```
**********************************************************
   38518 Thu Feb  7 01:35:55 2019
new/usr/src/cmd/sgs/libld/common/_libld.h
10346 ld(1) should not reduce symbol visibility of COMDAT symbols when producing
**********************************************************
______unchanged_portion_omitted_

 646 /*
 647  * Local data items.
 648  */
 649 extern char             *Plibpath;
 650 extern char             *Llibdir;
 651 extern char             *Ulibdir;
 652 extern Ld_heap          *ld_heap;
 653 extern APlist           *lib_support;
 654 extern int              demangle_flag;
 655 extern const Msg        reject[];
 656 extern int              Verbose;
 657 extern const int        ldynsym_symtype[];
 658 extern const int        dynsymsort_symtype[];

 660 /*
 661  * Local functions.
 662  */
 663 extern char             *add_string(char *, char *);
 664 extern const char       *demangle(const char *);
 665 extern int              cap_names_match(Alist *, Alist *);

 667 extern void             lds_atexit(Ofl_desc *, int);

 669 extern void             libld_free(void *);
 670 extern void             *libld_malloc(size_t);
 671 extern void             *libld_realloc(void *, size_t);

 673 extern int              isdavl_compare(const void *, const void *);

 675 extern Sdf_desc         *sdf_add(const char *, APlist **);
 676 extern Sdf_desc         *sdf_find(const char *, APlist *);

 678 #if     defined(_ELF64)

 680 #define ld_add_actrel           ld64_add_actrel
 681 #define ld_add_libdir           ld64_add_libdir
 682 #define ld_adj_movereloc        ld64_adj_movereloc
 683 #define ld_am_I_partial         ld64_am_I_partial
 684 #define ld_ar_member            ld64_ar_member
 685 #define ld_ar_setup             ld64_ar_setup
 686 #define ld_assign_got_TLS       ld64_assign_got_TLS
 687 #define ld_bswap_Word           ld64_bswap_Word
 688 #define ld_bswap_Xword          ld64_bswap_Xword
 689 #define ld_cap_add_family       ld64_cap_add_family
 690 #define ld_cap_move_symtoobj    ld64_cap_move_symtoobj
 691 #define ld_comdat_validate      ld64_comdat_validate
 692 #define ld_disp_errmsg          ld64_disp_errmsg
 693 #define ld_ent_check            ld64_ent_check
 694 #define ld_ent_lookup           ld64_ent_lookup
 695 #define ld_eprintf              ld64_eprintf
 696 #define ld_exit                 ld64_exit
 697 #define ld_find_library         ld64_find_library
 698 #define ld_finish_libs          ld64_finish_libs
 699 #define ld_get_group            ld64_get_group
 700 #define ld_group_process        ld64_group_process
 701 #define ld_lib_setup            ld64_lib_setup
 702 #define ld_init_sighandler      ld64_init_sighandler
 703 #define ld_lcm                  ld64_lcm
 704 #define ld_make_bss             ld64_make_bss
```



```
 705 #define ld_make_data            ld64_make_data
 706 #define ld_make_got             ld64_make_got
 707 #define ld_make_parexpn_data    ld64_make_parexpn_data
 708 #define ld_make_sunwmove        ld64_make_sunmove
 709 #define ld_make_text            ld64_make_text
 710 #define ld_map_out              ld64_map_out
 711 #define ld_map_parse            ld64_map_parse
 712 #define ld_map_post_process     ld64_map_post_process
 713 #define ld_open_outfile         ld64_open_outfile
 714 #define ld_os_first_isdesc      ld64_os_first_isdesc
 715 #define ld_place_path_info_init ld64_place_path_info_init
 716 #define ld_place_section        ld64_place_section
 717 #define ld_process_archive      ld64_process_archive
 718 #define ld_process_files        ld64_process_files
 719 #define ld_process_flags        ld64_process_flags
 720 #define ld_process_ifl          ld64_process_ifl
 721 #define ld_process_move         ld64_process_move
 722 #define ld_process_open         ld64_process_open
 723 #define ld_process_ordered      ld64_process_ordered
 724 #define ld_process_sym_reloc    ld64_process_sym_reloc
 725 #define ld_reloc_enter          ld64_reloc_enter
 726 #define ld_reloc_GOT_relative   ld64_reloc_GOT_relative
 727 #define ld_reloc_plt            ld64_reloc_plt
 728 #define ld_reloc_remain_entry   ld64_reloc_remain_entry
 729 #define ld_reloc_set_aux_osdesc ld64_reloc_set_aux_osdesc
 730 #define ld_reloc_set_aux_usym   ld64_reloc_set_aux_usym
 731 #define ld_reloc_sym_name       ld64_reloc_sym_name
 732 #define ld_reloc_targval_get    ld64_reloc_targval_get
 733 #define ld_reloc_targval_set    ld64_reloc_targval_set
 734 #define ld_sec_validate         ld64_sec_validate
 735 #define ld_seg_lookup           ld64_seg_lookup
 736 #define ld_sort_ordered         ld64_sort_ordered
 737 #define ld_stt_section_sym_name ld64_stt_section_sym_name
 738 #define ld_sunw_ldmach          ld64_sunw_ldmach
 739 #define ld_sup_atexit           ld64_sup_atexit
 740 #define ld_sup_open             ld64_sup_open
 741 #define ld_sup_file             ld64_sup_file
 742 #define ld_sup_loadso           ld64_sup_loadso
 743 #define ld_sup_input_done       ld64_sup_input_done
 744 #define ld_sup_input_section    ld64_sup_input_section
 745 #define ld_sup_section          ld64_sup_section
 746 #define ld_sup_start            ld64_sup_start
 747 #define ld_swap_reloc_data      ld64_swap_reloc_data
 748 #define ld_sym_add_u            ld64_sym_add_u
 749 #define ld_sym_adjust_vis       ld64_sym_adjust_vis
 750 #define ld_sym_avl_comp         ld64_sym_avl_comp
 751 #define ld_sym_copy             ld64_sym_copy
 752 #define ld_sym_enter            ld64_sym_enter
 753 #define ld_sym_find             ld64_sym_find
 754 #define ld_sym_nodirect         ld64_sym_nodirect
 755 #define ld_sym_process          ld64_sym_process
 756 #define ld_sym_resolve          ld64_sym_resolve
 757 #define ld_sym_reducable        ld64_sym_reducable
 758 #endif /* ! codereview */
 759 #define ld_sym_spec             ld64_sym_spec
 760 #define ld_targ                 ld64_targ
 761 #define ld_targ_init_sparc      ld64_targ_init_sparc
 762 #define ld_targ_init_x86        ld64_targ_init_x86
 763 #define ld_unwind_make_hdr      ld64_unwind_make_hdr
 764 #define ld_unwind_populate_hdr  ld64_unwind_populate_hdr
 765 #define ld_unwind_register      ld64_unwind_register
 766 #define ld_vers_base            ld64_vers_base
 767 #define ld_vers_check_defs      ld64_vers_check_defs
 768 #define ld_vers_check_need      ld64_vers_check_need
 769 #define ld_vers_def_process     ld64_vers_def_process
 770 #define ld_vers_desc            ld64_vers_desc
```

```
771 #define ld_vers_find            ld64_vers_find
772 #define ld_vers_need_process    ld64_vers_need_process
773 #define ld_vers_promote         ld64_vers_promote
774 #define ld_vers_sym_process     ld64_vers_sym_process
775 #define ld_vers_verify          ld64_vers_verify
776 #define ld_wrap_enter           ld64_wrap_enter

778 #else

780 #define ld_add_actrel           ld32_add_actrel
781 #define ld_add_libdir           ld32_add_libdir
782 #define ld_adj_movereloc        ld32_adj_movereloc
783 #define ld_am_I_partial         ld32_am_I_partial
784 #define ld_ar_member            ld32_ar_member
785 #define ld_ar_setup             ld32_ar_setup
786 #define ld_assign_got_TLS       ld32_assign_got_TLS
787 #define ld_bswap_Word           ld32_bswap_Word
788 #define ld_bswap_Xword          ld32_bswap_Xword
789 #define ld_cap_add_family       ld32_cap_add_family
790 #define ld_cap_move_symtoobj    ld32_cap_move_symtoobj
791 #define ld_comdat_validate      ld32_comdat_validate
792 #define ld_disp_errmsg          ld32_disp_errmsg
793 #define ld_ent_check            ld32_ent_check
794 #define ld_ent_lookup           ld32_ent_lookup
795 #define ld_eprintf              ld32_eprintf
796 #define ld_exit                 ld32_exit
797 #define ld_find_library         ld32_find_library
798 #define ld_finish_libs          ld32_finish_libs
799 #define ld_get_group            ld32_get_group
800 #define ld_group_process        ld32_group_process
801 #define ld_lib_setup            ld32_lib_setup
802 #define ld_init_sighandler      ld32_init_sighandler
803 #define ld_lcm                  ld32_lcm
