
999                 if (bswap)
1000                        *offset = ld_bswap_Word(*offset);
1001                return (FIX_DONE);
1002        }
1003        return (FIX_RELOC);
1004 }

1006 #define GOTOP_ADDINST   0x80000000      /* add %g0, %g0, %g0 */

1008 static Fixupret
1009 gotop_fixups(Ofl_desc *ofl, Rel_desc *arsp)
1010 {
1011        Word            rtype = arsp->rel_rtype;
1012        Word            *offset, w;
1013        const char      *ifl_name;
1014        Conv_inv_buf_t  inv_buf;
1015        int             bswap;

1017        switch (rtype) {
1018        case R_SPARC_GOTDATA_OP_HIX22:
1019                DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1020                    R_SPARC_GOTDATA_HIX22, arsp, ld_reloc_sym_name));
1021                arsp->rel_rtype = R_SPARC_GOTDATA_HIX22;
1022                return (FIX_RELOC);

1024        case R_SPARC_GOTDATA_OP_LOX10:
1025                DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1026                    R_SPARC_GOTDATA_LOX10, arsp, ld_reloc_sym_name));
1027                arsp->rel_rtype = R_SPARC_GOTDATA_LOX10;
1028                return (FIX_RELOC);

1030        case R_SPARC_GOTDATA_OP:
1031                /*
1032                 * Current instruction:
1033                 *      ld{x}   [r1 + r2], r3
```



```
1034                 *
1035                 * Need to update this to:
1036                 *
1037                 *      add     r1, r2, r3
1038                 */
1039                DBG_CALL(Dbg_reloc_transition(ofl->ofl_lml, M_MACH,
1040                    R_SPARC_NONE, arsp, ld_reloc_sym_name));
1041                offset = (Word *)(uintptr_t)(arsp->rel_roffset +
1042                    _elf_getxoff(arsp->rel_isdesc->is_indata) +
1043                    (uintptr_t)RELAUX_GET_OSDESC(arsp)->os_outdata->d_buf);
1044                bswap = OFL_SWAP_RELOC_DATA(ofl, arsp);
1045                w = bswap ? ld_bswap_Word(*offset) : *offset;
1046                w = (w & (FM3_REG_MSK_RS1 |
1047                    FM3_REG_MSK_RS2 | FM3_REG_MSK_RD)) | GOTOP_ADDINST;
1048                *offset = bswap ? ld_bswap_Word(w) : w;
1049                return (FIX_DONE);
1050        }
1051        /*
1052         * We should not get here
1053         */
1054        if (arsp->rel_isdesc->is_file)
1055                ifl_name = arsp->rel_isdesc->is_file->ifl_name;
1056        else
1057                ifl_name = MSG_INTL(MSG_STR_NULL);

1059        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_BADGOTFIX),
1060            conv_reloc_SPARC_type(arsp->rel_rtype, 0, &inv_buf),
1061            ifl_name, ld_reloc_sym_name(arsp));

1063        assert(0);
1064        return (FIX_ERROR);
1065 }

1067 static uintptr_t
1068 ld_do_activerelocs(Ofl_desc *ofl)
1069 {
1070        Rel_desc        *arsp;
1071        Rel_cachebuf    *rcbp;
1072        Aliste          idx;
1073        uintptr_t       return_code = 1;
1074        ofl_flag_t      flags = ofl->ofl_flags;

1076        if (aplist_nitems(ofl->ofl_actrels.rc_list) != 0)
1077                DBG_CALL(Dbg_reloc_doact_title(ofl->ofl_lml));

1079        /*
1080         * Process active relocations.
1081         */
1082        REL_CACHE_TRAVERSE(&ofl->ofl_actrels, idx, rcbp, arsp) {
1083                uchar_t         *addr;
1084                Xword           value;
1085                Sym_desc        *sdp;
1086                const char      *ifl_name;
1087                Xword           refaddr;
1088                Os_desc         *osp;

1090                /*
1091                 * If the section this relocation is against has been discarded
1092                 * (-zignore), then discard (skip) the relocation itself.
1093                 */
1094                if ((arsp->rel_isdesc->is_flags & FLG_IS_DISCARD) &&
1095                    ((arsp->rel_flags & (FLG_REL_GOT | FLG_REL_BSS |
1096                    FLG_REL_PLT | FLG_REL_NOINFO)) == 0)) {
1097                        DBG_CALL(Dbg_reloc_discard(ofl->ofl_lml, M_MACH, arsp));
1098                        continue;
1099                }
```

```
1101                /*
1102                 * Perform any required TLS fixups.
1103                 */
1104                if (arsp->rel_flags & FLG_REL_TLSFIX) {
1105                        Fixupret        ret;
1106
1107                        if ((ret = tls_fixups(ofl, arsp)) == FIX_ERROR)
1108                                return (S_ERROR);
1109                        if (ret == FIX_DONE)
1110                                continue;
1111                }
1112
1113                /*
1114                 * Perform any required GOTOP fixups.
1115                 */
1116                if (arsp->rel_flags & FLG_REL_GOTFIX) {
1117                        Fixupret        ret;
1118
1119                        if ((ret = gotop_fixups(ofl, arsp)) == FIX_ERROR)
1120                                return (S_ERROR);
1121                        if (ret == FIX_DONE)
1122                                continue;
1123                }
1124
1125                /*
1126                 * If this is a relocation against the move table, or
1127                 * expanded move table, adjust the relocation entries.
1128                 */
1129                if (RELAUX_GET_MOVE(arsp))
1130                        ld_adj_movereloc(ofl, arsp);
1131
1132                sdp = arsp->rel_sym;
1133                refaddr = arsp->rel_roffset +
1134                    (Off)_elf_getxoff(arsp->rel_isdesc->is_indata);
1135
1136                if ((arsp->rel_flags & FLG_REL_CLVAL) ||
1137                    (arsp->rel_flags & FLG_REL_GOTCL))
1138                        value = 0;
1139                else if (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION) {
1140                        Sym_desc        *sym;
1141
1142                        /*
1143                         * The value for a symbol pointing to a SECTION
1144                         * is based off of that sections position.
1145                         */
1146                        if ((sdp->sd_isc->is_flags & FLG_IS_RELUPD) &&
1147                            (sym = ld_am_I_partial(arsp, arsp->rel_raddend))) {
1148                                /*
1149                                 * The symbol was moved, so adjust the value
1150                                 * relative to the new section.
1151                                 */
1152                                value = _elf_getxoff(sym->sd_isc->is_indata);
1153                                if (sym->sd_isc->is_shdr->sh_flags & SHF_ALLOC)
1154                                        value += sym->sd_isc->
1155                                            is_osdesc->os_shdr->sh_addr;
1156
1157                                /*
1158                                 * The original raddend covers the displacement
1159                                 * from the section start to the desired
1160                                 * address. The value computed above gets us
1161                                 * from the section start to the start of the
1162                                 * symbol range. Adjust the old raddend to
1163                                 * remove the offset from section start to
1164                                 * symbol start, leaving the displacement
1165                                 * within the range of the symbol.
```



```
1166                                 */
1167                                arsp->rel_raddend -= sym->sd_osym->st_value;
1168                        } else {
1169                                value = _elf_getxoff(sdp->sd_isc->is_indata);
1170                                if (sdp->sd_isc->is_shdr->sh_flags & SHF_ALLOC)
1171                                        value += sdp->sd_isc->
1172                                            is_osdesc->os_shdr->sh_addr;
1173                        }
1174
1175                        if (sdp->sd_isc->is_shdr->sh_flags & SHF_TLS)
1176                                value -= ofl->ofl_tlsphdr->p_vaddr;
1177
1178                } else if (IS_SIZE(arsp->rel_rtype)) {
1179                        /*
1180                         * Size relocations require the symbols size.
1181                         */
1182                        value = sdp->sd_sym->st_size;
1183
1184                } else if ((sdp->sd_flags & FLG_SY_CAP) &&
1185                    sdp->sd_aux && sdp->sd_aux->sa_PLTndx) {
1186                        /*
1187                         * If relocation is against a capabilities symbol, we
1188                         * need to jump to an associated PLT, so that at runtime
1189                         * ld.so.1 is involved to determine the best binding
1190                         * choice. Otherwise, the value is the symbols value.
1191                         */
1192                        value = ld_calc_plt_addr(sdp, ofl);
1193
1194                } else
1195                        value = sdp->sd_sym->st_value;
1196
1197                /*
1198                 * Relocation against the GLOBAL_OFFSET_TABLE.
1199                 */
1200                if ((arsp->rel_flags & FLG_REL_GOT) &&
1201                    !ld_reloc_set_aux_osdesc(ofl, arsp, ofl->ofl_osgot))
1202                        return (S_ERROR);
1203                osp = RELAUX_GET_OSDESC(arsp);
1204
1205                /*
1206                 * If loadable and not producing a relocatable object add the
1207                 * sections virtual address to the reference address.
1208                 */
1209                if ((arsp->rel_flags & FLG_REL_LOAD) &&
1210                    ((flags & FLG_OF_RELOBJ) == 0))
1211                        refaddr +=
1212                            arsp->rel_isdesc->is_osdesc->os_shdr->sh_addr;
1213
1214                /*
1215                 * If this entry has a PLT assigned to it, its value is actually
1216                 * the address of the PLT (and not the address of the function).
1217                 */
1218                if (IS_PLT(arsp->rel_rtype)) {
1219                        if (sdp->sd_aux && sdp->sd_aux->sa_PLTndx)
1220                                value = ld_calc_plt_addr(sdp, ofl);
1221                }
1222
1223                /*
1224                 * Add relocations addend to value.  Add extra
1225                 * relocation addend if needed.
1226                 */
1227                value += arsp->rel_raddend;
1228                if (IS_EXTOFFSET(arsp->rel_rtype))
1229                        value += RELAUX_GET_TYPEDATA(arsp);
1230
1231                /*
```

```
1232                 * Determine whether the value needs further adjustment. Filter
1233                 * through the attributes of the relocation to determine what
1234                 * adjustment is required.  Note, many of the following cases
1235                 * are only applicable when a .got is present.  As a .got is
1236                 * not generated when a relocatable object is being built,
1237                 * any adjustments that require a .got need to be skipped.
1238                 */
1239                if ((arsp->rel_flags & FLG_REL_GOT) &&
1240                    ((flags & FLG_OF_RELOBJ) == 0)) {
1241                        Xword           R1addr;
1242                        uintptr_t       R2addr;
1243                        Sword           gotndx;
1244                        Gotndx          *gnp;
1245                        Gotref          gref;

1247                        /*
1248                         * Clear the GOT table entry, on SPARC we clear
1249                         * the entry and the 'value' if needed is stored
1250                         * in an output relocations addend.
1251                         *
1252                         * Calculate offset into GOT at which to apply
1253                         * the relocation.
1254                         */
1255                        if (arsp->rel_flags & FLG_REL_DTLS)
1256                                gref = GOT_REF_TLSGD;
1257                        else if (arsp->rel_flags & FLG_REL_MTLS)
1258                                gref = GOT_REF_TLSLD;
1259                        else if (arsp->rel_flags & FLG_REL_STLS)
1260                                gref = GOT_REF_TLSIE;
1261                        else
1262                                gref = GOT_REF_GENERIC;

1264                        gnp = ld_find_got_ndx(sdp->sd_GOTndxs, gref, ofl, arsp);
1265                        assert(gnp);

1267                        if (arsp->rel_rtype == M_R_DTPOFF)
1268                                gotndx = gnp->gn_gotndx + 1;
1269                        else
1270                                gotndx = gnp->gn_gotndx;

1272                        /* LINTED */
1273                        R1addr = (Xword)((-neggotoffset * M_GOT_ENTSIZE) +
1274                            (gotndx * M_GOT_ENTSIZE));

1276                        /*
1277                         * Add the GOTs data's offset.
1278                         */
1279                        R2addr = R1addr + (uintptr_t)osp->os_outdata->d_buf;

1281                        DBG_CALL(Dbg_reloc_doact(ofl->ofl_lml,
1282                            ELF_DBG_LD_ACT, M_MACH, SHT_RELA,
1283                            arsp, R1addr, value, ld_reloc_sym_name));

1285                        /*
1286                         * And do it.
1287                         */
1288                        if (ofl->ofl_flags1 & FLG_OF1_ENCDIFF)
1289                                *(Xword *)R2addr = ld_bswap_Xword(value);
1290                        else
1291                                *(Xword *)R2addr = value;
1292                        continue;

1294                } else if (IS_GOT_BASED(arsp->rel_rtype) &&
1295                    ((flags & FLG_OF_RELOBJ) == 0)) {
1296                        value -= (ofl->ofl_osgot->os_shdr->sh_addr +
1297                            (-neggotoffset * M_GOT_ENTSIZE));
```



```
1299                } else if (IS_PC_RELATIVE(arsp->rel_rtype)) {
1300                        value -= refaddr;

1302                } else if (IS_TLS_INS(arsp->rel_rtype) &&
1303                    IS_GOT_RELATIVE(arsp->rel_rtype) &&
1304                    ((flags & FLG_OF_RELOBJ) == 0)) {
1305                        Gotndx  *gnp;
1306                        Gotref  gref;