804 #define ld_make_bss             ld32_make_bss
805 #define ld_make_data            ld32_make_data
806 #define ld_make_got             ld32_make_got
807 #define ld_make_parexpn_data    ld32_make_parexpn_data
808 #define ld_make_sunwmove        ld32_make_sunmove
809 #define ld_make_text            ld32_make_text
810 #define ld_map_out              ld32_map_out
811 #define ld_map_parse            ld32_map_parse
812 #define ld_map_post_process     ld32_map_post_process
813 #define ld_open_outfile         ld32_open_outfile
814 #define ld_os_first_isdesc      ld32_os_first_isdesc
815 #define ld_place_path_info_init ld32_place_path_info_init
816 #define ld_place_section        ld32_place_section
817 #define ld_process_archive      ld32_process_archive
818 #define ld_process_files        ld32_process_files
819 #define ld_process_flags        ld32_process_flags
820 #define ld_process_ifl          ld32_process_ifl
821 #define ld_process_move         ld32_process_move
822 #define ld_process_open         ld32_process_open
823 #define ld_process_ordered      ld32_process_ordered
824 #define ld_process_sym_reloc    ld32_process_sym_reloc
825 #define ld_reloc_enter          ld32_reloc_enter
826 #define ld_reloc_GOT_relative   ld32_reloc_GOT_relative
827 #define ld_reloc_plt            ld32_reloc_plt
828 #define ld_reloc_remain_entry   ld32_reloc_remain_entry
829 #define ld_reloc_set_aux_osdesc ld32_reloc_set_aux_osdesc
830 #define ld_reloc_set_aux_usym   ld32_reloc_set_aux_usym
831 #define ld_reloc_sym_name       ld32_reloc_sym_name
832 #define ld_reloc_targval_get    ld32_reloc_targval_get
833 #define ld_reloc_targval_set    ld32_reloc_targval_set
834 #define ld_sec_validate         ld32_sec_validate
835 #define ld_seg_lookup           ld32_seg_lookup
836 #define ld_sort_ordered         ld32_sort_ordered
```



```
837 #define ld_stt_section_sym_name ld32_stt_section_sym_name
838 #define ld_sunw_ldmach          ld32_sunw_ldmach
839 #define ld_sup_atexit           ld32_sup_atexit
840 #define ld_sup_open             ld32_sup_open
841 #define ld_sup_file             ld32_sup_file
842 #define ld_sup_loadso           ld32_sup_loadso
843 #define ld_sup_input_done       ld32_sup_input_done
844 #define ld_sup_input_section    ld32_sup_input_section
845 #define ld_sup_section          ld32_sup_section
846 #define ld_sup_start            ld32_sup_start
847 #define ld_swap_reloc_data      ld32_swap_reloc_data
848 #define ld_sym_add_u            ld32_sym_add_u
849 #define ld_sym_adjust_vis       ld32_sym_adjust_vis
850 #define ld_sym_avl_comp         ld32_sym_avl_comp
851 #define ld_sym_copy             ld32_sym_copy
852 #define ld_sym_enter            ld32_sym_enter
853 #define ld_sym_find             ld32_sym_find
854 #define ld_sym_nodirect         ld32_sym_nodirect
855 #define ld_sym_process          ld32_sym_process
856 #define ld_sym_resolve          ld32_sym_resolve
857 #define ld_sym_reducable        ld32_sym_reducable
858 #endif /* ! codereview */
859 #define ld_sym_spec             ld32_sym_spec
860 #define ld_targ                 ld32_targ
861 #define ld_targ_init_sparc      ld32_targ_init_sparc
862 #define ld_targ_init_x86        ld32_targ_init_x86
863 #define ld_unwind_make_hdr      ld32_unwind_make_hdr
864 #define ld_unwind_populate_hdr  ld32_unwind_populate_hdr
865 #define ld_unwind_register      ld32_unwind_register
866 #define ld_vers_base            ld32_vers_base
867 #define ld_vers_check_defs      ld32_vers_check_defs
868 #define ld_vers_check_need      ld32_vers_check_need
869 #define ld_vers_def_process     ld32_vers_def_process
870 #define ld_vers_desc            ld32_vers_desc
871 #define ld_vers_find            ld32_vers_find
872 #define ld_vers_need_process    ld32_vers_need_process
873 #define ld_vers_promote         ld32_vers_promote
874 #define ld_vers_sym_process     ld32_vers_sym_process
875 #define ld_vers_verify          ld32_vers_verify
876 #define ld_wrap_enter           ld32_wrap_enter

878 #endif

880 extern void             dbg_cleanup(void);
881 extern int              dbg_setup(Ofl_desc *, const char *, int);

883 extern uintptr_t        ld_add_actrel(Word, Rel_desc *, Ofl_desc *);
884 extern uintptr_t        ld_add_libdir(Ofl_desc *, const char *);
885 extern void             ld_adj_movereloc(Ofl_desc *, Rel_desc *);
886 extern Sym_desc *       ld_am_I_partial(Rel_desc *, Xword);
887 extern void             ld_ar_member(Ar_desc *, Elf_Arsym *, Ar_aux *,
888                             Ar_mem *);
889 extern Ar_desc          *ld_ar_setup(const char *, Elf *, Ofl_desc *);
890 extern uintptr_t        ld_assign_got_TLS(Boolean, Rel_desc *, Ofl_desc *,
891                             Sym_desc *, Gotndx *, Gotref, Word, Word,
892                             Word, Word);

894 extern Word             ld_bswap_Word(Word);
895 extern Xword            ld_bswap_Xword(Xword);

897 extern uintptr_t        ld_cap_add_family(Ofl_desc *, Sym_desc *, Sym_desc *,
898                             Cap_group *, APlist **);
899 extern void             ld_cap_move_symtoobj(Ofl_desc *);

901 extern void             ld_comdat_validate(Ofl_desc *, Ifl_desc *);
```

```
903 extern void             ld_disp_errmsg(const char *, Rel_desc *, Ofl_desc *);

905 extern void             ld_ent_check(Ofl_desc *);
906 extern Ent_desc         *ld_ent_lookup(Ofl_desc *, const char *name,
907                             avl_index_t *where);
908 extern void             ld_eprintf(Ofl_desc *, Error, const char *, ...);
909 extern int              ld_exit(Ofl_desc *);

911 extern uintptr_t        ld_find_library(const char *, Ofl_desc *);
912 extern uintptr_t        ld_finish_libs(Ofl_desc *);

914 extern const char       *ld_stt_section_sym_name(Is_desc *);

916 extern Group_desc       *ld_get_group(Ofl_desc *, Is_desc *);
917 extern uintptr_t        ld_group_process(Is_desc *, Ofl_desc *);

919 extern uintptr_t        ld_lib_setup(Ofl_desc *);

921 extern void             ld_init_sighandler(Ofl_desc *);

923 extern Xword            ld_lcm(Xword, Xword);

925 extern uintptr_t        ld_make_bss(Ofl_desc *, Xword, Xword, uint_t);
926 extern Is_desc          *ld_make_data(Ofl_desc *, size_t);
927 extern uintptr_t        ld_make_got(Ofl_desc *);
928 extern uintptr_t        ld_make_parexpn_data(Ofl_desc *, size_t, Xword);
929 extern uintptr_t        ld_make_sunwmove(Ofl_desc *, int);
930 extern Is_desc          *ld_make_text(Ofl_desc *, size_t);
931 extern void             ld_map_out(Ofl_desc *);
932 extern Boolean          ld_map_parse(const char *, Ofl_desc *);
933 extern Boolean          ld_map_post_process(Ofl_desc *);

935 extern uintptr_t        ld_open_outfile(Ofl_desc *);

937 extern Is_desc          *ld_os_first_isdesc(Os_desc *);
938 extern Place_path_info  *ld_place_path_info_init(Ofl_desc *, Ifl_desc *,
939                             Place_path_info *);
940 extern Os_desc          *ld_place_section(Ofl_desc *, Is_desc *,
941                             Place_path_info *path_info,  int, const char *);
942 extern Boolean          ld_process_archive(const char *, int, Ar_desc *,
943                             Ofl_desc *);
944 extern uintptr_t        ld_process_files(Ofl_desc *, int, char **);
945 extern uintptr_t        ld_process_flags(Ofl_desc *, int, char **);
946 extern uintptr_t        ld_process_ifl(const char *, const char *, int, Elf *,
947                             Word, Ofl_desc *, Rej_desc *, Ifl_desc **);
948 extern uintptr_t        ld_process_move(Ofl_desc *);
949 extern uintptr_t        ld_process_open(const char *, const char *, int *,
950                             Ofl_desc *, Word, Rej_desc *, Ifl_desc **);
951 extern uintptr_t        ld_process_ordered(Ofl_desc *, Ifl_desc *,
952                             Place_path_info *path_info,  Word);
953 extern uintptr_t        ld_process_sym_reloc(Ofl_desc *, Rel_desc *, Rel *,
954                             Is_desc *, const char *, Word);

956 extern Rel_desc         *ld_reloc_enter(Ofl_desc *, Rel_cache *, Rel_desc *,
957                             Word);