1308                        if (arsp->rel_flags & FLG_REL_STLS)
1309                                gref = GOT_REF_TLSIE;
1310                        else if (arsp->rel_flags & FLG_REL_DTLS)
1311                                gref = GOT_REF_TLSGD;
1312                        else if (arsp->rel_flags & FLG_REL_MTLS)
1313                                gref = GOT_REF_TLSLD;

1315                        gnp = ld_find_got_ndx(sdp->sd_GOTndxs, gref, ofl, arsp);
1316                        assert(gnp);

1318                        value = gnp->gn_gotndx * M_GOT_ENTSIZE;

1320                } else if (IS_GOT_RELATIVE(arsp->rel_rtype) &&
1321                    ((flags & FLG_OF_RELOBJ) == 0)) {
1322                        Gotndx  *gnp;

1324                        gnp = ld_find_got_ndx(sdp->sd_GOTndxs,
1325                            GOT_REF_GENERIC, ofl, arsp);
1326                        assert(gnp);

1328                        value = gnp->gn_gotndx * M_GOT_ENTSIZE;

1330                } else if ((arsp->rel_flags & FLG_REL_STLS) &&
1331                    ((flags & FLG_OF_RELOBJ) == 0)) {
1332                        Xword   tlsstatsize;

1334                        /*
1335                         * This is the LE TLS reference model. Static offset is
1336                         * hard-coded, and negated so that it can be added to
1337                         * the thread pointer (%g7)
1338                         */
1339                        tlsstatsize =
1340                            S_ROUND(ofl->ofl_tlsphdr->p_memsz, M_TLSSTATALIGN);
1341                        value = -(tlsstatsize - value);
1342                }

1344                if (arsp->rel_isdesc->is_file)
1345                        ifl_name = arsp->rel_isdesc->is_file->ifl_name;
1346                else
1347                        ifl_name = MSG_INTL(MSG_STR_NULL);

1349                /*
1350                 * Make sure we have data to relocate.  Compiler and assembler
1351                 * developers have been known to generate relocations against
1352                 * invalid sections (normally .bss), so for their benefit give
1353                 * them sufficient information to help analyze the problem.
1354                 * End users should never see this.
1355                 */
1356                if (arsp->rel_isdesc->is_indata->d_buf == 0) {
1357                        Conv_inv_buf_t  inv_buf;

1359                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_EMPTYSEC),
1360                            conv_reloc_SPARC_type(arsp->rel_rtype, 0, &inv_buf),
1361                            ifl_name, ld_reloc_sym_name(arsp),
1362                            EC_WORD(arsp->rel_isdesc->is_scnndx),
1363                            arsp->rel_isdesc->is_name);
```

```
1364                        return (S_ERROR);
1365                }
1366
1367                /*
1368                 * Get the address of the data item we need to modify.
1369                 */
1370                addr = (uchar_t *)((uintptr_t)arsp->rel_roffset +
1371                    (uintptr_t)_elf_getxoff(arsp->rel_isdesc->is_indata));
1372
1373                DBG_CALL(Dbg_reloc_doact(ofl->ofl_lml, ELF_DBG_LD_ACT,
1374                    M_MACH, SHT_RELA, arsp, EC_NATPTR(addr), value,
1375                    ld_reloc_sym_name));
1376                addr += (uintptr_t)osp->os_outdata->d_buf;
1377
1378                if ((((uintptr_t)addr - (uintptr_t)ofl->ofl_nehdr) >
1379                    ofl->ofl_size) || (arsp->rel_roffset >
1380                    osp->os_shdr->sh_size)) {
1381                        Conv_inv_buf_t  inv_buf;
1382                        int             class;
1383
1384                        if (((uintptr_t)addr - (uintptr_t)ofl->ofl_nehdr) >
1385                            ofl->ofl_size)
1386                                class = ERR_FATAL;
1387                        else
1388                                class = ERR_WARNING;
1389
1390                        ld_eprintf(ofl, class, MSG_INTL(MSG_REL_INVALOFFSET),
1391                            conv_reloc_SPARC_type(arsp->rel_rtype, 0, &inv_buf),
1392                            ifl_name, EC_WORD(arsp->rel_isdesc->is_scnndx),
1393                            arsp->rel_isdesc->is_name, ld_reloc_sym_name(arsp),
1394                            EC_ADDR((uintptr_t)addr -
1395                            (uintptr_t)ofl->ofl_nehdr));
1396
1397                        if (class == ERR_FATAL) {
1398                                return_code = S_ERROR;
1399                                continue;
1400                        }
1401                }
1402
1403                /*
1404                 * If '-z noreloc' is specified - skip the do_reloc stage.
1405                 */
1406                if (OFL_DO_RELOC(ofl)) {
1407                        if (do_reloc_ld(arsp, addr, &value, ld_reloc_sym_name,
1408                            ifl_name, OFL_SWAP_RELOC_DATA(ofl, arsp),
1409                            ofl->ofl_lml) == 0) {
1410                                ofl->ofl_flags |= FLG_OF_FATAL;
1411                                return_code = S_ERROR;
1412                        }
1413                }
1414        }
1415        return (return_code);
1416 }
1417
1418 static uintptr_t
1419 ld_add_outrel(Word flags, Rel_desc *rsp, Ofl_desc *ofl)
1420 {
1421        Rel_desc        *orsp;
1422        Sym_desc        *sdp = rsp->rel_sym;
1423        Conv_inv_buf_t  inv_buf;
1424
1425        /*
1426         * Static executables *do not* want any relocations against them.
1427         * Since our engine still creates relocations against a WEAK UNDEFINED
1428         * symbol in a static executable, it's best to disable them here
1429         * instead of through out the relocation code.
```



```
1430         */
1431        if (OFL_IS_STATIC_EXEC(ofl))
1432                return (1);
1433
1434        /*
1435         * If the symbol will be reduced, we can't leave outstanding
1436         * relocations against it, as nothing will ever be able to satisfy them
1437         * (and the symbol won't be in .dynsym
1438         */
1439        if ((sdp != NULL) &&
1440            (sdp->sd_sym->st_shndx == SHN_UNDEF) &&
1441            (rsp->rel_rtype != M_R_NONE) &&
1442            (rsp->rel_rtype != M_R_REGISTER) &&
1443            (rsp->rel_rtype != M_R_RELATIVE)) {
1444                if (ld_sym_reducable(ofl, sdp))
1445                        return (1);
1446        }
1447 #endif /* ! codereview */
1448
1449        /*
1450         * Certain relocations do not make sense in a 64bit shared object,
1451         * if building a shared object do a sanity check on the output
1452         * relocations being created.
1453         */
1454        if (ofl->ofl_flags & FLG_OF_SHAROBJ) {
1455                Word    rtype = rsp->rel_rtype;
1456                /*
1457                 * Because the R_SPARC_HIPLT22 & R_SPARC_LOPLT10 relocations
1458                 * are not relative they make no sense to create in a shared
1459                 * object - so emit the proper error message if that occurs.
1460                 */
1461                if ((rtype == R_SPARC_HIPLT22) || (rtype == R_SPARC_LOPLT10)) {
1462                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_UNRELREL),
1463                            conv_reloc_SPARC_type(rsp->rel_rtype, 0, &inv_buf),
1464                            rsp->rel_isdesc->is_file->ifl_name,
1465                            ld_reloc_sym_name(rsp));
1466                        return (S_ERROR);
1467                }
1468 #if     defined(_ELF64)
1469                /*
1470                 * Each of the following relocations requires that the
1471                 * object being built be loaded in either the upper 32 or
1472                 * 44 bit range of memory.  Since shared libraries traditionally
1473                 * are loaded in the lower range of memory - this isn't going
1474                 * to work.
1475                 */
1476                if ((rtype == R_SPARC_H44) || (rtype == R_SPARC_M44) ||
1477                    (rtype == R_SPARC_L44)) {
1478                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_SHOBJABS44),
1479                            conv_reloc_SPARC_type(rsp->rel_rtype, 0, &inv_buf),
1480                            rsp->rel_isdesc->is_file->ifl_name,
1481                            ld_reloc_sym_name(rsp));
1482                        return (S_ERROR);
1483                }
1484 #endif
1485        }
1486
1487        /*
1488         * If we are adding a output relocation against a section
1489         * symbol (non-RELATIVE) then mark that section.  These sections
1490         * will be added to the .dynsym symbol table.
1491         */
1492        if (sdp && (rsp->rel_rtype != M_R_RELATIVE) &&
1493            ((flags & FLG_REL_SCNNDX) ||
1494            (ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION))) {
```

```
1496                 /*
1497                  * If this is a COMMON symbol - no output section
1498                  * exists yet - (it's created as part of sym_validate()).
1499                  * So - we mark here that when it's created it should
1500                  * be tagged with the FLG_OS_OUTREL flag.
1501                  */
1502                 if ((sdp->sd_flags & FLG_SY_SPECSEC) &&
1503                     (sdp->sd_sym->st_shndx == SHN_COMMON)) {
1504                         if (ELF_ST_TYPE(sdp->sd_sym->st_info) != STT_TLS)
1505                                 ofl->ofl_flags1 |= FLG_OF1_BSSOREL;
1506                         else
1507                                 ofl->ofl_flags1 |= FLG_OF1_TLSOREL;
1508                 } else {
1509                         Os_desc *osp;
1510                         Is_desc *isp = sdp->sd_isc;
1511
1512                         if (isp && ((osp = isp->is_osdesc) != NULL) &&
1513                             ((osp->os_flags & FLG_OS_OUTREL) == 0)) {
1514                                 ofl->ofl_dynshdrcnt++;
1515                                 osp->os_flags |= FLG_OS_OUTREL;
1516                         }
1517                 }
1518         }
1519
1520         /* Enter it into the output relocation cache */
1521         if ((orsp = ld_reloc_enter(ofl, &ofl->ofl_outrels, rsp, flags)) == NULL)
1522                 return (S_ERROR);
1523
1524         if (flags & FLG_REL_GOT)
1525                 ofl->ofl_relocgotsz += (Xword)sizeof (Rela);
1526         else if (flags & FLG_REL_PLT)
1527                 ofl->ofl_relocpltsz += (Xword)sizeof (Rela);
1528         else if (flags & FLG_REL_BSS)
1529                 ofl->ofl_relocbsssz += (Xword)sizeof (Rela);
1530         else if (flags & FLG_REL_NOINFO)
1531                 ofl->ofl_relocrelsz += (Xword)sizeof (Rela);
1532         else
1533                 RELAUX_GET_OSDESC(orsp)->os_szoutrels += (Xword)sizeof (Rela);
1534
1535         if (orsp->rel_rtype == M_R_RELATIVE)
1536                 ofl->ofl_relocrelcnt++;
1537
1538 #if     defined(_ELF64)
1539         /*
1540          * When building a 64-bit object any R_SPARC_WDISP30 relocation is given
1541          * a plt padding entry, unless we're building a relocatable object
1542          * (ld -r) or -b is in effect.
1543          */
1544         if ((orsp->rel_rtype == R_SPARC_WDISP30) &&
1545             ((ofl->ofl_flags & (FLG_OF_BFLAG | FLG_OF_RELOBJ)) == 0) &&
1546             ((orsp->rel_sym->sd_flags & FLG_SY_PLTPAD) == 0)) {
1547                 ofl->ofl_pltpad++;
1548                 orsp->rel_sym->sd_flags |= FLG_SY_PLTPAD;
1549         }
1550 #endif
1551         /*
1552          * We don't perform sorting on PLT relocations because
1553          * they have already been assigned a PLT index and if we
1554          * were to sort them we would have to re-assign the plt indexes.
1555          */
1556         if (!(flags & FLG_REL_PLT))
1557                 ofl->ofl_reloccnt++;
1558
1559         /*
1560          * Insure a GLOBAL_OFFSET_TABLE is generated if required.
1561          */
```



```
1562         if (IS_GOT_REQUIRED(orsp->rel_rtype))
1563                 ofl->ofl_flags |= FLG_OF_BLDGOT;
1564
1565         /*
1566          * Identify and possibly warn of a displacement relocation.
1567          */
1568         if (orsp->rel_flags & FLG_REL_DISP) {
1569                 ofl->ofl_dtflags_1 |= DF_1_DISPRELPND;
1570
1571                 if (ofl->ofl_flags & FLG_OF_VERBOSE)
1572                         ld_disp_errmsg(MSG_INTL(MSG_REL_DISPREL4), orsp, ofl);
1573         }
1574         DBG_CALL(Dbg_reloc_ors_entry(ofl->ofl_lml, ELF_DBG_LD, SHT_RELA,
1575             M_MACH, orsp));
1576         return (1);
1577 }
1578
1579 /*
1580  * Process relocation against a register symbol.  Note, of -z muldefs is in
1581  * effect there may have been multiple register definitions, which would have
1582  * been processed as non-fatal, with the first definition winning.  But, we
1583  * will also process multiple relocations for these multiple definitions.  In
1584  * this case we must only preserve the relocation for the definition that was
1585  * kept.  The sad part is that register relocations don't typically specify
1586  * the register symbol with which they are associated, so we might have to
1587  * search the input files global symbols to determine if this relocation is
1588  * appropriate.
1589  */
1590 static uintptr_t
1591 ld_reloc_register(Rel_desc *rsp, Is_desc *isp, Ofl_desc *ofl)
1592 {
1593         if (ofl->ofl_flags & FLG_OF_MULDEFS) {
1594                 Ifl_desc        *ifl = isp->is_file;
1595                 Sym_desc        *sdp = rsp->rel_sym;
1596
1597                 if (sdp == 0) {
1598                         Xword           offset = rsp->rel_roffset;
1599                         Word            ndx;
1600
1601                         for (ndx = ifl->ifl_locscnt;
1602                             ndx < ifl->ifl_symscnt; ndx++) {
1603                                 if (((sdp = ifl->ifl_oldndx[ndx]) != 0) &&
1604                                     (sdp->sd_flags & FLG_SY_REGSYM) &&
1605                                     (sdp->sd_sym->st_value == offset))
1606                                         break;
1607                         }
1608                 }
1609                 if (sdp && (sdp->sd_file != ifl))
1610                         return (1);
1611         }
1612         return (ld_add_outrel((rsp->rel_flags | FLG_REL_REG), rsp, ofl));
1613 }
1614
1615 /*
1616  * process relocation for a LOCAL symbol
1617  */
1618 static uintptr_t
1619 ld_reloc_local(Rel_desc *rsp, Ofl_desc *ofl)
1620 {
1621         ofl_flag_t      flags = ofl->ofl_flags;
1622         Sym_desc        *sdp = rsp->rel_sym;
1623         Word            shndx = sdp->sd_sym->st_shndx;
1624
1625         /*
1626          * if ((shared object) and (not pc relative relocation) and
1627          *    (not against ABS symbol))
```