958 extern uintptr_t        ld_reloc_GOT_relative(Boolean, Rel_desc *, Ofl_desc *);
959 extern uintptr_t        ld_reloc_plt(Rel_desc *, Ofl_desc *);
960 extern void             ld_reloc_remain_entry(Rel_desc *, Os_desc *,
961                             Ofl_desc *, Boolean *);
962 extern Boolean          ld_reloc_set_aux_osdesc(Ofl_desc *, Rel_desc *,
963                             Os_desc *);
964 extern Boolean          ld_reloc_set_aux_usym(Ofl_desc *, Rel_desc *,
965                             Sym_desc *);

967 extern const char       *ld_reloc_sym_name(Rel_desc *);
968 extern int              ld_reloc_targval_get(Ofl_desc *, Rel_desc *,
```



```
969                             uchar_t *, Xword *);
970 extern int              ld_reloc_targval_set(Ofl_desc *, Rel_desc *,
971                             uchar_t *, Xword);

973 extern Sg_desc          *ld_seg_lookup(Ofl_desc *, const char *,
974                             avl_index_t *where);
975 extern void             ld_sec_validate(Ofl_desc *);
976 extern uintptr_t        ld_sort_ordered(Ofl_desc *);
977 extern Half             ld_sunw_ldmach();
978 extern void             ld_sup_atexit(Ofl_desc *, int);
979 extern void             ld_sup_open(Ofl_desc *, const char **, const char **,
980                             int *, int, Elf **, Elf *ref, size_t,
981                             const Elf_Kind);
982 extern void             ld_sup_file(Ofl_desc *, const char *, const Elf_Kind,
983                             int flags, Elf *);
984 extern uintptr_t        ld_sup_loadso(Ofl_desc *, const char *);
985 extern void             ld_sup_input_done(Ofl_desc *);
986 extern void             ld_sup_section(Ofl_desc *, const char *, Shdr *, Word,
987                             Elf_Data *, Elf *);
988 extern uintptr_t        ld_sup_input_section(Ofl_desc*, Ifl_desc *,
989                             const char *, Shdr **, Word, Elf_Scn *, Elf *);
990 extern void             ld_sup_start(Ofl_desc *, const Half, const char *);
991 extern int              ld_swap_reloc_data(Ofl_desc *, Rel_desc *);
992 extern Sym_desc         *ld_sym_add_u(const char *, Ofl_desc *, Msg);
993 extern void             ld_sym_adjust_vis(Sym_desc *, Ofl_desc *);
994 extern int              ld_sym_avl_comp(const void *, const void *);
995 extern uintptr_t        ld_sym_copy(Sym_desc *);
996 extern Sym_desc         *ld_sym_enter(const char *, Sym *, Word, Ifl_desc *,
997                             Ofl_desc *, Word, Word, sd_flag_t, avl_index_t *);
998 extern Sym_desc         *ld_sym_find(const char *, Word, avl_index_t *,
999                             Ofl_desc *);
1000 extern uintptr_t        ld_sym_nodirect(Is_desc *, Ifl_desc *, Ofl_desc *);
1001 extern uintptr_t        ld_sym_process(Is_desc *, Ifl_desc *, Ofl_desc *);
1002 extern uintptr_t        ld_sym_resolve(Sym_desc *, Sym *, Ifl_desc *,
1003                             Ofl_desc *, int, Word, sd_flag_t);
1004 extern Boolean          ld_sym_reducable(Ofl_desc *, Sym_desc *);
1005 #endif /* ! codereview */
1006 extern uintptr_t        ld_sym_spec(Ofl_desc *);

1008 extern Target           ld_targ;
1009 extern const Target     *ld_targ_init_sparc(void);
1010 extern const Target     *ld_targ_init_x86(void);

1012 extern uintptr_t        ld_unwind_make_hdr(Ofl_desc *);
1013 extern uintptr_t        ld_unwind_populate_hdr(Ofl_desc *);
1014 extern uintptr_t        ld_unwind_register(Os_desc *, Ofl_desc *);

1016 extern Ver_desc         *ld_vers_base(Ofl_desc *);
1017 extern uintptr_t        ld_vers_check_defs(Ofl_desc *);
1018 extern uintptr_t        ld_vers_check_need(Ofl_desc *);
1019 extern uintptr_t        ld_vers_def_process(Is_desc *, Ifl_desc *, Ofl_desc *);
1020 extern Ver_desc         *ld_vers_desc(const char *, Word, APlist **);
1021 extern Ver_desc         *ld_vers_find(const char *, Word, APlist *);
1022 extern uintptr_t        ld_vers_need_process(Is_desc *, Ifl_desc *, Ofl_desc *);
1023 extern void             ld_vers_promote(Sym_desc *, Word, Ifl_desc *,
1024                             Ofl_desc *);
1025 extern int              ld_vers_sym_process(Ofl_desc *, Is_desc *, Ifl_desc *);
1026 extern int              ld_vers_verify(Ofl_desc *);
1027 extern WrapSymNode      *ld_wrap_enter(Ofl_desc *, const char *);

1029 extern uintptr_t        add_regsym(Sym_desc *, Ofl_desc *);
1030 extern Word             hashbkts(Word);
1031 extern Xword            lcm(Xword, Xword);

1033 /*
1034  * Most platforms have both a 32 and 64-bit variant (e.g. EM_SPARC and
```

```
1035  * EM_SPARCV9). To support this, there many files in libld that are built
1036  * twice, once for ELFCLASS64 (_ELF64), and once for ELFCLASS32. In these
1037  * files, we sometimes want to supply one value for the ELFCLASS32 case
1038  * and another for ELFCLASS64. The LD_TARG_BYCLASS macro is used to do
1039  * this. It is called with both both alternatives, and yields the one
1040  * that applies to the current compilation environment.
1041  */
1042 #ifdef  _ELF64
1043 #define LD_TARG_BYCLASS(_ec32, _ec64) (_ec64)
1044 #else
1045 #define LD_TARG_BYCLASS(_ec32, _ec64) (_ec32)
1046 #endif


1049 #ifdef  __cplusplus
1050 }
1051 #endif

1053 #endif /* _LIBLD_DOT_H */
```

```
**********************************************************
   95992 Thu Feb  7 01:35:55 2019
new/usr/src/cmd/sgs/libld/common/syms.c
10346 ld(1) should not reduce symbol visibility of COMDAT symbols when producing
**********************************************************
______unchanged_portion_omitted_

 952 /*
 953  * Determine a potential capability symbol's visibility.
 954  *
 955  * The -z symbolcap option transforms an object capabilities relocatable object
 956  * into a symbol capabilities relocatable object.  Any global function symbols,
 957  * or initialized global data symbols are candidates for transforming into local
 958  * symbol capabilities definitions.  However, if a user indicates that a symbol
 959  * should be demoted to local using a mapfile, then there is no need to
 960  * transform the associated global symbol.
 961  *
 962  * Normally, a symbol's visibility is determined after the symbol resolution
 963  * process, after all symbol state has been gathered and resolved.  However,
 964  * for -z symbolcap, this determination is too late.  When a global symbol is
 965  * read from an input file we need to determine it's visibility so as to decide
 966  * whether to create a local or not.
 967  *
 968  * If a user has explicitly defined this symbol as having local scope within a
 969  * mapfile, then a symbol of the same name already exists.  However, explicit
 970  * local definitions are uncommon, as most mapfiles define the global symbol
 971  * requirements together with an auto-reduction directive '*'.  If this state
 972  * has been defined, then we must make sure that the new symbol isn't a type
 973  * that can not be demoted to local.
 974  */
 975 static int
 976 sym_cap_vis(const char *name, Word hash, Sym *sym, Ofl_desc *ofl)
 977 {
 978         Sym_desc        *sdp;
 979         uchar_t         vis;
 980         avl_index_t     where;
 981         sd_flag_t       sdflags = 0;

 983         /*
 984          * Determine the visibility of the new symbol.
 985          */
 986         vis = ELF_ST_VISIBILITY(sym->st_other);
 987         switch (vis) {
 988         case STV_EXPORTED:
 989                 sdflags |= FLG_SY_EXPORT;
 990                 break;
 991         case STV_SINGLETON:
 992                 sdflags |= FLG_SY_SINGLE;
 993                 break;
 994         case STV_HIDDEN:
 995                 sdflags |= FLG_SY_HIDDEN;
 996                 break;
 997 #endif /* ! codereview */
 998         }

1000         /*
1001          * Determine whether a symbol definition already exists, and if so
1002          * obtain the visibility.
1003          */
1004         if ((sdp = ld_sym_find(name, hash, &where, ofl)) != NULL)
1005                 sdflags |= sdp->sd_flags;

1007         /*
1008          * Determine whether the symbol flags indicate this symbol should be
1009          * hidden.