```
1628          * then
1629          *      if (rtype != R_SPARC_32)
1630          *      then
1631          *              build relocation against section
1632          *      else
1633          *              build R_SPARC_RELATIVE
1634          *      fi
1635          * fi
1636          */
1637         if ((flags & FLG_OF_SHAROBJ) && (rsp->rel_flags & FLG_REL_LOAD) &&
1638             !(IS_PC_RELATIVE(rsp->rel_rtype)) && !(IS_SIZE(rsp->rel_rtype)) &&
1639             !(IS_GOT_BASED(rsp->rel_rtype)) &&
1640             !(rsp->rel_isdesc != NULL &&
1641             (rsp->rel_isdesc->is_shdr->sh_type == SHT_SUNW_dof)) &&
1642             (((sdp->sd_flags & FLG_SY_SPECSEC) == 0) ||
1643             (shndx != SHN_ABS) || (sdp->sd_aux && sdp->sd_aux->sa_symspec))) {
1644                 Word    ortype = rsp->rel_rtype;
1645
1646                 if ((rsp->rel_rtype != R_SPARC_32) &&
1647                     (rsp->rel_rtype != R_SPARC_PLT32) &&
1648                     (rsp->rel_rtype != R_SPARC_64))
1649                         return (ld_add_outrel((FLG_REL_SCNNDX | FLG_REL_ADVAL),
1650                             rsp, ofl));
1651
1652                 rsp->rel_rtype = R_SPARC_RELATIVE;
1653                 if (ld_add_outrel(FLG_REL_ADVAL, rsp, ofl) == S_ERROR)
1654                         return (S_ERROR);
1655                 rsp->rel_rtype = ortype;
1656                 return (1);
1657         }
1658
1659         /*
1660          * If the relocation is against a 'non-allocatable' section
1661          * and we can not resolve it now - then give a warning
1662          * message.
1663          *
1664          * We can not resolve the symbol if either:
1665          *      a) it's undefined
1666          *      b) it's defined in a shared library and a
1667          *         COPY relocation hasn't moved it to the executable
1668          *
1669          * Note: because we process all of the relocations against the
1670          *      text segment before any others - we know whether
1671          *      or not a copy relocation will be generated before
1672          *      we get here (see reloc_init()->reloc_segments()).
1673          */
1674         if (!(rsp->rel_flags & FLG_REL_LOAD) &&
1675             ((shndx == SHN_UNDEF) ||
1676             ((sdp->sd_ref == REF_DYN_NEED) &&
1677             ((sdp->sd_flags & FLG_SY_MVTOCOMM) == 0)))) {
1678                 Conv_inv_buf_t  inv_buf;
1679                 Os_desc         *osp = RELAUX_GET_OSDESC(rsp);
1680
1681                 /*
1682                  * If the relocation is against a SHT_SUNW_ANNOTATE
1683                  * section - then silently ignore that the relocation
1684                  * can not be resolved.
1685                  */
1686                 if (osp && (osp->os_shdr->sh_type == SHT_SUNW_ANNOTATE))
1687                         return (0);
1688                 ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_REL_EXTERNSYM),
1689                     conv_reloc_SPARC_type(rsp->rel_rtype, 0, &inv_buf),
1690                     rsp->rel_isdesc->is_file->ifl_name,
1691                     ld_reloc_sym_name(rsp), osp->os_name);
1692                 return (1);
1693         }
```



```
1695         /*
1696          * Perform relocation.
1697          */
1698         return (ld_add_actrel(NULL, rsp, ofl));
1699 }
1700
1701 /*
1702  * Establish a relocation transition.  Note, at this point of input relocation
1703  * processing, we have no idea of the relocation value that will be used in
1704  * the eventual relocation calculation.  This value is only known after the
1705  * initial image has been constructed.  Therefore, there is a small chance
1706  * that a value can exceed the capabilities of the transitioned relocation.
1707  * One example might be the offset from the GOT to a symbol.
1708  *
1709  * The only instance of this failure discovered so far has been via the use of
1710  * ABS symbols to represent an external memory location.  This situation is
1711  * rare, since ABS symbols aren't typically generated by the compilers.
1712  * Therefore, our solution is to excluded ABS symbols from the transition
1713  * relocation possibilities.  As an additional safeguard, if an inappropriate
1714  * value is passed to the final relocation engine, a verification ("V")
1715  * relocation should trigger a fatal error condition.
1716  */
1717 static uintptr_t
1718 ld_reloc_GOTOP(Boolean local, Rel_desc *rsp, Ofl_desc *ofl)
1719 {
1720         Word    rtype = rsp->rel_rtype;
1721
1722         if (!local || (rsp->rel_sym->sd_sym->st_shndx == SHN_ABS)) {
1723                 /*
1724                  * When binding to a external symbol, no fixups are required
1725                  * and the GOTDATA_OP relocation can be ignored.
1726                  */
1727                 if (rtype == R_SPARC_GOTDATA_OP)
1728                         return (1);
1729                 return (ld_reloc_GOT_relative(local, rsp, ofl));
1730         }
1731
1732         /*
1733          * When binding to a local symbol the relocations can be transitioned:
1734          *
1735          *      R_*_GOTDATA_OP_HIX22 -> R_*_GOTDATA_HIX22
1736          *      R_*_GOTDATA_OP_LOX10 -> R_*_GOTDATA_LOX10
1737          *      R_*_GOTDATA_OP ->       instruction fixup
1738          */
1739         return (ld_add_actrel(FLG_REL_GOTFIX, rsp, ofl));
1740 }
1741
1742 static uintptr_t
1743 ld_reloc_TLS(Boolean local, Rel_desc *rsp, Ofl_desc *ofl)
1744 {
1745         Word            rtype = rsp->rel_rtype;
1746         Sym_desc        *sdp = rsp->rel_sym;
1747         ofl_flag_t      flags = ofl->ofl_flags;
1748         Gotndx          *gnp;
1749
1750         /*
1751          * If we're building an executable - use either the IE or LE access
1752          * model.  If we're building a shared object process any IE model.
1753          */
1754         if ((flags & FLG_OF_EXEC) || (IS_TLS_IE(rtype))) {
1755                 /*
1756                  * Set the DF_STATIC_TLS flag.
1757                  */
1758                 ofl->ofl_dtflags |= DF_STATIC_TLS;
```

```
1760                if (!local || ((flags & FLG_OF_EXEC) == 0)) {
1761                        /*
1762                         * When processing static TLS - these relocations
1763                         * can be ignored.
1764                         */
1765                        if ((rtype == R_SPARC_TLS_IE_LD) ||
1766                            (rtype == R_SPARC_TLS_IE_LDX) ||
1767                            (rtype == R_SPARC_TLS_IE_ADD))
1768                                return (1);

1770                        /*
1771                         * Assign a GOT entry for IE static TLS references.
1772                         */
1773                        if (((rtype == R_SPARC_TLS_GD_HI22) ||
1774                            (rtype == R_SPARC_TLS_GD_LO10) ||
1775                            (rtype == R_SPARC_TLS_IE_HI22) ||
1776                            (rtype == R_SPARC_TLS_IE_LO10)) &&
1777                            ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs,
1778                            GOT_REF_TLSIE, ofl, rsp)) == NULL)) {

1780                                if (ld_assign_got_TLS(local, rsp, ofl, sdp,
1781                                    gnp, GOT_REF_TLSIE, FLG_REL_STLS,
1782                                    rtype, M_R_TPOFF, NULL) == S_ERROR)
1783                                        return (S_ERROR);
1784                        }

1786                        /*
1787                         * IE access model.
1788                         */
1789                        if (IS_TLS_IE(rtype))
1790                                return (ld_add_actrel(FLG_REL_STLS, rsp, ofl));

1792                        /*
1793                         * Fixups are required for other executable models.
1794                         */
1795                        return (ld_add_actrel((FLG_REL_TLSFIX | FLG_REL_STLS),
1796                            rsp, ofl));
1797                }

1799                /*
1800                 * LE access model.
1801                 */
1802                if (IS_TLS_LE(rtype))
1803                        return (ld_add_actrel(FLG_REL_STLS, rsp, ofl));

1805                /*
1806                 * When processing static TLS - these relocations can be
1807                 * ignored.
1808                 */
1809                if (rtype == R_SPARC_TLS_IE_ADD)
1810                        return (1);

1812                return (ld_add_actrel((FLG_REL_TLSFIX | FLG_REL_STLS),
1813                    rsp, ofl));
1814        }

1816        /*
1817         * Building a shared object.
1818         *
1819         * For dynamic TLS references, ADD relocations are ignored.
1820         */
1821        if ((rtype == R_SPARC_TLS_GD_ADD) || (rtype == R_SPARC_TLS_LDM_ADD) ||
1822            (rtype == R_SPARC_TLS_LDO_ADD))
1823                return (1);

1825        /*
```



```
1826         * Assign a GOT entry for a dynamic TLS reference.
1827         */
1828        if (((rtype == R_SPARC_TLS_LDM_HI22) ||
1829            (rtype == R_SPARC_TLS_LDM_LO10)) &&
1830            ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs, GOT_REF_TLSLD,
1831            ofl, rsp)) == NULL)) {

1833                if (ld_assign_got_TLS(local, rsp, ofl, sdp, gnp, GOT_REF_TLSLD,
1834                    FLG_REL_MTLS, rtype, M_R_DTPMOD, 0) == S_ERROR)
1835                        return (S_ERROR);

1837        } else if (((rtype == R_SPARC_TLS_GD_HI22) ||
1838            (rtype == R_SPARC_TLS_GD_LO10)) &&
1839            ((gnp = ld_find_got_ndx(sdp->sd_GOTndxs, GOT_REF_TLSGD,
1840            ofl, rsp)) == NULL)) {

1842                if (ld_assign_got_TLS(local, rsp, ofl, sdp, gnp, GOT_REF_TLSGD,
1843                    FLG_REL_DTLS, rtype, M_R_DTPMOD, M_R_DTPOFF) == S_ERROR)
1844                        return (S_ERROR);
1845        }

1847        /*
1848         * For GD/LD TLS reference - TLS_{GD,LD}_CALL, this will eventually
1849         * cause a call to __tls_get_addr().  Convert this relocation to that
1850         * symbol now, and prepare for the PLT magic.
1851         */
1852        if ((rtype == R_SPARC_TLS_GD_CALL) || (rtype == R_SPARC_TLS_LDM_CALL)) {
1853                Sym_desc        *tlsgetsym;

1855                if ((tlsgetsym = ld_sym_add_u(MSG_ORIG(MSG_SYM_TLSGETADDR_U),
1856                    ofl, MSG_STR_TLSREL)) == (Sym_desc *)S_ERROR)
1857                        return (S_ERROR);

1859                rsp->rel_sym = tlsgetsym;
1860                rsp->rel_rtype = R_SPARC_WPLT30;

1862                if (ld_reloc_plt(rsp, ofl) == S_ERROR)
1863                        return (S_ERROR);

1865                rsp->rel_sym = sdp;
1866                rsp->rel_rtype = rtype;
1867                return (1);
1868        }

1870        if (IS_TLS_LD(rtype))
1871                return (ld_add_actrel(FLG_REL_MTLS, rsp, ofl));

1873        return (ld_add_actrel(FLG_REL_DTLS, rsp, ofl));
1874 }

1876 /*
1877  * ld_allocate_got: if a GOT is to be made, after the section is built this
1878  * function is called to allocate all the GOT slots.  The allocation is
1879  * deferred until after all GOTs have been counted and sorted according
1880  * to their size, for only then will we know how to allocate them on
1881  * a processor like SPARC which has different models for addressing the
1882  * GOT.  SPARC has two: small and large, small uses a signed 13-bit offset
1883  * into the GOT, whereas large uses an unsigned 32-bit offset.
1884  */
1885 static  Sword small_index;      /* starting index for small GOT entries */
1886 static  Sword mixed_index;      /* starting index for mixed GOT entries */
1887 static  Sword large_index;      /* starting index for large GOT entries */