1010          */
```



```
1011         if ((ofl->ofl_flags & (FLG_OF_AUTOLCL | FLG_OF_AUTOELM)) &&
1012             ((sdflags & MSK_SY_NOAUTO) == 0))
1013                 sdflags |= FLG_SY_HIDDEN;

1015         return ((sdflags & FLG_SY_HIDDEN) == 0);
1016 }

1018 /*
1019  * This routine checks to see if a symbols visibility needs to be reduced to
1020  * either SYMBOLIC or LOCAL.  This routine can be called from either
1021  * reloc_init() or sym_validate().
1022  */
1023 void
1024 ld_sym_adjust_vis(Sym_desc *sdp, Ofl_desc *ofl)
1025 {
1026         ofl_flag_t      oflags = ofl->ofl_flags;
1027         Sym             *sym = sdp->sd_sym;

1029         if ((sdp->sd_ref == REF_REL_NEED) &&
1030             (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
1031                 /*
1032                  * If auto-reduction/elimination is enabled, reduce any
1033                  * non-versioned, and non-local capabilities global symbols.
1034                  * A symbol is a candidate for auto-reduction/elimination if:
1035                  *
1036                  *  -   the symbol wasn't explicitly defined within a mapfile
1037                  *      (in which case all the necessary state has been applied
1038                  *      to the symbol), or
1039                  *  -   the symbol isn't one of the family of reserved
1040                  *      special symbols (ie. _end, _etext, etc.), or
1041                  *  -   the symbol isn't a SINGLETON, or
1042                  *  -   the symbol wasn't explicitly defined within a version
1043                  *      definition associated with an input relocatable object.
1044                  *
1045                  * Indicate that the symbol has been reduced as it may be
1046                  * necessary to print these symbols later.
1047                  */
1048                 if ((oflags & (FLG_OF_AUTOLCL | FLG_OF_AUTOELM)) &&
1049                     ((sdp->sd_flags & MSK_SY_NOAUTO) == 0)) {
1050                         if ((sdp->sd_flags & FLG_SY_HIDDEN) == 0) {
1051                                 sdp->sd_flags |=
1052                                     (FLG_SY_REDUCED | FLG_SY_HIDDEN);
1053                         }

1055                         if (oflags & (FLG_OF_REDLSYM | FLG_OF_AUTOELM)) {
1056                                 sdp->sd_flags |= FLG_SY_ELIM;
1057                                 sym->st_other = STV_ELIMINATE |
1058                                     (sym->st_other & ~MSK_SYM_VISIBILITY);
1059                         } else if (ELF_ST_VISIBILITY(sym->st_other) !=
1060                             STV_INTERNAL)
1061                                 sym->st_other = STV_HIDDEN |
1062                                     (sym->st_other & ~MSK_SYM_VISIBILITY);
1063                 }

1065                 /*
1066                  * If -Bsymbolic is in effect, and the symbol hasn't explicitly
1067                  * been defined nodirect (via a mapfile), then bind the global
1068                  * symbol symbolically and assign the STV_PROTECTED visibility
1069                  * attribute.
1070                  */
1071                 if ((oflags & FLG_OF_SYMBOLIC) &&
1072                     ((sdp->sd_flags & (FLG_SY_HIDDEN | FLG_SY_NDIR)) == 0)) {
1073                         sdp->sd_flags |= FLG_SY_PROTECT;
1074                         if (ELF_ST_VISIBILITY(sym->st_other) == STV_DEFAULT)
1075                                 sym->st_other = STV_PROTECTED |
1076                                     (sym->st_other & ~MSK_SYM_VISIBILITY);
```

```
1077                 }
1078         }

1080         /*
1081          * Indicate that this symbol has had it's visibility checked so that
1082          * we don't need to do this investigation again.
1083          */
1084         sdp->sd_flags |= FLG_SY_VISIBLE;
1085 }

1087 /*
1088  * Make sure a symbol definition is local to the object being built.
1089  */
1090 inline static int
1091 ensure_sym_local(Ofl_desc *ofl, Sym_desc *sdp, const char *str)
1092 {
1093         if (sdp->sd_sym->st_shndx == SHN_UNDEF) {
1094                 if (str) {
1095                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_SYM_UNDEF),
1096                             str, demangle((char *)sdp->sd_name));
1097                 }
1098                 return (1);
1099         }
1100         if (sdp->sd_ref != REF_REL_NEED) {
1101                 if (str) {
1102                         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_SYM_EXTERN),
1103                             str, demangle((char *)sdp->sd_name),
1104                             sdp->sd_file->ifl_name);
1105                 }
1106                 return (1);
1107         }

1109         sdp->sd_flags |= FLG_SY_UPREQD;
1110         if (sdp->sd_isc) {
1111                 sdp->sd_isc->is_flags |= FLG_IS_SECTREF;
1112                 sdp->sd_isc->is_file->ifl_flags |= FLG_IF_FILEREF;
1113         }
1114         return (0);
1115 }

1117 /*
1118  * Make sure all the symbol definitions required for initarray, finiarray, or
1119  * preinitarray's are local to the object being built.
1120  */
1121 static int
1122 ensure_array_local(Ofl_desc *ofl, APlist *apl, const char *str)
1123 {
1124         Aliste          idx;
1125         Sym_desc        *sdp;
1126         int             ret = 0;

1128         for (APLIST_TRAVERSE(apl, idx, sdp))
1129                 ret += ensure_sym_local(ofl, sdp, str);

1131         return (ret);
1132 }

1134 /*
1135  * After all symbol table input processing has been finished, and all relocation
1136  * counting has been carried out (ie. no more symbols will be read, generated,
1137  * or modified), validate and count the relevant entries:
1138  *
1139  *  -   check and print any undefined symbols remaining.  Note that if a symbol
1140  *      has been defined by virtue of the inclusion of  an implicit shared
1141  *      library, it is still classed as undefined.
1142  *
```



```
1143  *  -   count the number of global needed symbols together with the size of
1144  *      their associated name strings (if scoping has been indicated these
1145  *      symbols may be reduced to locals).
1146  *
1147  *  -   establish the size and alignment requirements for the global .bss
1148  *      section (the alignment of this section is based on the  first symbol
1149  *      that it will contain).
1150  */
1151 uintptr_t
1152 ld_sym_validate(Ofl_desc *ofl)
1153 {
1154         Sym_avlnode     *sav;
1155         Sym_desc        *sdp;
1156         Sym             *sym;
1157         ofl_flag_t      oflags = ofl->ofl_flags;
1158         ofl_flag_t      undef = 0, needed = 0, verdesc = 0;
1159         Xword           bssalign = 0, tlsalign = 0;
1160         Boolean         need_bss, need_tlsbss;
1161         Xword           bsssize = 0, tlssize = 0;
1162 #if     defined(_ELF64)
1163         Xword           lbssalign = 0, lbsssize = 0;
1164         Boolean         need_lbss;
1165 #endif
1166         int             ret, allow_ldynsym;
1167         uchar_t         type;
1168         ofl_flag_t      undef_state = 0;

1170         DBG_CALL(Dbg_basic_validate(ofl->ofl_lml));

1172         /*
1173          * The need_XXX booleans are used to determine whether we need to
1174          * create each type of bss section. We used to create these sections
1175          * if the sum of the required sizes for each type were non-zero.
1176          * However, it is possible for a compiler to generate COMMON variables
1177          * of zero-length and this tricks that logic --- even zero-length
1178          * symbols need an output section.
1179          */
1180         need_bss = need_tlsbss = FALSE;
1181 #if     defined(_ELF64)
1182         need_lbss = FALSE;
1183 #endif

1185         /*
1186          * Determine how undefined symbols are handled:
1187          *
1188          * fatal:
1189          *      If this link-edit calls for no undefined symbols to remain
1190          *      (this is the default case when generating an executable but
1191          *      can be enforced for any object using -z defs), a fatal error
1192          *      condition will be indicated.
1193          *
1194          * warning:
1195          *      If we're creating a shared object, and either the -Bsymbolic
1196          *      flag is set, or the user has turned on the -z guidance feature,
1197          *      then a non-fatal warning is issued for each symbol.
1198          *
1199          * ignore:
1200          *      In all other cases, undefined symbols are quietly allowed.
1201          */
1202         if (oflags & FLG_OF_NOUNDEF) {
1203                 undef = FLG_OF_FATAL;
1204         } else if (oflags & FLG_OF_SHAROBJ) {
1205                 if ((oflags & FLG_OF_SYMBOLIC) ||
1206                     OFL_GUIDANCE(ofl, FLG_OFG_NO_DEFS))
1207                         undef = FLG_OF_WARN;
1208         }
```

```
1210         /*
1211          * If the symbol is referenced from an implicitly included shared object
1212          * (ie. it's not on the NEEDED list) then the symbol is also classified
1213          * as undefined and a fatal error condition will be indicated.