1889 static uintptr_t
1890 ld_assign_got(Ofl_desc *ofl, Sym_desc *sdp)
1891 {
```

```
1892         Aliste idx;
1893         Gotndx *gnp;

1895         for (ALIST_TRAVERSE(sdp->sd_GOTndxs, idx, gnp)) {
1896                 uint_t  gotents;
1897                 Gotref  gref = gnp->gn_gotref;

1899                 if ((gref == GOT_REF_TLSGD) || (gref == GOT_REF_TLSLD))
1900                         gotents = 2;
1901                 else
1902                         gotents = 1;

1904                 switch (gnp->gn_gotndx) {
1905                 case M_GOT_SMALL:
1906                         gnp->gn_gotndx = small_index;
1907                         small_index += gotents;
1908                         if (small_index == 0)
1909                                 small_index = M_GOT_XNumber;
1910                         break;
1911                 case M_GOT_MIXED:
1912                         gnp->gn_gotndx = mixed_index;
1913                         mixed_index += gotents;
1914                         break;
1915                 case M_GOT_LARGE:
1916                         gnp->gn_gotndx = large_index;
1917                         large_index += gotents;
1918                         break;
1919                 default:
1920                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_ASSIGNGOT),
1921                             EC_XWORD(gnp->gn_gotndx), demangle(sdp->sd_name));
1922                         return (S_ERROR);
1923                 }
1924         }
1925         return (1);
1926 }

1928 static uintptr_t
1929 ld_assign_got_ndx(Alist **alpp, Gotndx *pgnp, Gotref gref, Ofl_desc *ofl,
1930     Rel_desc *rsp, Sym_desc *sdp)
1931 {
1932         Xword           raddend;
1933         Gotndx          gn, *gnp;
1934         Aliste          idx;
1935         uint_t          gotents;

1937         /* Some TLS requires two relocations with two GOT entries */
1938         if ((gref == GOT_REF_TLSGD) || (gref == GOT_REF_TLSLD))
1939                 gotents = 2;
1940         else
1941                 gotents = 1;

1943         raddend = rsp->rel_raddend;
1944         if (pgnp && (pgnp->gn_addend == raddend) && (pgnp->gn_gotref == gref)) {

1946                 /*
1947                  * If an entry for this addend already exists, determine if it
1948                  * has mixed mode GOT access (both PIC and pic).
1949                  *
1950                  * In order to be accessible by both large and small pic,
1951                  * a mixed mode GOT must be located in the positive index
1952                  * range above _GLOBAL_OFFSET_TABLE_, and in the range
1953                  * reachable small pic. This is necessary because the large
1954                  * PIC mode cannot use a negative offset. This implies that
1955                  * there can be no more than (M_GOT_MAXSMALL/2 - M_GOT_XNumber)
1956                  * such entries.
1957                  */
```



```
1958                 switch (pgnp->gn_gotndx) {
1959                 case M_GOT_SMALL:
1960                         /*
1961                          * This one was previously identified as a small
1962                          * GOT. If this access is large, then convert
1963                          * it to mixed.
1964                          */
1965                         if (rsp->rel_rtype != R_SPARC_GOT13) {
1966                                 pgnp->gn_gotndx = M_GOT_MIXED;
1967                                 mixgotcnt += gotents;
1968                         }
1969                         break;

1971                 case M_GOT_LARGE:
1972                         /*
1973                          * This one was previously identified as a large
1974                          * GOT. If this access is small, convert it to mixed.
1975                          */
1976                         if (rsp->rel_rtype == R_SPARC_GOT13) {
1977                                 smlgotcnt += gotents;
1978                                 mixgotcnt += gotents;
1979                                 pgnp->gn_gotndx = M_GOT_MIXED;
1980                                 sdp->sd_flags |= FLG_SY_SMGOT;
1981                         }
1982                         break;
1983                 }
1984                 return (1);
1985         }

1987         gn.gn_addend = raddend;
1988         gn.gn_gotref = gref;

1990         if (rsp->rel_rtype == R_SPARC_GOT13) {
1991                 gn.gn_gotndx = M_GOT_SMALL;
1992                 smlgotcnt += gotents;
1993                 sdp->sd_flags |= FLG_SY_SMGOT;
1994         } else
1995                 gn.gn_gotndx = M_GOT_LARGE;

1997         ofl->ofl_gotcnt += gotents;

1999         if (gref == GOT_REF_TLSLD) {
2000                 if (ofl->ofl_tlsldgotndx == NULL) {
2001                         if ((gnp = libld_malloc(sizeof (Gotndx))) == NULL)
2002                                 return (S_ERROR);
2003                         (void) memcpy(gnp, &gn, sizeof (Gotndx));
2004                         ofl->ofl_tlsldgotndx = gnp;
2005                 }
2006                 return (1);
2007         }

2009         idx = 0;
2010         for (ALIST_TRAVERSE(*alpp, idx, gnp)) {
2011                 if (gnp->gn_addend > raddend)
2012                         break;
2013         }

2015         /*
2016          * GOT indexes are maintained on an Alist, where there is typically
2017          * only one index.  The usage of this list is to scan the list to find
2018          * an index, and then apply that index immediately to a relocation.
2019          * Thus there are no external references to these GOT index structures
2020          * that can be compromised by the Alist being reallocated.
2021          */
2022         if (alist_insert(alpp, &gn, sizeof (Gotndx),
2023             AL_CNT_SDP_GOT, idx) == NULL)
```

```
2024                 return (S_ERROR);

2026         return (1);
2027 }

2029 static void
2030 ld_assign_plt_ndx(Sym_desc * sdp, Ofl_desc *ofl)
2031 {
2032         sdp->sd_aux->sa_PLTndx = 1 + ofl->ofl_pltcnt++;
2033 }


2036 static uintptr_t
2037 ld_allocate_got(Ofl_desc * ofl)
2038 {
2039         const Sword     first_large_ndx = M_GOT_MAXSMALL / 2;
2040         Sym_desc        *sdp;
2041         Addr            addr;

2043         /*
2044          * Sanity check -- is this going to fit at all? There are two
2045          * limits to be concerned about:
2046          *      1) There is a limit on the number of small pic GOT indices,
2047          *              given by M_GOT_MAXSMALL.
2048          *      2) If there are more than (M_GOT_MAXSMALL/2 - M_GOT_XNumber)
2049          *              small GOT indices, there will be items at negative
2050          *              offsets from _GLOBAL_OFFSET_TABLE_. Items that are
2051          *              accessed via large (PIC) code cannot reach these
2052          *              negative slots, so mixed mode items must be in the
2053          *              non-negative range. This implies a limit of
2054          *              (M_GOT_MAXSMALL/2 - M_GOT_XNumber) mixed mode indices.
2055          */
2056         if (smlgotcnt > M_GOT_MAXSMALL) {
2057                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_SMALLGOT),
2058                     EC_WORD(smlgotcnt), M_GOT_MAXSMALL);
2059                 return (S_ERROR);
2060         }
2061         if (mixgotcnt > (first_large_ndx - M_GOT_XNumber)) {
2062                 ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_REL_MIXEDGOT),
2063                     EC_WORD(mixgotcnt), first_large_ndx - M_GOT_XNumber);
2064                 return (S_ERROR);
2065         }

2067         /*
2068          * Set starting offset to be either 0, or a negative index into
2069          * the GOT based on the number of small symbols we've got.
2070          */
2071         neggotoffset = ((smlgotcnt >= first_large_ndx) ?
2072             (first_large_ndx - smlgotcnt) : 0);

2074         /*
2075          * Initialize the got offsets used by assign_got() to
2076          * locate GOT items:
2077          *      small - Starting index of items referenced only
2078          *              by small offsets (-Kpic).
2079          *      mixed - Starting index of items referenced
2080          *              by both large (-KPIC) and small (-Kpic).
2081          *      large - Indexes referenced only by large (-KPIC)
2082          *
2083          *  Small items can have negative indexes (i.e. lie below
2084          *      _GLOBAL_OFFSET_TABLE_). Mixed and large items must have
2085          *      non-negative offsets.
2086          */
2087         small_index = (neggotoffset == 0) ? M_GOT_XNumber : neggotoffset;
2088         large_index = neggotoffset + smlgotcnt;
2089         mixed_index = large_index - mixgotcnt;
```



```
2091         /*
2092          * Assign bias to GOT symbols.
2093          */
2094         addr = -neggotoffset * M_GOT_ENTSIZE;
2095         if ((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_GOFTBL), SYM_NOHASH,
2096             NULL, ofl)) != NULL)
2097                 sdp->sd_sym->st_value = addr;
2098         if ((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_GOFTBL_U), SYM_NOHASH,
2099             NULL, ofl)) != NULL)
2100                 sdp->sd_sym->st_value = addr;

2102         if (ofl->ofl_tlsldgotndx) {
2103                 ofl->ofl_tlsldgotndx->gn_gotndx = large_index;
2104                 large_index += 2;
2105         }
2106         return (1);
2107 }

2109 /*
2110  * Initializes .got[0] with the _DYNAMIC symbol value.
2111  */
2112 static uintptr_t
2113 ld_fillin_gotplt(Ofl_desc *ofl)
2114 {
2115         if (ofl->ofl_osgot) {
2116                 Sym_desc        *sdp;

2118                 if ((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_DYNAMIC_U),
2119                     SYM_NOHASH, NULL, ofl)) != NULL) {
2120                         uchar_t *genptr;

2122                         genptr = ((uchar_t *)ofl->ofl_osgot->os_outdata->d_buf +
2123                             (-neggotoffset * M_GOT_ENTSIZE) +
2124                             (M_GOT_XDYNAMIC * M_GOT_ENTSIZE));
2125                         /* LINTED */
2126                         *((Xword *)genptr) = sdp->sd_sym->st_value;
2127                         if (ofl->ofl_flags1 & FLG_OF1_ENCDIFF)
2128                                 /* LINTED */
2129                                 *((Xword *)genptr) =
2130                                     /* LINTED */
2131                                     ld_bswap_Xword(*((Xword *)genptr));
2132                 }
2133         }
2134         return (1);
2135 }



2139 /*
2140  * Template for generating "void (*)(void)" function
2141  */
2142 static const uchar_t nullfunc_tmpl[] = {
2143 /* 0x00 */      0x81, 0xc3, 0xe0, 0x08,         /* retl */
2144 /* 0x04 */      0x01, 0x00, 0x00, 0x00          /* nop */
2145 };



2149 /*
2150  * Return the ld_targ definition for this target.
2151  */
2152 const Target *
2153 ld_targ_init_sparc(void)
2154 {
2155         static const Target _ld_targ = {
```

```
2156                {                           /* Target_mach */
2157                        M_MACH,                 /* m_mach */
2158                        M_MACHPLUS,             /* m_machplus */
2159                        M_FLAGSPLUS,            /* m_flagsplus */
2160                        M_CLASS,                /* m_class */
2161                        M_DATA,                 /* m_data */

2163                        M_SEGM_ALIGN,           /* m_segm_align */
2164                        M_SEGM_ORIGIN,          /* m_segm_origin */
2165                        M_SEGM_AORIGIN,         /* m_segm_aorigin */
2166                        M_DATASEG_PERM,         /* m_dataseg_perm */
2167                        M_STACK_PERM,           /* m_stack_perm */
2168                        M_WORD_ALIGN,           /* m_word_align */
2169                                                /* m_def_interp */
2170 #if     defined(_ELF64)
2171                        MSG_ORIG(MSG_PTH_RTLD_SPARCV9),
2172 #else
2173                        MSG_ORIG(MSG_PTH_RTLD),
2174 #endif

2176                        /* Relocation type codes */
2177                        M_R_ARRAYADDR,          /* m_r_arrayaddr */
2178                        M_R_COPY,               /* m_r_copy */
2179                        M_R_GLOB_DAT,           /* m_r_glob_dat */
2180                        M_R_JMP_SLOT,           /* m_r_jmp_slot */
2181                        M_R_NUM,                /* m_r_num */
2182                        M_R_NONE,               /* m_r_none */
2183                        M_R_RELATIVE,           /* m_r_relative */
2184                        M_R_REGISTER,           /* m_r_register */

2186                        /* Relocation related constants */
2187                        M_REL_DT_COUNT,         /* m_rel_dt_count */
2188                        M_REL_DT_ENT,           /* m_rel_dt_ent */
2189                        M_REL_DT_SIZE,          /* m_rel_dt_size */
2190                        M_REL_DT_TYPE,          /* m_rel_dt_type */
2191                        M_REL_SHT_TYPE,         /* m_rel_sht_type */

2193                        /* GOT related constants */
2194                        M_GOT_ENTSIZE,          /* m_got_entsize */
2195                        M_GOT_XNumber,          /* m_got_xnumber */

2197                        /* PLT related constants */
2198                        M_PLT_ALIGN,            /* m_plt_align */
2199                        M_PLT_ENTSIZE,          /* m_plt_entsize */
2200                        M_PLT_RESERVSZ,         /* m_plt_reservsz */
2201                        M_PLT_SHF_FLAGS,        /* m_plt_shf_flags */