1214          */
1215         if ((oflags & FLG_OF_NOUNDEF) || !(oflags & FLG_OF_SHAROBJ))
1216                 needed = FLG_OF_FATAL;
1217         else if ((oflags & FLG_OF_SHAROBJ) &&
1218             OFL_GUIDANCE(ofl, FLG_OFG_NO_DEFS))
1219                 needed = FLG_OF_WARN;

1221         /*
1222          * If the output image is being versioned, then all symbol definitions
1223          * must be associated with a version.  Any symbol that isn't associated
1224          * with a version is classified as undefined, and a fatal error
1225          * condition is indicated.
1226          */
1227         if ((oflags & FLG_OF_VERDEF) && (ofl->ofl_vercnt > VER_NDX_GLOBAL))
1228                 verdesc = FLG_OF_FATAL;

1230         allow_ldynsym = OFL_ALLOW_LDYNSYM(ofl);

1232         if (allow_ldynsym) {
1233                 /*
1234                  * Normally, we disallow symbols with 0 size from appearing
1235                  * in a dyn[sym|tls]sort section. However, there are some
1236                  * symbols that serve special purposes that we want to exempt
1237                  * from this rule. Look them up, and set their
1238                  * FLG_SY_DYNSORT flag.
1239                  */
1240                 static const char *special[] = {
1241                         MSG_ORIG(MSG_SYM_INIT_U),       /* _init */
1242                         MSG_ORIG(MSG_SYM_FINI_U),       /* _fini */
1243                         MSG_ORIG(MSG_SYM_START),        /* _start */
1244                         NULL
1245                 };
1246                 int i;

1248                 for (i = 0; special[i] != NULL; i++) {
1249                         if (((sdp = ld_sym_find(special[i],
1250                             SYM_NOHASH, NULL, ofl)) != NULL) &&
1251                             (sdp->sd_sym->st_size == 0)) {
1252                                 if (ld_sym_copy(sdp) == S_ERROR)
1253                                         return (S_ERROR);
1254                                 sdp->sd_flags |= FLG_SY_DYNSORT;
1255                         }
1256                 }
1257         }

1259         /*
1260          * Collect and validate the globals from the internal symbol table.
1261          */
1262         for (sav = avl_first(&ofl->ofl_symavl); sav;
1263             sav = AVL_NEXT(&ofl->ofl_symavl, sav)) {
1264                 Is_desc         *isp;
1265                 int             undeferr = 0;
1266                 uchar_t         vis;

1268                 sdp = sav->sav_sdp;

1270                 /*
1271                  * If undefined symbols are allowed, and we're not being
1272                  * asked to supply guidance, ignore any symbols that are
1273                  * not needed.
1274                  */
```



```
1275                 if (!(oflags & FLG_OF_NOUNDEF) &&
1276                     !OFL_GUIDANCE(ofl, FLG_OFG_NO_DEFS) &&
1277                     (sdp->sd_ref == REF_DYN_SEEN))
1278                         continue;

1280                 /*
1281                  * If the symbol originates from an external or parent mapfile
1282                  * reference and hasn't been matched to a reference from a
1283                  * relocatable object, ignore it.
1284                  */
1285                 if ((sdp->sd_flags & (FLG_SY_EXTERN | FLG_SY_PARENT)) &&
1286                     ((sdp->sd_flags & FLG_SY_MAPUSED) == 0)) {
1287                         sdp->sd_flags |= FLG_SY_INVALID;
1288                         continue;
1289                 }

1291                 sym = sdp->sd_sym;
1292                 type = ELF_ST_TYPE(sym->st_info);

1294                 /*
1295                  * Sanity check TLS.
1296                  */
1297                 if ((type == STT_TLS) && (sym->st_size != 0) &&
1298                     (sym->st_shndx != SHN_UNDEF) &&
1299                     (sym->st_shndx != SHN_COMMON)) {
1300                         Is_desc         *isp = sdp->sd_isc;
1301                         Ifl_desc        *ifl = sdp->sd_file;

1303                         if ((isp == NULL) || (isp->is_shdr == NULL) ||
1304                             ((isp->is_shdr->sh_flags & SHF_TLS) == 0)) {
1305                                 ld_eprintf(ofl, ERR_FATAL,
1306                                     MSG_INTL(MSG_SYM_TLS),
1307                                     demangle(sdp->sd_name), ifl->ifl_name);
1308                                 continue;
1309                         }
1310                 }

1312                 if ((sdp->sd_flags & FLG_SY_VISIBLE) == 0)
1313                         ld_sym_adjust_vis(sdp, ofl);

1315                 if ((sdp->sd_flags & FLG_SY_REDUCED) &&
1316                     (oflags & FLG_OF_PROCRED)) {
1317                         DBG_CALL(Dbg_syms_reduce(ofl, DBG_SYM_REDUCE_GLOBAL,
1318                             sdp, 0, 0));
1319                 }

1321                 /*
1322                  * Record any STV_SINGLETON existence.
1323                  */
1324                 if ((vis = ELF_ST_VISIBILITY(sym->st_other)) == STV_SINGLETON)
1325                         ofl->ofl_dtflags_1 |= DF_1_SINGLETON;

1327                 /*
1328                  * If building a shared object or executable, and this is a
1329                  * non-weak UNDEF symbol with reduced visibility (STV_*), then
1330                  * give a fatal error.
1331                  */
1332                 if (((oflags & FLG_OF_RELOBJ) == 0) &&
1333                     (sym->st_shndx == SHN_UNDEF) &&
1334                     (ELF_ST_BIND(sym->st_info) != STB_WEAK)) {
1335                         if (vis && (vis != STV_SINGLETON)) {
1336                                 sym_undef_entry(ofl, sdp, BNDLOCAL,
1337                                     FLG_OF_FATAL, &undef_state);
1338                                 continue;
1339                         }
1340                 }
```

```
1342                /*
1343                 * If this symbol is defined in a non-allocatable section,
1344                 * reduce it to local symbol.
1345                 */
1346                if (((isp = sdp->sd_isc) != 0) && isp->is_shdr &&
1347                    ((isp->is_shdr->sh_flags & SHF_ALLOC) == 0)) {
1348                        sdp->sd_flags |= (FLG_SY_REDUCED | FLG_SY_HIDDEN);
1349                }

1351                /*
1352                 * If this symbol originated as a SHN_SUNW_IGNORE, it will have
1353                 * been processed as an SHN_UNDEF.  Return the symbol to its
1354                 * original index for validation, and propagation to the output
1355                 * file.
1356                 */
1357                if (sdp->sd_flags & FLG_SY_IGNORE)
1358                        sdp->sd_shndx = SHN_SUNW_IGNORE;

1360                if (undef) {
1361                        /*
1362                         * If a non-weak reference remains undefined, or if a
1363                         * mapfile reference is not bound to the relocatable
1364                         * objects that make up the object being built, we have
1365                         * a fatal error.
1366                         *
1367                         * The exceptions are symbols which are defined to be
1368                         * found in the parent (FLG_SY_PARENT), which is really
1369                         * only meaningful for direct binding, or are defined
1370                         * external (FLG_SY_EXTERN) so as to suppress -zdefs
1371                         * errors.
1372                         *
1373                         * Register symbols are always allowed to be UNDEF.
1374                         *
1375                         * Note that we don't include references created via -u
1376                         * in the same shared object binding test.  This is for
1377                         * backward compatibility, in that a number of archive
1378                         * makefile rules used -u to cause archive extraction.
1379                         * These same rules have been cut and pasted to apply
1380                         * to shared objects, and thus although the -u reference
1381                         * is redundant, flagging it as fatal could cause some
1382                         * build to fail.  Also we have documented the use of
1383                         * -u as a mechanism to cause binding to weak version
1384                         * definitions, thus giving users an error condition
1385                         * would be incorrect.
1386                         */
1387                        if (!(sdp->sd_flags & FLG_SY_REGSYM) &&
1388                            ((sym->st_shndx == SHN_UNDEF) &&
1389                            ((ELF_ST_BIND(sym->st_info) != STB_WEAK) &&
1390                            ((sdp->sd_flags &
1391                            (FLG_SY_PARENT | FLG_SY_EXTERN)) == 0)) ||
1392                            ((sdp->sd_flags &
1393                            (FLG_SY_MAPREF | FLG_SY_MAPUSED | FLG_SY_HIDDEN |
1394                            FLG_SY_PROTECT)) == FLG_SY_MAPREF))) {
1395                                sym_undef_entry(ofl, sdp, UNDEF, undef,
1396                                    &undef_state);
1397                                undeferr = 1;
1398                        }

1400                } else {
1401                        /*
1402                         * For building things like shared objects (or anything
1403                         * -znodefs), undefined symbols are allowed.
1404                         *
1405                         * If a mapfile reference remains undefined the user
1406                         * would probably like a warning at least (they've
```



```
1407                         * usually mis-spelt the reference).  Refer to the above
1408                         * comments for discussion on -u references, which
1409                         * are not tested for in the same manner.
1410                         */
1411                        if ((sdp->sd_flags &
1412                            (FLG_SY_MAPREF | FLG_SY_MAPUSED)) ==
1413                            FLG_SY_MAPREF) {
1414                                sym_undef_entry(ofl, sdp, UNDEF, FLG_OF_WARN,
1415                                    &undef_state);
1416                                undeferr = 1;
1417                        }
1418                }

1420                /*
1421                 * If this symbol comes from a dependency mark the dependency
1422                 * as required (-z ignore can result in unused dependencies
1423                 * being dropped).  If we need to record dependency versioning
1424                 * information indicate what version of the needed shared object
1425                 * this symbol is part of.  Flag the symbol as undefined if it
1426                 * has not been made available to us.
1427                 */
1428                if ((sdp->sd_ref == REF_DYN_NEED) &&
1429                    (!(sdp->sd_flags & FLG_SY_REFRSD))) {
1430                        sdp->sd_file->ifl_flags |= FLG_IF_DEPREQD;

1432                        /*
1433                         * Capture that we've bound to a symbol that doesn't
1434                         * allow being directly bound to.
1435                         */
1436                        if (sdp->sd_flags & FLG_SY_NDIR)
1437                                ofl->ofl_flags1 |= FLG_OF1_NGLBDIR;

1439                        if (sdp->sd_file->ifl_vercnt) {
1440                                int             vndx;
1441                                Ver_index       *vip;

1443                                vndx = sdp->sd_aux->sa_dverndx;
1444                                vip = &sdp->sd_file->ifl_verndx[vndx];
1445                                if (vip->vi_flags & FLG_VER_AVAIL) {
1446                                        vip->vi_flags |= FLG_VER_REFER;
1447                                } else {
1448                                        sym_undef_entry(ofl, sdp, NOTAVAIL,
1449                                            FLG_OF_FATAL, &undef_state);
1450                                        continue;
1451                                }
1452                        }
1453                }

1455                /*
1456                 * Test that we do not bind to symbol supplied from an implicit
1457                 * shared object.  If a binding is from a weak reference it can
1458                 * be ignored.
1459                 */
1460                if (needed && !undeferr && (sdp->sd_flags & FLG_SY_GLOBREF) &&
1461                    (sdp->sd_ref == REF_DYN_NEED) &&
1462                    (sdp->sd_flags & FLG_SY_NOTAVAIL)) {
1463                        sym_undef_entry(ofl, sdp, IMPLICIT, needed,
1464                            &undef_state);
1465                        if (needed == FLG_OF_FATAL)
1466                                continue;
1467                }

1469                /*
1470                 * Test that a symbol isn't going to be reduced to local scope
1471                 * which actually wants to bind to a shared object - if so it's
1472                 * a fatal error.
```

```
1473                 */
1474                if ((sdp->sd_ref == REF_DYN_NEED) &&
1475                    (sdp->sd_flags & (FLG_SY_HIDDEN | FLG_SY_PROTECT))) {
1476                        sym_undef_entry(ofl, sdp, BNDLOCAL, FLG_OF_FATAL,
1477                            &undef_state);
1478                        continue;
1479                }

1481                /*
1482                 * If the output image is to be versioned then all symbol
1483                 * definitions must be associated with a version.  Remove any
1484                 * versioning that might be left associated with an undefined
1485                 * symbol.
1486                 */
1487                if (verdesc && (sdp->sd_ref == REF_REL_NEED)) {
1488                        if (sym->st_shndx == SHN_UNDEF) {
1489                                if (sdp->sd_aux && sdp->sd_aux->sa_overndx)
1490                                        sdp->sd_aux->sa_overndx = 0;
1491                        } else {
1492                                if (!SYM_IS_HIDDEN(sdp) && sdp->sd_aux &&
1493                                    (sdp->sd_aux->sa_overndx == 0)) {
1494                                        sym_undef_entry(ofl, sdp, NOVERSION,
1495                                            verdesc, &undef_state);
1496                                        continue;
1497                                }
1498                        }
1499                }

1501                /*
1502                 * If we don't need the symbol there's no need to process it
1503                 * any further.
1504                 */
1505                if (sdp->sd_ref == REF_DYN_SEEN)
1506                        continue;

1508                /*
1509                 * Calculate the size and alignment requirements for the global
1510                 * .bss and .tls sections.  If we're building a relocatable
1511                 * object only account for scoped COMMON symbols (these will
1512                 * be converted to .bss references).
1513                 *
1514                 * When -z nopartial is in effect, partially initialized
1515                 * symbols are directed to the special .data section
1516                 * created for that purpose (ofl->ofl_isparexpn).
1517                 * Otherwise, partially initialized symbols go to .bss.
1518                 *
1519                 * Also refer to make_mvsections() in sunwmove.c
1520                 */
1521                if ((sym->st_shndx == SHN_COMMON) &&
1522                    (((oflags & FLG_OF_RELOBJ) == 0) ||
1523                    ld_sym_reducable(ofl, sdp))) {
 994                    (SYM_IS_HIDDEN(sdp) && (oflags & FLG_OF_PROCRED)))) {
1524                        if ((sdp->sd_move == NULL) ||
1525                            ((sdp->sd_flags & FLG_SY_PAREXPN) == 0)) {
1526                                if (type != STT_TLS) {
1527                                        need_bss = TRUE;
1528                                        bsssize = (Xword)S_ROUND(bsssize,
1529                                            sym->st_value) + sym->st_size;
1530                                        if (sym->st_value > bssalign)
1531                                                bssalign = sym->st_value;
1532                                } else {
1533                                        need_tlsbss = TRUE;
1534                                        tlssize = (Xword)S_ROUND(tlssize,
1535                                            sym->st_value) + sym->st_size;
1536                                        if (sym->st_value > tlsalign)
1537                                                tlsalign = sym->st_value;
```



```
1538                                    }
1539                            }
1540                    }

1542 #if     defined(_ELF64)
1543                    /*
1544                     * Calculate the size and alignment requirement for the global
1545                     * .lbss. TLS or partially initialized symbols do not need to be
1546                     * considered yet.
1547                     */
1548                    if ((ld_targ.t_m.m_mach == EM_AMD64) &&
1549                        (sym->st_shndx == SHN_X86_64_LCOMMON)) {
1550                            need_lbss = TRUE;
1551                            lbsssize = (Xword)S_ROUND(lbsssize, sym->st_value) +
1552                                sym->st_size;
1553                            if (sym->st_value > lbssalign)
1554                                    lbssalign = sym->st_value;
1555                    }
1556 #endif
1557                    /*
1558                     * If a symbol was referenced via the command line
1559                     * (ld -u <>, ...), then this counts as a reference against the
1560                     * symbol. Mark any section that symbol is defined in.
1561                     */
1562                    if (((isp = sdp->sd_isc) != 0) &&
1563                        (sdp->sd_flags & FLG_SY_CMDREF)) {
1564                            isp->is_flags |= FLG_IS_SECTREF;
1565                            isp->is_file->ifl_flags |= FLG_IF_FILEREF;
1566                    }

1568                    /*
1569                     * Update the symbol count and the associated name string size.
1570                     * Note, a capabilities symbol must remain as visible as a
1571                     * global symbol.  However, the runtime linker recognizes the
1572                     * hidden requirement and ensures the symbol isn't made globally
1573                     * available at runtime.
1574                     */
1575                    if (ld_sym_reducable(ofl, sdp)) {
1046                    if (SYM_IS_HIDDEN(sdp) && (oflags & FLG_OF_PROCRED)) {
1576                            /*
1577                             * If any reductions are being processed, keep a count
1578                             * of eliminated symbols, and if the symbol is being
1579                             * reduced to local, count it's size for the .symtab.
1580                             */
1581                            if (sdp->sd_flags & FLG_SY_ELIM) {
1582                                    ofl->ofl_elimcnt++;
1583                            } else {
1584                                    ofl->ofl_scopecnt++;
1585                                    if ((((sdp->sd_flags & FLG_SY_REGSYM) == 0) ||
1586                                        sym->st_name) && (st_insert(ofl->ofl_strtab,
1587                                        sdp->sd_name) == -1))
1588                                            return (S_ERROR);
1589                                    if (allow_ldynsym && sym->st_name &&
1590                                        ldynsym_symtype[type]) {
1591                                            ofl->ofl_dynscopecnt++;
1592                                            if (st_insert(ofl->ofl_dynstrtab,
1593                                                sdp->sd_name) == -1)
1594                                                    return (S_ERROR);
1595                                            /* Include it in sort section? */
1596                                            DYNSORT_COUNT(sdp, sym, type, ++);
1597                                    }
1598                            }
1599                    } else {
1600                            ofl->ofl_globcnt++;

1602                            /*
```

```
1603                         * Check to see if this global variable should go into
1604                         * a sort section. Sort sections require a
1605                         * .SUNW_ldynsym section, so, don't check unless a
1606                         * .SUNW_ldynsym is allowed.