2203                        /* Section type of .eh_frame/.eh_frame_hdr sections */
2204                        SHT_PROGBITS,           /* m_sht_unwind */

2206                        M_DT_REGISTER,          /* m_dt_register */
2207                },
2208                {                           /* Target_machid */
2209                        M_ID_ARRAY,             /* id_array */
2210                        M_ID_BSS,               /* id_bss */
2211                        M_ID_CAP,               /* id_cap */
2212                        M_ID_CAPINFO,           /* id_capinfo */
2213                        M_ID_CAPCHAIN,          /* id_capchain */
2214                        M_ID_DATA,              /* id_data */
2215                        M_ID_DYNAMIC,           /* id_dynamic */
2216                        M_ID_DYNSORT,           /* id_dynsort */
2217                        M_ID_DYNSTR,            /* id_dynstr */
2218                        M_ID_DYNSYM,            /* id_dynsym */
2219                        M_ID_DYNSYM_NDX,        /* id_dynsym_ndx */
2220                        M_ID_GOT,               /* id_got */
2221                        M_ID_GOTDATA,           /* id_gotdata */
```



```
2222                        M_ID_HASH,              /* id_hash */
2223                        M_ID_INTERP,            /* id_interp */
2224                        M_ID_UNKNOWN,           /* id_lbss (unused) */
2225                        M_ID_LDYNSYM,           /* id_ldynsym */
2226                        M_ID_NOTE,              /* id_note */
2227                        M_ID_NULL,              /* id_null */
2228                        M_ID_PLT,               /* id_plt */
2229                        M_ID_REL,               /* id_rel */
2230                        M_ID_STRTAB,            /* id_strtab */
2231                        M_ID_SYMINFO,           /* id_syminfo */
2232                        M_ID_SYMTAB,            /* id_symtab */
2233                        M_ID_SYMTAB_NDX,        /* id_symtab_ndx */
2234                        M_ID_TEXT,              /* id_text */
2235                        M_ID_TLS,               /* id_tls */
2236                        M_ID_TLSBSS,            /* id_tlsbss */
2237                        M_ID_UNKNOWN,           /* id_unknown */
2238                        M_ID_UNWIND,            /* id_unwind */
2239                        M_ID_UNWINDHDR,         /* id_unwindhdr */
2240                        M_ID_USER,              /* id_user */
2241                        M_ID_VERSION,           /* id_version */
2242                },
2243                {                           /* Target_nullfunc */
2244                        nullfunc_tmpl,          /* nf_template */
2245                        sizeof (nullfunc_tmpl), /* nf_size */
2246                },
2247                {                           /* Target_fillfunc */
2248                        /*
2249                         * On sparc, special filling of executable sections
2250                         * is undesirable, and the default 0 fill supplied
2251                         * by libelf is preferred:
2252                         *
2253                         * -    0 fill is interpreted as UNIMP instructions,
2254                         *      which cause an illegal_instruction_trap. These
2255                         *      serve as a sentinel against poorly written
2256                         *      code. The sparc architecture manual discusses
2257                         *      this as providing a measure of runtime safety.
2258                         *
2259                         * -    The one place where a hole should conceivably
2260                         *      be filled with NOP instructions is in the
2261                         *      .init/.fini sections. However, the sparc
2262                         *      assembler sizes the sections it generates
2263                         *      to a multiple of the section alignment, and as
2264                         *      such, takes the filling task out of our hands.
2265                         *      Furthermore, the sparc assembler uses 0-fill
2266                         *      for this, forcing the authors of sparc
2267                         *      assembler for .init/.fini sections to be aware
2268                         *      of this case and explicitly supply NOP fill.
2269                         *      Hence, there is no role for the link-editor.
2270                         */
2271                        NULL                    /* ff_execfill */
2272                },
2273                {                           /* Target_machrel */
2274                        reloc_table,

2276                        ld_init_rel,            /* mr_init_rel */
2277                        ld_mach_eflags,         /* mr_mach_eflags */
2278                        ld_mach_make_dynamic,   /* mr_mach_make_dynamic */
2279                        ld_mach_update_odynamic, /* mr_mach_update_odynamic */
2280                        ld_calc_plt_addr,       /* mr_calc_plt_addr */
2281                        ld_perform_outreloc,    /* mr_perform_outreloc */
2282                        ld_do_activerelocs,     /* mr_do_activerelocs */
2283                        ld_add_outrel,          /* mr_add_outrel */
2284                        ld_reloc_register,      /* mr_reloc_register */
2285                        ld_reloc_local,         /* mr_reloc_local */
2286                        ld_reloc_GOTOP,         /* mr_reloc_GOTOP */
2287                        ld_reloc_TLS,           /* mr_reloc_TLS */
```

```
2288                        ld_assign_got,          /* mr_assign_got */
2289                        ld_find_got_ndx,        /* mr_find_got_ndx */
2290                        ld_calc_got_offset,     /* mr_calc_got_offset */
2291                        ld_assign_got_ndx,      /* mr_assign_got_ndx */
2292                        ld_assign_plt_ndx,      /* mr_assign_plt_ndx */
2293                        ld_allocate_got,        /* mr_allocate_got */
2294                        ld_fillin_gotplt,       /* mr_fillin_gotplt */
2295                },
2296                {                       /* Target_machsym */
2297                        ld_reg_check_sparc,     /* ms_reg_check */
2298                        ld_mach_sym_typecheck_sparc, /* ms_mach_sym_typecheck */
2299                        ld_is_regsym_sparc,     /* ms_is_regsym */
2300                        ld_reg_find_sparc,      /* ms_reg_find */
2301                        ld_reg_enter_sparc      /* ms_reg_enter */
2302                }
2303        };

2305        return (&_ld_targ);
2306 }
```

```
**********************************************************
   89012 Wed May 22 03:21:45 2019
new/usr/src/cmd/sgs/packages/common/SUNWonld-README
11057 hidden undefined weak symbols should not leave relocations
11058 libld entrance descriptor assertions get NDEBUG check backwards
**********************************************************
```

```
#
# Copyright (c) 1996, 2010, Oracle and/or its affiliates. All rights reserved.
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Note: The contents of this file are used to determine the versioning
#       information for the SGS toolset. The number of CRs listed in
#       this file must grow monotonically, or the SGS version will
#       move backwards, causing a great deal of confusion. As such,
#       CRs must never be removed from this file. See
#       libconv/common/bld_vernote.ksh, and bug#4519569 for more
#       details on SGS versioning.
#
-------------------------------------------------------------------------------
SUNWonld - link-editors development package.
-------------------------------------------------------------------------------

  The SUNWonld package is an internal development package containing the
  link-editors and some related tools.  All components live in the OSNET
  source base, but not all components are delivered as part of the normal
  OSNET consolidation.  The intent of this package is to provide access
  to new features/bugfixes before they become generally available.

  General link-editor information can be found:

    http://linkers.central/
    http://linkers.sfbay/       (also known as linkers.eng)

  Comments and Questions:

    Contact Rod Evans, Ali Bahrami, and/or Seizo Sakurai.

  Warnings:

    The postremove script for this package employs /usr/sbin/static/mv,
    and thus, besides the common core dependencies, this package also
    has a dependency on the SUNWsutl package.

  Patches:

    If the patch has been made official, you'll find it in:

        http://sunsolve.east/cgi/show.pl?target=patches/os-patches
```



```
    If it hasn't been released, the patch will be in:

        /net/sunsoftpatch/patches/temporary

    Note, any patches logged here refer to the temporary ("T") name, as we
    never know when they're made official, and although we try to keep all
    patch information up-to-date the real status of any patch can be
    determined from:

        http://sunsoftpatch.eng

    If it has been obsoleted, the patch will be in:

        /net/on${RELEASE}-patch/on${RELEASE}/patches/${MACH}/obsolete


  History:

    Note, starting after Solaris 10, letter codes in parenthesis may
    be found following the bug synopsis. Their meanings are as follows:

        (D) A documentation change accompanies the implementation change.
        (P) A packaging change accompanies the implementation change.

    In all cases, see the implementation bug report for details.

    The following bug fixes exist in the OSNET consolidation workspace
    from which this package is created:

---------
Solaris 8
---------
Bugid   Risk Synopsis
================================================================================
4225937 i386 linker emits sparc specific warning messages
4215164 shf_order flag handling broken by fix for 4194028.
4215587 using ld and the -r option on solaris 7 with compiler option -xarch=v9
        causes link errors.
4234657 103627-08 breaks purify 4.2 (plt padding should not be enabled for
        32-bit)
4235241 dbx no longer gets dlclose notification.
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-05  (never released)
        Solaris/SunOS 5.7_x86           patch 106951-05  (never released)
        Solaris/SunOS 5.6_sparc         patch 107733-02  (never released)
        Solaris/SunOS 5.6_x86           patch 107734-02
--------------------------------------------------------------------------------
4248290 inetd dumps core upon bootup - failure in dlclose() logic.
4238071 dlopen() leaks while descriptors under low memory conditions
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-06
        Solaris/SunOS 5.7_x86           patch 106951-06
        Solaris/SunOS 5.6_sparc         patch 107733-03  (never released)
        Solaris/SunOS 5.6_x86           patch 107734-03
--------------------------------------------------------------------------------
4267980 INITFIRST flag of the shard object could be ignored.
--------------------------------------------------------------------------------
All the above changes plus:
    4238973 fix for 4121152 affects linking of Ada objects
    4158744 patch 103627-02 causes core when RPATH has blank entry and
            dlopen/dlclose is used
are incorporated in the following patches:
        Solaris/SunOS 5.5.1_sparc       patch 103627-12  (never released)
```

```
        Solaris/SunOS 5.5.1_x86         patch 103628-11
--------------------------------------------------------------------------------
4256518 miscalculated calloc() during dlclose/tsorting can result in segv
4254171 DT_SPARC_REGISTER has invalid value associated with it.
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-07
        Solaris/SunOS 5.7_x86           patch 106951-07
        Solaris/SunOS 5.6_sparc         patch 107733-04  (never released)
        Solaris/SunOS 5.6_x86           patch 107734-04
--------------------------------------------------------------------------------
4293159 ld needs to combine sections with and without SHF_ORDERED flag(comdat)
4292238 linking a library which has a static char ptr invokes mprotect() call
--------------------------------------------------------------------------------
All the above changes except for:
    4256518 miscalculated calloc() during dlclose/tsorting can result in segv
    4254171 DT_SPARC_REGISTER has invalid value associated with it.
plus:
    4238973 fix for 4121152 affects linking of Ada objects
    4158744 patch 103627-02 causes core when RPATH has blank entry and
            dlopen/dlclose is used
are incorporated in the following patches:
        Solaris/SunOS 5.5.1_sparc       patch 103627-13
        Solaris/SunOS 5.5.1_x86         patch 103628-12
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-08
        Solaris/SunOS 5.7_x86           patch 106951-08
        Solaris/SunOS 5.6_sparc         patch 107733-05
        Solaris/SunOS 5.6_x86           patch 107734-05
--------------------------------------------------------------------------------
4295613 COMMON symbol resolution can be incorrect
--------------------------------------------------------------------------------
All the above changes plus:
    4238973 fix for 4121152 affects linking of Ada objects
    4158744 patch 103627-02 causes core when RPATH has blank entry and
            dlopen/dlclose is used
are incorporated in the following patches:
        Solaris/SunOS 5.5.1_sparc       patch 103627-14
        Solaris/SunOS 5.5.1_x86         patch 103628-13
--------------------------------------------------------------------------------
All the above changes plus:
    4351197 nfs performance problem by 103627-13
are incorporated in the following patches:
        Solaris/SunOS 5.5.1_sparc       patch 103627-15
        Solaris/SunOS 5.5.1_x86         patch 103628-14
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-09
        Solaris/SunOS 5.7_x86           patch 106951-09
        Solaris/SunOS 5.6_sparc         patch 107733-06
        Solaris/SunOS 5.6_x86           patch 107734-06
--------------------------------------------------------------------------------
4158971 increase the default segment alignment for i386 to 64k
4064994 Add an $ISALIST token to those understood by the dynamic linker
xxxxxxx ia64 common code putback
4239308 LD_DEBUG busted for sparc machines
4239008 Support MAP_ANON
4238494 link-auditing extensions required
4232239 R_SPARC_LOX10 truncates field
4231722 R_SPARC_UA* relocations are busted
4235514 R_SPARC_OLO10 relocation fails
4244025 sgsmsg update
4239281 need to support SECREL relocations for ia64
4253751 ia64 linker must support PT_IA_64_UNWIND tables
4259254 dlmopen mistakenly closes fd 0 (stdin) under certain error conditions
```



```
4260872 libelf hangs when libthread present
4224569 linker core dumping when profiling specified
4270937 need mechanism to suppress ld.so.1's use of a default search path.
1050476 ld.so to permit configuration of search path
4273654 filtee processing using $ISALIST could be optimized
4271860 get MERCED cruft out of elf.h
4248991 Dynamic loader (via PLT) corrupts register G4
4275754 cannot mmap file: Resource temporarily unavailable
4277689 The linker can not handle relocation against MOVE tabl
4270766 atexit processing required on dlclose().
4279229 Add a "release" token to those understood by the dynamic linker
4215433 ld can bus error when insufficient disc space exists for output file
4285571 Pssst, want some free disk space?  ld's miscalculating.
4286236 ar gives confusing "bad format" error with a null .stab section
4286838 ld.so.1 can't handle a no-bits segment
4287364 ld.so.1 runtime configuration cleanup
4289573 disable linking of ia64 binaries for Solaris8
4293966 crle(1)'s default directories should be supplied
--------------------------------------------------------------------------------