1607                         */
1608                        if (allow_ldynsym)
1609                                DYNSORT_COUNT(sdp, sym, type, ++);
1610
1611                        /*
1612                         * If global direct bindings are in effect, or this
1613                         * symbol has bound to a dependency which was specified
1614                         * as requiring direct bindings, and it hasn't
1615                         * explicitly been defined as a non-direct binding
1616                         * symbol, mark it.
1617                         */
1618                        if (((ofl->ofl_dtflags_1 & DF_1_DIRECT) || (isp &&
1619                            (isp->is_file->ifl_flags & FLG_IF_DIRECT))) &&
1620                            ((sdp->sd_flags & FLG_SY_NDIR) == 0))
1621                                sdp->sd_flags |= FLG_SY_DIR;
1622
1623                        /*
1624                         * Insert the symbol name.
1625                         */
1626                        if (((sdp->sd_flags & FLG_SY_REGSYM) == 0) ||
1627                            sym->st_name) {
1628                                if (st_insert(ofl->ofl_strtab,
1629                                    sdp->sd_name) == -1)
1630                                        return (S_ERROR);
1631
1632                                if (!(ofl->ofl_flags & FLG_OF_RELOBJ) &&
1633                                    (st_insert(ofl->ofl_dynstrtab,
1634                                    sdp->sd_name) == -1))
1635                                        return (S_ERROR);
1636                        }
1637
1638                        /*
1639                         * If this section offers a global symbol - record that
1640                         * fact.
1641                         */
1642                        if (isp) {
1643                                isp->is_flags |= FLG_IS_SECTREF;
1644                                isp->is_file->ifl_flags |= FLG_IF_FILEREF;
1645                        }
1646                }
1647        }
1648
1649        /*
1650         * Guidance: Use -z defs|nodefs when building shared objects.
1651         *
1652         * Our caller issues this, unless we mask it out here. So we mask it
1653         * out unless we've issued at least one warnings or fatal error.
1654         */
1655        if (!((oflags & FLG_OF_SHAROBJ) && OFL_GUIDANCE(ofl, FLG_OFG_NO_DEFS) &&
1656            (undef_state & (FLG_OF_FATAL | FLG_OF_WARN))))
1657                ofl->ofl_guideflags |= FLG_OFG_NO_DEFS;
1658
1659        /*
1660         * If we've encountered a fatal error during symbol validation then
1661         * return now.
1662         */
1663        if (ofl->ofl_flags & FLG_OF_FATAL)
1664                return (1);
1665
1666        /*
1667         * Now that symbol resolution is completed, scan any register symbols.
1668         * From now on, we're only interested in those that contribute to the
```



```
1669         * output file.
1670         */
1671        if (ofl->ofl_regsyms) {
1672                int     ndx;
1673
1674                for (ndx = 0; ndx < ofl->ofl_regsymsno; ndx++) {
1675                        if ((sdp = ofl->ofl_regsyms[ndx]) == NULL)
1676                                continue;
1677                        if (sdp->sd_ref != REF_REL_NEED) {
1678                                ofl->ofl_regsyms[ndx] = NULL;
1679                                continue;
1680                        }
1681
1682                        ofl->ofl_regsymcnt++;
1683                        if (sdp->sd_sym->st_name == 0)
1684                                sdp->sd_name = MSG_ORIG(MSG_STR_EMPTY);
1685
1686                        if (SYM_IS_HIDDEN(sdp) ||
1687                            (ELF_ST_BIND(sdp->sd_sym->st_info) == STB_LOCAL))
1688                                ofl->ofl_lregsymcnt++;
1689                }
1690        }
1691
1692        /*
1693         * Generate the .bss section now that we know its size and alignment.
1694         */
1695        if (need_bss) {
1696                if (ld_make_bss(ofl, bsssize, bssalign,
1697                    ld_targ.t_id.id_bss) == S_ERROR)
1698                        return (S_ERROR);
1699        }
1700        if (need_tlsbss) {
1701                if (ld_make_bss(ofl, tlssize, tlsalign,
1702                    ld_targ.t_id.id_tlsbss) == S_ERROR)
1703                        return (S_ERROR);
1704        }
1705 #if     defined(_ELF64)
1706        if ((ld_targ.t_m.m_mach == EM_AMD64) &&
1707            need_lbss && !(oflags & FLG_OF_RELOBJ)) {
1708                if (ld_make_bss(ofl, lbsssize, lbssalign,
1709                    ld_targ.t_id.id_lbss) == S_ERROR)
1710                        return (S_ERROR);
1711        }
1712 #endif
1713        /*
1714         * Determine what entry point symbol we need, and if found save its
1715         * symbol descriptor so that we can update the ELF header entry with the
1716         * symbols value later (see update_oehdr).  Make sure the symbol is
1717         * tagged to ensure its update in case -s is in effect.  Use any -e
1718         * option first, or the default entry points '_start' and 'main'.
1719         */
1720        ret = 0;
1721        if (ofl->ofl_entry) {
1722                if ((sdp = ld_sym_find(ofl->ofl_entry, SYM_NOHASH,
1723                    NULL, ofl)) == NULL) {
1724                        ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_NOENTRY),
1725                            ofl->ofl_entry);
1726                        ret++;
1727                } else if (ensure_sym_local(ofl, sdp,
1728                    MSG_INTL(MSG_SYM_ENTRY)) != 0) {
1729                        ret++;
1730                } else {
1731                        ofl->ofl_entry = (void *)sdp;
1732                }
1733        } else if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_START),
1734            SYM_NOHASH, NULL, ofl)) != NULL) && (ensure_sym_local(ofl,
```

```
1735             sdp, 0) == 0)) {
1736                 ofl->ofl_entry = (void *)sdp;

1738         } else if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_MAIN),
1739             SYM_NOHASH, NULL, ofl)) != NULL) && (ensure_sym_local(ofl,
1740             sdp, 0) == 0)) {
1741                 ofl->ofl_entry = (void *)sdp;
1742         }

1744         /*
1745          * If ld -zdtrace=<sym> was given, then validate that the symbol is
1746          * defined within the current object being built.
1747          */
1748         if ((sdp = ofl->ofl_dtracesym) != 0)
1749                 ret += ensure_sym_local(ofl, sdp, MSG_ORIG(MSG_STR_DTRACE));

1751         /*
1752          * If any initarray, finiarray or preinitarray functions have been
1753          * requested, make sure they are defined within the current object
1754          * being built.
1755          */
1756         if (ofl->ofl_initarray) {
1757                 ret += ensure_array_local(ofl, ofl->ofl_initarray,
1758                     MSG_ORIG(MSG_SYM_INITARRAY));
1759         }
1760         if (ofl->ofl_finiarray) {
1761                 ret += ensure_array_local(ofl, ofl->ofl_finiarray,
1762                     MSG_ORIG(MSG_SYM_FINIARRAY));
1763         }
1764         if (ofl->ofl_preiarray) {
1765                 ret += ensure_array_local(ofl, ofl->ofl_preiarray,
1766                     MSG_ORIG(MSG_SYM_PREINITARRAY));
1767         }

1769         if (ret)
1770                 return (S_ERROR);

1772         /*
1773          * If we're required to record any needed dependencies versioning
1774          * information calculate it now that all symbols have been validated.
1775          */
1776         if ((oflags & (FLG_OF_VERNEED | FLG_OF_NOVERSEC)) == FLG_OF_VERNEED)
1777                 return (ld_vers_check_need(ofl));
1778         else
1779                 return (1);
1780 }
```

*unchanged_portion_omitted_*

```
3098 /*
3099  * STT_SECTION symbols have their st_name field set to NULL, and consequently
3100  * have no name. Generate a name suitable for diagnostic use for such a symbol
3101  * and store it in the input section descriptor. The resulting name will be
3102  * of the form:
3103  *
3104  *      "XXX (section)"
3105  *
3106  * where XXX is the name of the section.
3107  *
3108  * entry:
3109  *      isc - Input section associated with the symbol.
3110  *      fmt - NULL, or format string to use.
3111  *
3112  * exit:
3113  *      Sets isp->is_sym_name to the allocated string. Returns the
3114  *      string pointer, or NULL on allocation failure.
3115  */
```



```
3116 const char *
3117 ld_stt_section_sym_name(Is_desc *isp)
3118 {
3119         const char      *fmt;
3120         char            *str;
3121         size_t          len;

3123         if ((isp == NULL) || (isp->is_name == NULL))
3124                 return (NULL);

3126         if (isp->is_sym_name == NULL) {
3127                 fmt = (isp->is_flags & FLG_IS_GNSTRMRG) ?