-----------------------------------
Solaris 8 600 (1st Q-update - s28u1)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4309212 dlsym can't find symbol
4311226 rejection of preloading in secure apps is inconsistent
4312449 dlclose: invalid deletion of dependency can occur using RTLD_GLOBAL
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-01
        Solaris/SunOS 5.8_x86           patch 109148-01
        Solaris/SunOS 5.7_sparc         patch 106950-10
        Solaris/SunOS 5.7_x86           patch 106951-10
        Solaris/SunOS 5.6_sparc         patch 107733-07
        Solaris/SunOS 5.6_x86           patch 107734-07
--------------------------------------------------------------------------------

-----------------------------------
Solaris 8 900 (2nd Q-update - s28u2)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4324775 non-PIC code & -zcombreloc don't mix very well...
4327653 run-time linker should preload tables it will process (madvise)
4324324 shared object code can be referenced before .init has fired
4321634 .init firing of multiple INITFIRST objects can fail
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-03
        Solaris/SunOS 5.8_x86           patch 109148-03
        Solaris/SunOS 5.7_sparc         patch 106950-11
        Solaris/SunOS 5.7_x86           patch 106951-11
        Solaris/SunOS 5.6_sparc         patch 107733-08
        Solaris/SunOS 5.6_x86           patch 107734-08
--------------------------------------------------------------------------------
4338812 crle(1) omits entries in the directory cache
4341496 RFE: provide a static version of /usr/bin/crle
4340878 rtld should treat $ORIGIN like LD_LIBRARY_PATH in security issues
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-04
        Solaris/SunOS 5.8_x86           patch 109148-04
        Solaris/SunOS 5.7_sparc         patch 106950-12
        Solaris/SunOS 5.7_x86           patch 106951-12
--------------------------------------------------------------------------------
```

```
4349563 auxiliary filter error handling regression introduced in 4165487
4355795 ldd -r now gives "displacement relocated" warnings
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-13
        Solaris/SunOS 5.7_x86           patch 106951-13
        Solaris/SunOS 5.6_sparc         patch 107733-09
        Solaris/SunOS 5.6_x86           patch 107734-09
--------------------------------------------------------------------------------
4210412 versioning a static executable causes ld to core dump
4219652 Linker gives misleading error about not finding main (xarch=v9)
4103449 ld command needs a command line flag to force 64-bits
4187211 problem with RDISP32 linking in copy-relocated objects
4287274 dladdr, dlinfo do not provide the full path name of a shared object
4297563 dlclose still does not remove all objects.
4250694 rtld_db needs a new auxvec entry
4235315 new features for rtld_db (DT_CHECKSUM, dynamic linked .o files
4303609 64bit libelf.so.1 does not properly implement elf_hash()
4310901 su.static fails when OSNet build with lazy-loading
4310324 elf_errno() causes Bus Error(coredump) in 64-bit multithreaded programs
4306415 ld core dump
4316531 BCP: possible failure with dlclose/_preexec_exit_handlers
4313765 LD_BREADTH should be shot
4318162 crle uses automatic strings in putenv.
4255943 Description of -t option incomplete.
4322528 sgs message test infrastucture needs improvement
4239213 Want an API to obtain linker's search path
4324134 use of extern mapfile directives can contribute unused symbols
4322581 ELF data structures could be layed out more efficiently...
4040628 Unnecessary section header symbols should be removed from .dynsym
4300018 rtld: bindlock should be freed before calling call_fini()
4336102 dlclose with non-deletable objects can mishandle dependencies
4329785 mixing of SHT_SUNW_COMDAT & SHF_ORDERED causes ld to seg fault
4334617 COPY relocations should be produces for references to .bss symbols
4248250 relcoation of local ABS symbols incorrect
4335801 For complimentary alignments eliminate ld: warning: symbol `ll'
        has differing a
4336980 ld.so.1 relative path processing revisited
4243097 dlerror(3DL) is not affected by setlocale(3C).
4344528 dump should remove -D and -l usage message
xxxxxxx enable LD_ALTEXEC to access alternate link-editor
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-06
        Solaris/SunOS 5.8_x86           patch 109148-06
--------------------------------------------------------------------------------

-----------------------------------
Solaris 8 101 (3rd Q-update - s28u3)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4346144 link-auditing: plt_tracing fails if LA_SYMB_NOPLTENTER given after
        being bound
4346001 The ld should support mapfile syntax to generate PT_SUNWSTACK segment
4349137 rtld_db: A third fallback method for locating the linkmap
4343417 dladdr interface information inadequate
4343801 RFE: crle(1): provide option for updating configuration files
4346615 ld.so.1 attempting to open a directory gives: No such device
4352233 crle should not honor umask
4352330 LD_PRELOAD cannot use absolute path for privileged program
4357805 RFE: man page for ld(1) does not document all -z or -B options in
        Solaris 8 9/00
4358751 ld.so.1: LD_XXX environ variables and LD_FLAGS should be synchronized.
4358862 link editors should reference "64" symlinks instead of sparcv9 (ia64).
4356879 PLTs could use faster code sequences in some cases
```



```
4367118 new fast baplt's fail when traversed twice in threaded application
4366905 Need a way to determine path to a shared library
4351197 nfs performance problem by 103627-13
4367405 LD_LIBRARY_PATH_64 not being used
4354500 SHF_ORDERED ordered scections does not properly sort sections
4369068 ld(1)'s weak symbol processing is inefficient (slow and doesn't scale).
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-07
        Solaris/SunOS 5.8_x86           patch 109148-07
        Solaris/SunOS 5.7_sparc         patch 106950-14
        Solaris/SunOS 5.7_x86           patch 106951-14
--------------------------------------------------------------------------------

-----------------------------------
Solaris 8 701 (5th Q-update - s28u5)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4368846 ld(1) fails to version some interfaces given in a mapfile
4077245 dump core dump on null pointer.
4372554 elfdump should demangle symbols (like nm, dump)
4371114 dlclose may unmap a promiscuous object while it's still in use.
4204447 elfdump should understand SHN_AFTER/SHN_BEGIN macro
4377941 initialization of interposers may not occur
4381116 ldd/ld.so.1 could aid in detecting unused dependencies
4381783 dlopen/dlclose of a libCrun+libthread can dump core
4385402 linker & run-time linker must support gABI ELF updates
4394698 ld.so.1 does not process DF_SYMBOLIC - not gABI conforming
4394212 the link editor quietly ignores missing support libraries
4390308 ld.so.1 should provide more flexibility LD_PRELOAD'ing 32-bit/64-bit
        objects
4401232 crle(1) could provide better flexibility for alternatives
4401815 fix misc nits in debugging output...
4402861 cleanup /usr/demo/link_audit & /usr/tmp/librtld_db demo source code...
4393044 elfdump should allow raw dumping of sections
4413168 SHF_ORDERED bit causes linker to generate a separate section
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-08
        Solaris/SunOS 5.8_x86           patch 109148-08
--------------------------------------------------------------------------------
4452202 Typos in <sys/link.h>
4452220 dump doesn't support RUNPATH
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-09
        Solaris/SunOS 5.8_x86           patch 109148-09
--------------------------------------------------------------------------------

-----------------------------------
Solaris 8 1001 (6th Q-update - s28u6)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4421842 fixups in SHT_GROUP processing required...
4450433 problem with liblddbg output on -Dsection,detail when
        processing SHF_LINK_ORDER
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.8_sparc         patch 109147-10
        Solaris/SunOS 5.8_x86           patch 109148-10
        Solaris/SunOS 5.7_sparc         patch 106950-15
        Solaris/SunOS 5.7_x86           patch 106951-15
--------------------------------------------------------------------------------
4463473 pldd showing wrong output
```

```
391 ------------------------------------------------------------------------------
392 All the above changes are incorporated in the following patches:
393         Solaris/SunOS 5.8_sparc         patch 109147-11
394         Solaris/SunOS 5.8_x86           patch 109148-11
395 ------------------------------------------------------------------------------

397 -----------------------------------
398 Solaris 8 202 (7th Q-update - s28u7)
399 -----------------------------------
400 Bugid   Risk Synopsis
401 ==============================================================================
402 4488954 ld.so.1 reuses same buffer to send ummapping range to
403         _preexec_exit_handlers()
404 ------------------------------------------------------------------------------
405 All the above changes are incorporated in the following patches:
406         Solaris/SunOS 5.8_sparc         patch 109147-12
407         Solaris/SunOS 5.8_x86           patch 109148-12
408 ------------------------------------------------------------------------------

410 ---------
411 Solaris 9
412 ---------
413 Bugid   Risk Synopsis
414 ==============================================================================
415 4505289 incorrect handling of _START_ and _END_
416 4506164 mcs does not recognize #linkbefore or #linkafter qualifiers
417 4447560 strip is creating unexecutable files...
418 4513842 library names not in ld.so string pool cause corefile bugs
419 ------------------------------------------------------------------------------
420 All the above changes are incorporated in the following patches:
421         Solaris/SunOS 5.8_sparc         patch 109147-13
422         Solaris/SunOS 5.8_x86           patch 109148-13
423         Solaris/SunOS 5.7_sparc         patch 106950-16
424         Solaris/SunOS 5.7_x86           patch 106951-16
425 ------------------------------------------------------------------------------
426 4291384 ld -M with a mapfile does not properly align Fortran REAL*8 data
427 4413322 SunOS 5.9 librtld_db doesn't show dlopened ".o" files anymore?
428 4429371 librtld_db busted on ia32 with SC6.x compilers...
429 4418274 elfdump dumps core on invalid input
430 4432224 libelf xlate routines are out of date
431 4433643 Memory leak using dlopen()/dlclose() in Solaris 8
432 4446564 ldd/lddstub - core dump conditions
433 4446115 translating SUNW_move sections is broken
434 4450225 The rdb command can fall into an infinite loop
435 4448531 Linker Causes Segmentation Fault
436 4453241 Regression in 4291384 can result in empty symbol table.
437 4453398 invalid runpath token can cause ld to spin.
438 4460230 ld (for OS 5.8 and 5.9) loses error message
439 4462245 ld.so.1 core dumps when executed directly...
440 4455802 need more flexibility in establishing a support library for ld
441 4467068 dyn_plt_entsize not properly initialized in ld.so.1
442 4468779 elf_plt_trace_write() broken on i386 (link-auditing)
443 4465871 -zld32 and -zld64 does not work the way it should
444 4461890 bad shared object created with -zredlocsym
445 4469400 ld.so.1: is_so_loaded isn't as efficient as we thought...
446 4469566 lazy loading fallback can reference un-relocated objects
447 4470493 libelf incorectly translates NOTE sections accross architectures...
448 4469684 rtld leaks dl_handles and permits on dlopen/dlclose
449 4475174 ld.so.1 prematurly reports the failure to load a object...
450 4475514 ld.so.1 can core dump in memory allocation fails (no swap)
451 4481851 Setting ld.so.1 environment variables globally would be useful
452 4482035 setting LD_PROFILE & LD_AUDIT causes ping command to issue warnings
453         on 5.8
454 4377735 segment reservations cause sbrk() to fail
455 4491434 ld.so.1 can leak file-descriptors when loading same named objects
456 4289232 some of warning/error/debugging messages from libld.so can be revised
```



```
457 4462748 Linker Portion of TLS Support
458 4496718 run-time linkers mutex_locks not working with ld_libc interface
459 4497270 The -zredlocsym option should not eliminate partially initialized local
460         symbols
461 4496963 dumping an object with crle(1) that uses $ORIGIN can loose its
462         dependencies
463 4499413 Sun linker orders of magnitude slower than gnu linker
464 4461760 lazy loading libXm and libXt can fail.
465 4469031 The partial initialized (local) symbols for intel platform is not
466         working.
467 4492883 Add link-editor option to multi-pass archives to resolve unsatisfied
468         symbols
469 4503731 linker-related commands misspell "argument"
470 4503768 whocalls(1) should output messages to stderr, not stdout
471 4503748 whocalls(1) usage message and manpage could be improved
472 4503625 nm should be taught about TLS symbols - that they aren't allowed that is
473 4300120 segment address validation is too simplistic to handle segment
474         reservations
475 4404547 krtld/reloc.h could have better error message, has typos
476 4270931 R_SPARC_HIX22 relocation is not handled properly
477 4485320 ld needs to support more the 32768 PLTs
478 4516434 sotruss can not watch libc_psr.so.1
479 4213100 sotruss could use more flexible pattern matching
480 4503457 ld seg fault with comdat
481 4510264 sections with SHF_TLS can come in different orders...
482 4518079 link-editor support library unable to modify section header flags
483 4515913 ld.so.1 can incorrectly decrement external reference counts on dlclose()
484 4519569 ld -V does not return a interesting value...
485 4524512 ld.so.1 should allow alternate termination signals
486 4524767 elfdump dies on bogus sh_name fields...
487 4524735 ld getopt processing of '-' changed
488 4521931 subroutine in a shared object as LOCL instead of GLOB
489 ------------------------------------------------------------------------------
490 All the above changes are incorporated in the following patches:
491         Solaris/SunOS 5.8_sparc         patch 109147-14
492         Solaris/SunOS 5.8_x86           patch 109148-14
493         Solaris/SunOS 5.7_sparc         patch 106950-17
494         Solaris/SunOS 5.7_x86           patch 106951-17
495 ------------------------------------------------------------------------------
496 4532729 tentative definition of TLS variable causes linker to dump core
497 4526745 fixup ld error message about duplicate dependencies/needed names
498 4522999 Solaris linker one order of magnitude slower than GNU linker
499 4518966 dldump undoes existing relocations with no thought of alignment or size.
500 4587441 Certain libraries have race conditions when setting error codes
501 4523798 linker option to align bss to large pagesize alignments.
502 4524008 ld can improperly set st_size of symbols named "_init" or "_fini"
503 4619282 ld cannot link a program with the option -sb
504 4620846 Perl Configure probing broken by ld changes
505 4621122 multiple ld '-zinitarray=' on a commandline fails
506 ------------------------------------------------------------------------------
507         Solaris/SunOS 5.8_sparc         patch 109147-15
508         Solaris/SunOS 5.8_x86           patch 109148-15
509         Solaris/SunOS 5.7_sparc         patch 106950-18
510         Solaris/SunOS 5.7_x86           patch 106951-18
511         Solaris/SunOS 5.6_sparc         patch 107733-10
512         Solaris/SunOS 5.6_x86           patch 107734-10
513 ------------------------------------------------------------------------------
514 All the above changes plus:
515         4616944 ar seg faults when order of object file is reversed.
516 are incorporated in the following patches:
517         Solaris/SunOS 5.8_sparc         patch 109147-16
518         Solaris/SunOS 5.8_x86           patch 109148-16
519 ------------------------------------------------------------------------------
520 All the above changes plus:
521         4872634 Large LD_PRELOAD values can cause SEGV of process
522 are incorporated in the following patches:
```

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

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

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



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

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

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

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



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

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

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

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



```
 853 ----------------------------------------------------------------------------
 854 All the above changes are incorporated in the following patches:
 855         Solaris/SunOS 5.9_sparc         patch T112963-15
 856         Solaris/SunOS 5.9_x86           patch T113986-11
 857         Solaris/SunOS 5.8_sparc         patch T109147-30
 858         Solaris/SunOS 5.8_x86           patch T109148-30
 859 ----------------------------------------------------------------------------
 860 5071614 109147-29 & -30 break the build of on28-patch on Solaris 8 2/04
 861 5029830 crle: provide for optional alternative dependencies.
 862 5034652 ld.so.1 should save, and print, more error messages
 863 5036561 ld.so.1 outputs non-fatal fatal message about auxiliary filter libraries
 864 5042713 4866170 broke ld.so's ::setenv
 865 5047082 ld can core dump on bad gcc objects
 866 5047612 ld.so.1: secure pathname verification is flawed with filter use
 867 5047235 elfdump can core dump printing PT_INTERP section
 868 4798376 nits in demo code
 869 5041446 gelf_update_*() functions inconsistently return NULL or 0
 870 5032364 M_ID_TLSBSS and M_ID_UNKNOWN have the same value
 871 4707030 Empty LD_PRELOAD_64 doesn't override LD_PRELOAD
 872 4968618 symbolic linkage causes core dump
 873 5062313 dladdr() can cause deadlock in MT apps.
 874 5056867 $ISALIST/$HWCAP expansion should be more flexible.
 875 4918303 0@0.so.1 should not use compiler-supplied crt*.o files
 876 5058415 whocalls cannot take more than 10 arguments
 877 5067518 The fix for 4918303 breaks the build if a new work space is used.
 878 ----------------------------------------------------------------------------
 879 All the above changes are incorporated in the following patches:
 880         Solaris/SunOS 5.9_sparc         patch T112963-16
 881         Solaris/SunOS 5.9_x86           patch T113986-12
 882         Solaris/SunOS 5.8_sparc         patch T109147-31
 883         Solaris/SunOS 5.8_x86           patch T109148-31
 884 ----------------------------------------------------------------------------
 885 5013759 *file* should report hardware/software capabilities (link-editor
 886         components only)
 887 5063580 libldstab: file /tmp/posto..: .stab[.index|.sbfocus] found with no
 888         matching stri
 889 5076838 elfdump(1) is built with a CTF section (the wrong one)
 890 5080344 Hardware capabilities are not enforced for a.out
 891 5079061 RTLD_DEFAULT can be expensive
 892         PSARC/2004/747 New dlsym(3c) Handle - RTLD_PROBE
 893 5064973 allow normal relocs against TLS symbols for some sections
 894 5085792 LD_XXXX_64 should override LD_XXXX
 895 5096272 every executable or library has a .SUNW_dof section
 896 5094135 Bloomberg wants a faster ldd.
 897 5086352 libld.so.3 should be built with a .SUNW_ctf ELF section, ready for CR
 898 5098205 elfdump gives wrong section name for the global offset table
 899 5092414 Linker patch 109147-29 makes Broadvison One-To-One server v4.1
 900         installation fail
 901 5080256 dump(1) doesn't list ELF hardware capabilities
 902 5097347 recursive read lock in gelf_getsym()
 903 ----------------------------------------------------------------------------
 904 All the above changes are incorporated in the following patches:
 905         Solaris/SunOS 5.9_sparc         patch T112963-17
 906         Solaris/SunOS 5.9_x86           patch T113986-13
 907         Solaris/SunOS 5.8_sparc         patch T109147-32
 908         Solaris/SunOS 5.8_x86           patch T109148-32
 909 ----------------------------------------------------------------------------
 910 5106206 ld.so.1 fail to run a Solaris9 program that has libc linked with
 911         -z lazyload
 912 5102601 ON should deliver a 64-bit operating system for Opteron systems
 913         (link-editor components only)
 914 6173852 enable link_auditing technology for amd64
 915 6174599 linker does not create .eh_frame_hdr sections for eh_frame sections
 916         with SHF_LINK_ORDER
 917 6175609 amd64 run-time linker has a corrupted note section
 918 6175843 amd64 rdb_demo files not installed
```

```
 919 6182293 ld.so.1 can repeatedly relocate object .plts (RTLD_NOW).
 920 6183645 ld core dumps when automounter fails
 921 6178667 ldd list unexpected (file not found) in x86 environment.
 922 6181928 Need new reloc types R_AMD64_GOTOFF64 and R_AMD64_GOTPC32
 923 6182884 AMD64: ld coredumps when building a shared library
 924 6173559 The ld may set incorrect value for sh_addralign under some conditions.
 925 5105601 ld.so.1 gets a little too enthusiastic with interposition
 926 6189384 ld.so.1 should accommodate a files dev/inode change (libc loopback mnt)
 927 6177838 AMD64: linker cannot resolve PLT for 32-bit a.out(s) on amd64-S2 kernel
 928 6190863 sparc disassembly code should be removed from rdb_demo
 929 6191488 unwind eh_frame_hdr needs corrected encoding value
 930 6192490 moe(1) returns /lib/libc.so.1 for optimal expansion of libc HWCAP
 931         libraries
 932 6192164 AMD64: introduce dlamd64getunwind interface
 933         PSARC/2004/747 libc::dlamd64getunwind()
 934 6195030 libdl has bad version name
 935 6195521 64-bit moe(1) missed the train
 936 6198358 AMD64: bad eh_frame_hdr data when C and C++ mixed in a.out
 937 6204123 ld.so.1: symbol lookup fails even after lazy loading fallback
 938 6207495 UNIX98/UNIX03 vsx namespace violation DYNL.hdr/misc/dlfcn/T.dlfcn
 939         14 Failed
 940 6217285 ctfmerge crashed during full onnv build
 941 ----------------------------------------------------------------------------

 943 ------------------------------------
 944 Solaris 10 106 (1st Q-update - s10u1)
 945 ------------------------------------
 946 Bugid   Risk Synopsis
 947 ============================================================================
 948 6209350 Do not include signature section from dynamic dependency library into
 949         relocatable object
 950 6212797 The binary compiled on SunOS4.x doesn't run on Solaris8 with Patch
 951         109147-31
 952 ----------------------------------------------------------------------------
 953 All the above changes are incorporated in the following patches:
 954         Solaris/SunOS 5.9_sparc         patch T112963-18
 955         Solaris/SunOS 5.9_x86           patch T113986-14
 956         Solaris/SunOS 5.8_sparc         patch T109147-33
 957         Solaris/SunOS 5.8_x86           patch T109148-33
 958 ----------------------------------------------------------------------------
 959 6219538 112963-17: linker patch causes binary to dump core
 960 ----------------------------------------------------------------------------
 961 All the above changes are incorporated in the following patches:
 962         Solaris/SunOS 5.10_sparc        patch T117461-01
 963         Solaris/SunOS 5.10_x86          patch T118345-01
 964         Solaris/SunOS 5.9_sparc         patch T112963-19
 965         Solaris/SunOS 5.9_x86           patch T113986-15
 966         Solaris/SunOS 5.8_sparc         patch T109147-34
 967         Solaris/SunOS 5.8_x86           patch T109148-34
 968 ----------------------------------------------------------------------------
 969 6257177 incremental builds of usr/src/cmd/sgs can fail...
 970 6219651 AMD64: Linker does not issue error for out of range R_AMD64_PC32
 971 ----------------------------------------------------------------------------
 972 All the above changes are incorporated in the following patches:
 973         Solaris/SunOS 5.10_sparc        patch T117461-02
 974         Solaris/SunOS 5.10_x86          patch T118345-02
 975         Solaris/SunOS 5.9_sparc         patch T112963-20
 976         Solaris/SunOS 5.9_x86           patch T113986-16
 977         Solaris/SunOS 5.8_sparc         patch T109147-35
 978         Solaris/SunOS 5.8_x86           patch T109148-35
 979 NOTE: The fix for 6219651 is only applicable for 5.10_x86 platform.
 980 ----------------------------------------------------------------------------
 981 5080443 lazy loading failure doesn't clean up after itself (D)
 982 6226206 ld.so.1 failure when processing single segment hwcap filtee
 983 6228472 ld.so.1: link-map control list stacking can loose objects
 984 6235000 random packages not getting installed in snv_09 and snv_10 -
```



```
 985         rtld/common/malloc.c Assertion
 986 6219317 Large page support is needed for mapping executables, libraries and
 987         files (link-editor components only)
 988 6244897 ld.so.1 can't run apps from commandline
 989 6251798 moe(1) returns an internal assertion failure message in some
 990         circumstances
 991 6251722 ld fails silently with exit 1 status when -z ignore passed
 992 6254364 ld won't build libgenunix.so with absolute relocations
 993 6215444 ld.so.1 caches "not there" lazy libraries, foils svc.startd(1M)'s logic
 994 6222525 dlsym(3C) trusts caller(), which may return wrong results with tail call
 995         optimization
 996 6241995 warnings in sgs should be fixed (link-editor components only)
 997 6258834 direct binding availability should be verified at runtime
 998 6260361 lari shouldn't count a.out non-zero undefined entries as interesting
 999 6260780 ldd doesn't recognize LD_NOAUXFLTR
1000 6266261 Add ld(1) -Bnodirect support (D)
1001 6261990 invalid e_flags error could be a little more friendly
1002 6261803 lari(1) should find more events uninteresting (D)
1003 6267352 libld_malloc provides inadequate alignment
1004 6268693 SHN_SUNW_IGNORE symbols should be allowed to be mulitiply defined
1005 6262789 Infosys wants a faster linker
1006 ----------------------------------------------------------------------------
1007 All the above changes are incorporated in the following patches:
1008         Solaris/SunOS 5.10_sparc        patch T117461-03
1009         Solaris/SunOS 5.10_x86          patch T118345-03
1010         Solaris/SunOS 5.9_sparc         patch T112963-21
1011         Solaris/SunOS 5.9_x86           patch T113986-17
1012         Solaris/SunOS 5.8_sparc         patch T109147-36
1013         Solaris/SunOS 5.8_x86           patch T109148-36
1014 ----------------------------------------------------------------------------
1015 6283601 The usr/src/cmd/sgs/packages/common/copyright contains old information
1016         legally problematic
1017 6276905 dlinfo gives inconsistent results (relative vs absolute linkname) (D)
1018         PSARC/2005/357 dlinfo(3c) RTLD_DI_ARGSINFO
1019 6284941 excessive link times with many groups/sections
1020 6280467 dlclose() unmaps shared library before library's _fini() has finished
1021 6291547 ld.so mishandles LD_AUDIT causing security problems.
1022 ----------------------------------------------------------------------------
1023 All the above changes are incorporated in the following patches:
1024         Solaris/SunOS 5.10_sparc        patch T117461-04
1025         Solaris/SunOS 5.10_x86          patch T118345-04
1026         Solaris/SunOS 5.9_sparc         patch T112963-22
1027         Solaris/SunOS 5.9_x86           patch T113986-18
1028         Solaris/SunOS 5.8_sparc         patch T109147-37
1029         Solaris/SunOS 5.8_x86           patch T109148-37
1030 ----------------------------------------------------------------------------
1031 6295971 UNIX98/UNIX03 *vsx* DYNL.hdr/misc/dlfcn/T.dlfcn 14 fails, auxv.h syntax
1032         error
1033 6299525 .init order failure when processing cycles
1034 6273855 gcc and sgs/crle don't get along
1035 6273864 gcc and sgs/libld don't get along
1036 6273875 gcc and sgs/rtld don't get along
1037 6272563 gcc and amd64/krtld/doreloc.c don't get along
1038 6290157 gcc and sgs/librtld_db/rdb_demo don't get along
1039 6301218 Matlab dumps core on startup when running on 112963-22 (D)
1040 ----------------------------------------------------------------------------
1041 All the above changes are incorporated in the following patches:
1042         Solaris/SunOS 5.10_sparc        patch T117461-06
1043         Solaris/SunOS 5.10_x86          patch T118345-08
1044         Solaris/SunOS 5.9_sparc         patch T112963-23
1045         Solaris/SunOS 5.9_x86           patch T113986-19
1046         Solaris/SunOS 5.8_sparc         patch T109147-38
1047         Solaris/SunOS 5.8_x86           patch T109148-38
1048 ----------------------------------------------------------------------------
1049 6314115 Checkpoint refuses to start, crashes on start, after application of
1050         linker patch 112963-22
```

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

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



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

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

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

1179 ------------------------------------
1180 Solaris 10 508 (5th Q-update - s10u5)
1181 ------------------------------------
1182 Bugid   Risk Synopsis
```

```
1183 ============================================================================
1184 6561987 data vac_conflict faults on lipthread libthread libs in s10.
1185 ----------------------------------------------------------------------------
1186 All the above changes are incorporated in the following patches:
1187         Solaris/SunOS 5.10_sparc        patch T127111-01
1188         Solaris/SunOS 5.10_x86          patch T127112-01
1189 ----------------------------------------------------------------------------
1190 6501793 GOTOP relocation transition (optimization) fails with offsets > 2^32
1191 6532924 AMD64: Solaris 5.11 55b: SEGV after whocatches
1192 6551627 OGL: SIGSEGV when trying to use OpenGL pipeline with splash screen,
1193         Solaris/Nvidia only
1194 ----------------------------------------------------------------------------
1195 All the above changes are incorporated in the following patches:
1196         Solaris/SunOS 5.10_sparc        patch T127111-04
1197         Solaris/SunOS 5.10_x86          patch T127112-04
1198 ----------------------------------------------------------------------------
1199 6479848 Enhancements to the linker support interface needed. (D)
1200         PSARC/2006/595 link-editor support library interface - ld_open()
1201 6521608 assertion failure in runtime linker related to auditing
1202 6494228 pclose() error when an audit library calls popen() and the main target
1203         is being run under ldd (D)
1204 6568745 segfault when using LD_DEBUG with bit_audit library when instrumenting
1205         mozilla (D)
1206         PSARC/2007/413 Add -zglobalaudit option to ld
1207 6602294 ps_pbrandname breaks apps linked directly against librtld_db
1208 ----------------------------------------------------------------------------
1209 All the above changes are incorporated in the following patches:
1210         Solaris/SunOS 5.10_sparc        patch T127111-07
1211         Solaris/SunOS 5.10_x86          patch T127112-07
1212 ----------------------------------------------------------------------------

1214 ------------------------------------
1215 Solaris 10 908 (6th Q-update - s10u6)
1216 ------------------------------------
1217 Bugid   Risk Synopsis
1218 ============================================================================
1219 6672544 elf_rtbndr must support non-ABI aligned stacks on amd64
1220 6668050 First trip through PLT does not preserve args in xmm registers
1221 ----------------------------------------------------------------------------
1222 All the above changes are incorporated in the following patch:
1223         Solaris/SunOS 5.10_x86          patch T137138-01
1224 ----------------------------------------------------------------------------

1226 ------------------------------------
1227 Solaris 10 409 (7th Q-update - s10u7)
1228 ------------------------------------
1229 Bugid   Risk Synopsis
1230 ============================================================================
1231 6629404 ld with -z ignore doesn't scale
1232 6606203 link editor ought to allow creation of >2gb sized objects (P)
1233 ----------------------------------------------------------------------------
1234 All the above changes are incorporated in the following patches:
1235         Solaris/SunOS 5.10_sparc        patch T139574-01
1236         Solaris/SunOS 5.10_x86          patch T139575-01
1237 ----------------------------------------------------------------------------
1238 6746674 setuid applications do not find libraries any more because trusted
1239         directories behavior changed (D)
1240 ----------------------------------------------------------------------------
1241 All the above changes are incorporated in the following patches:
1242         Solaris/SunOS 5.10_sparc        patch T139574-02
1243         Solaris/SunOS 5.10_x86          patch T139575-02
1244 ----------------------------------------------------------------------------
1245 6703683 Can't build VirtualBox on Build 88 or 89
1246 6737579 process_req_lib() in libld consumes file descriptors
1247 6685125 ld/elfdump do not handle ZERO terminator .eh_frame amd64 unwind entry
1248 ----------------------------------------------------------------------------
```



```
1249 All the above changes are incorporated in the following patches:
1250         Solaris/SunOS 5.10_sparc        patch T139574-03
1251         Solaris/SunOS 5.10_x86          patch T139575-03
1252 ----------------------------------------------------------------------------

1254 ------------------------------------
1255 Solaris 10 1009 (8th Q-update - s10u8)
1256 ------------------------------------
1257 Bugid   Risk Synopsis
1258 ============================================================================
1259 6782597 32-bit ld.so.1 needs to accept objects with large inode number
1260 6805502 The addition of "inline" keywords to sgs code broke the lint
1261         verification in S10
1262 6807864 ld.so.1 is susceptible to a fatal dlsym()/setlocale() race
1263 ----------------------------------------------------------------------------
1264 All the above changes are incorporated in the following patches:
1265         Solaris/SunOS 5.10_sparc        patch T141692-01
1266         Solaris/SunOS 5.10_x86          patch T141693-01
1267 NOTE: The fix for 6805502 is only applicable to s10.
1268 ----------------------------------------------------------------------------
1269 6826410 ld needs to sort sections using 32-bit sort keys
1270 ----------------------------------------------------------------------------
1271 All the above changes are incorporated in the following patches:
1272         Solaris/SunOS 5.10_sparc        patch T141771-01
1273         Solaris/SunOS 5.10_x86          patch T141772-01
1274 NOTE: The fix for 6826410 is also available for s9 in the following patches:
1275         Solaris/SunOS 5.9_sparc         patch T112963-33
1276         Solaris/SunOS 5.9_x86           patch T113986-27
1277 ----------------------------------------------------------------------------
1278 6568447 bcp is broken by 6551627
1279 6599700 librtld_db needs better plugin support
1280 6713830 mdb dumped core reading a gcore
1281 6756048 rd_loadobj_iter() should always invoke brand plugin callback
1282 6786744 32-bit dbx failed with unknown rtld_db.so error on snv_104
1283 ----------------------------------------------------------------------------
1284 All the above changes are incorporated in the following patches:
1285         Solaris/SunOS 5.10_sparc        patch T141444-06
1286         Solaris/SunOS 5.10_x86          patch T141445-06
1287 ----------------------------------------------------------------------------

1289 ------------------------------------
1290 Solaris 10 1005 (9th Q-update - s10u9)
1291 ------------------------------------
1292 Bugid   Risk Synopsis
1293 ============================================================================
1294 6850124 dlopen reports "No such file or directory" in spite of ENOMEM
1295         when mmap fails in anon_map()
1296 6826513 ldd gets confused by a crle(1) LD_PRELOAD setting
1297 6684577 ld should propagate SHF_LINK_ORDER flag to ET_REL objects
1298 6524709 executables using /usr/lib/libc.so.1 as the ELF interpreter dump core
1299         (link-editor components only)
1300 ----------------------------------------------------------------------------
1301 All the above changes are incorporated in the following patches:
1302         Solaris/SunOS 5.10_sparc        patch T143895-01
1303         Solaris/SunOS 5.10_x86          patch T143896-01
1304 ----------------------------------------------------------------------------

1306 ---------------------------------------
1307 Solaris 10 XXXX (10th Q-update - s10u10)
1308 ---------------------------------------
1309 Bugid   Risk Synopsis
1310 ============================================================================
1311 6478684 isainfo/cpuid reports pause instruction not supported on amd64
1312         PSARC/2010/089 Removal of AV_386_PAUSE and AV_386_MON
1313 ----------------------------------------------------------------------------
1314 All the above changes are incorporated in the following patches:
```

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

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



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

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

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

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

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



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

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

```
1579 6903688 mdb(1) can't resolve certain symbols in solaris10-branded processes
1580         from the global zone
1581 6923449 elfdump misinterprets _init/_fini symbols in dynamic section test
1582 6914728 Add dl_iterate_phdr() function to ld.so.1 (D)
1583         PSARC/2010/015 dl_iterate_phdr
1584 6916788 ld version 2 mapfile syntax (D)
1585         PSARC/2009/688 Human readable and extensible ld mapfile syntax
1586 6929607 ld generates incorrect VERDEF entries for ET_REL output objects
1587 6924224 linker should ignore SUNW_dof when calculating the elf checksum
1588 6918143 symbol capabilities (D)
1589         PSARC/2010/022 Linker-editors: Symbol Capabilities
1590 6910387 .tdata and .tbss separation invalidates TLS program header information
1591 6934123 elfdump -d coredumps on PA-RISC elf
1592 6931044 ld should not allow SHT_PROGBITS .eh_frame sections on amd64 (D)
1593 6931056 pvs -r output can include empty versions in output
1594 6938628 ld.so.1 should produce diagnostics for all dl*() entry points
1595 6938111 nm 'No symbol table data' message goes to stdout
1596 6941727 ld relocation cache memory use is excessive
1597 6932220 ld -z allextract skips objects that lack global symbols
1598 6943772 Testing for a symbols existence with RTLD_PROBE is compromised by
1599         RTLD_BIND_NOW
1600         PSARC/2010/XXX Deferred symbol references
1601 6943432 dlsym(RTLD_PROBE) should only bind to symbol definitions
1602 6668759 an external method for determining whether an ELF dependency is optional
1603 6954032 Support library with ld_open and -z allextract in snv_139 do not mix
1604 6949596 wrong section alignment generated in joint compilation with shared
1605         library
1606 6961755 ld.so.1's -e arguments should take precedence over environment
1607         variables. (D)
1608 6748925 moe returns wrong hwcap library in some circumstances
1609 6916796 OSnet mapfiles should use version 2 link-editor syntax
1610 6964517 OSnet mapfiles should use version 2 link-editor syntax (2nd pass)
1611 6948720 SHT_INIT_ARRAY etc. section names don't follow ELF gABI (D)
1612 6962343 sgsmsg should use mkstemp() for temporary file creation
1613 6965723 libsoftcrypto symbol capabilities rely on compiler generated
1614         capabilities - gcc failure (link-editor components only)
1615 6952219 ld support for archives larger than 2 GB (D, P)
1616         PSARC/2010/224 Support for archives larger than 2 GB
1617 6956152 dlclose() from an auditor can be fatal. Preinit/activity events should
1618         be more flexible. (D)
1619 6971440 moe can core dump while processing libc.
1620 6972234 sgs demo's could use some cleanup
1621 6935867 .dynamic could be readonly in sharable objects
1622 6975290 ld mishandles GOT relocation against local ABS symbol
1623 6972860 ld should provide user guidance to improve objects (D)
1624         PSARC/2010/312 Link-editor guidance
1625 --------------------------------------------------------------------------------

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

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



```
1645 3436    relocatable objects also need sloppy relocation
1646 3451    archive libraries with no symbols shouldn't require a string table
1647 3616    SHF_GROUP sections should not be discarded via other COMDAT mechanisms
1648 3709    need sloppy relocation for GNU .debug_macro
1649 3722    link-editor is over restrictive of R_AMD64_32 addends
1650 3926    multiple extern map file definitions corrupt symbol table entry
1651 3999    libld extended section handling is broken
1652 4003    dldump() can't deal with extended sections
1653 4227    ld --library-path is translated to -l-path, not -L
1654 4270    ld(1) argument error reporting is still pretty bad
1655 4383    libelf can't write extended sections when ELF_F_LAYOUT
1656 4959    completely discarded merged string sections will corrupt output objects
1657 4996    rtld _init race leads to incorrect symbol values
1658 5688    ELF tools need to be more careful with dwarf data
1659 6098    ld(1) should not require symbols which identify group sections be global
1660 6252    ld should merge function/data-sections in the same manner as GNU ld
1661 7323    ld(1) -zignore can erroneously discard init and fini arrays as unreferen
1662 7594    ld -zaslr should accept Solaris-compatible values
1663 8616    ld has trouble parsing -z options specified with -Wl
1664 10267   ld and GCC disagree about i386 local dynamic TLS
1665 10471   ld(1) amd64 LD->LE TLS transition causes memory corruption
1666 10346   ld(1) should not reduce symbol visibility of COMDAT symbols when
1667         producing relocatable objects
1668 10366   ld(1) should support GNU-style linker sets
1669 10581   ld(1) should know kernel modules are a thing
1670 11057   hidden undefined weak symbols should not leave relocations
1671 #endif /* ! codereview */
```