3128                     MSG_INTL(MSG_STR_SECTION_MSTR) : MSG_INTL(MSG_STR_SECTION);

3130                 len = strlen(fmt) + strlen(isp->is_name) + 1;

3132                 if ((str = libld_malloc(len)) == NULL)
3133                         return (NULL);
3134                 (void) snprintf(str, len, fmt, isp->is_name);
3135                 isp->is_sym_name = str;
3136         }

3138         return (isp->is_sym_name);
3139 }

3141 /*
3142  * If we're producing a relocatable object and the symbol is eligible for
3143  * COMDAT section, it shouldn't be reduced in scope as that will break the
3144  * COMDAT matching when the output object is later consumed.  Leave it alone,
3145  * and any reduction (and COMDAT) processing will occur then.
3146  *
3147  * Otherwise, any hidden symbol is reduced when reductions are being processed.
3148  */
3149 Boolean
3150 ld_sym_reducable(Ofl_desc *ofl, Sym_desc *sdp)
3151 {
3152         Is_desc *isc = sdp->sd_isc;

3154         if (((ofl->ofl_flags & FLG_OF_RELOBJ) != 0) &&
3155             (isc != NULL) &&
3156             ((isc->is_flags & FLG_IS_COMDAT) != 0)) {
3157                 return (FALSE);
3158         } else {
3159                 return (SYM_IS_HIDDEN(sdp) &&
3160                     (ofl->ofl_flags & FLG_OF_PROCRED));
3161         }
3162 #endif /* ! codereview */
3163 }
```

```
**********************************************************
  118288 Thu Feb  7 01:35:56 2019
new/usr/src/cmd/sgs/libld/common/update.c
10346 ld(1) should not reduce symbol visibility of COMDAT symbols when producing
**********************************************************
______unchanged_portion_omitted_

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

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



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

 231         /*
 232          * Initialize pointers to the symbol table entries and the symbol
 233          * table strings.  Skip the first symbol entry and the first string
 234          * table byte.  Note that if we are not generating any output symbol
 235          * tables we must still generate and update internal copies so
 236          * that the relocation phase has the correct information.
 237          */
 238         if (!(flags & FLG_OF_STRIP) || (flags & FLG_OF_RELOBJ) ||
 239             ((flags & FLG_OF_STATIC) && ofl->ofl_osversym)) {
 240                 symtab = (Sym *)ofl->ofl_ossymtab->os_outdata->d_buf;
 241                 symtab[symtab_ndx++] = _sym;
 242                 if (ofl->ofl_ossymshndx)
 243                         symshndx =
 244                             (Word *)ofl->ofl_ossymshndx->os_outdata->d_buf;
 245         }
 246         if (OFL_ALLOW_DYNSYM(ofl)) {
 247                 dynsym = (Sym *)ofl->ofl_osdynsym->os_outdata->d_buf;
 248                 dynsym[dynsym_ndx++] = _sym;
 249                 /*
 250                  * If we are also constructing a .SUNW_ldynsym section
 251                  * to contain local function symbols, then set it up too.
 252                  */
 253                 if (ofl->ofl_osldynsym) {
 254                         ldynsym = (Sym *)ofl->ofl_osldynsym->os_outdata->d_buf;
 255                         ldynsym[ldynsym_ndx++] = _sym;
 256                         ldynsym_cnt = 1 + ofl->ofl_dynlocscnt +
```

```
257                             ofl->ofl_dynscopecnt;

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

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

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

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

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



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

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

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

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

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

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

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

385                 if (phd->p_type == PT_LOAD) {
386                         if (sgp->sg_osdescs != NULL) {
387                                 Word    _flags = phd->p_flags & (PF_W | PF_R);
```

```
389                                 if (_flags == PF_R)
390                                         tsgp = sgp;
391                                 else if (_flags == (PF_W | PF_R))
392                                         dsgp = sgp;
393                         } else if (sgp->sg_flags & FLG_SG_EMPTY)
394                                 esgp = sgp;
395                 }
396
397                 /*
398                  * Generate a section symbol for each output section.
399                  */
400                 for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
401                         Word    sectndx;
402
403                         sym = &_sym;
404                         sym->st_value = osp->os_shdr->sh_addr;
405                         sym->st_info = ELF_ST_INFO(STB_LOCAL, STT_SECTION);
406                         /* LINTED */
407                         sectndx = elf_ndxscn(osp->os_scn);
408
409                         if (symtab) {
410                                 if (sectndx >= SHN_LORESERVE) {
411                                         symshndx[symtab_ndx] = sectndx;
412                                         sym->st_shndx = SHN_XINDEX;
413                                 } else {
414                                         /* LINTED */
415                                         sym->st_shndx = (Half)sectndx;
416                                 }
417                                 symtab[symtab_ndx++] = *sym;
418                         }
419
420                         if (dynsym && (osp->os_flags & FLG_OS_OUTREL))
421                                 dynsym[dynsym_ndx++] = *sym;
422
423                         if ((dynsym == NULL) ||
424                             (osp->os_flags & FLG_OS_OUTREL)) {
425                                 if (versym)
426                                         versym[*symndx - 1] = 0;
427                                 osp->os_identndx = *symndx - 1;
428                                 DBG_CALL(Dbg_syms_sec_entry(ofl->ofl_lml,
429                                     osp->os_identndx, sgp, osp));
430                         }
431
432                         /*
433                          * Generate the .shstrtab for this section.
434                          */
435                         (void) st_setstring(shstrtab, osp->os_name, &stoff);
436                         osp->os_shdr->sh_name = (Word)stoff;
437
438                         /*
439                          * Find the section index for our special symbols.
440                          */
441                         if (sgp == tsgp) {
442                                 /* LINTED */
443                                 etext_ndx = elf_ndxscn(osp->os_scn);
444                         } else if (dsgp == sgp) {
445                                 if (osp->os_shdr->sh_type != SHT_NOBITS) {
446                                         /* LINTED */
447                                         edata_ndx = elf_ndxscn(osp->os_scn);
448                                 }
449                         }
450
451                         if (start_set == 0) {
452                                 start = sgp->sg_phdr.p_vaddr;
453                                 /* LINTED */
454                                 start_ndx = elf_ndxscn(osp->os_scn);
```



```
455                                 start_set++;
456                         }
457
458                         /*
459                          * While we're here, determine whether a .init or .fini
460                          * section exist.
461                          */
462                         if ((iosp == NULL) && (strcmp(osp->os_name,
463                             MSG_ORIG(MSG_SCN_INIT)) == 0))
464                                 iosp = osp;
465                         if ((fosp == NULL) && (strcmp(osp->os_name,
466                             MSG_ORIG(MSG_SCN_FINI)) == 0))
467                                 fosp = osp;
468                 }
469         }
470
471         /*
472          * Add local register symbols to the .dynsym.  These are required as
473          * DT_REGISTER .dynamic entries must have a symbol to reference.
474          */
475         if (ofl->ofl_regsyms && dynsym) {
476                 int     ndx;
477
478                 for (ndx = 0; ndx < ofl->ofl_regsymsno; ndx++) {
479                         Sym_desc        *rsdp;
480
481                         if ((rsdp = ofl->ofl_regsyms[ndx]) == NULL)
482                                 continue;
483
484                         if (!SYM_IS_HIDDEN(rsdp) &&
485                             (ELF_ST_BIND(rsdp->sd_sym->st_info) != STB_LOCAL))
486                                 continue;
487
488                         dynsym[dynsym_ndx] = *(rsdp->sd_sym);
489                         rsdp->sd_symndx = *symndx;
490
491                         if (dynsym[dynsym_ndx].st_name) {
492                                 (void) st_setstring(dynstr, rsdp->sd_name,
493                                     &stoff);
494                                 dynsym[dynsym_ndx].st_name = stoff;
495                         }
496                         dynsym_ndx++;
497                 }
498         }
499
500         /*
501          * Having traversed all the output segments, warn the user if the
502          * traditional text or data segments don't exist.  Otherwise from these
503          * segments establish the values for 'etext', 'edata', 'end', 'END',
504          * and 'START'.
505          */
506         if (!(flags & FLG_OF_RELOBJ)) {
507                 Sg_desc *sgp;
508
509                 if (tsgp)
510                         etext = tsgp->sg_phdr.p_vaddr + tsgp->sg_phdr.p_filesz;
511                 else {
512                         etext = (Addr)0;
513                         etext_ndx = SHN_ABS;
514                         etext_abs = 1;
515                         if (flags & FLG_OF_VERBOSE)
516                                 ld_eprintf(ofl, ERR_WARNING,
517                                     MSG_INTL(MSG_UPD_NOREADSEG));
518                 }
519                 if (dsgp) {
520                         edata = dsgp->sg_phdr.p_vaddr + dsgp->sg_phdr.p_filesz;
```

```
521                 } else {
522                         edata = (Addr)0;
523                         edata_ndx = SHN_ABS;
524                         edata_abs = 1;
525                         if (flags & FLG_OF_VERBOSE)
526                                 ld_eprintf(ofl, ERR_WARNING,
527                                     MSG_INTL(MSG_UPD_NORDWRSEG));
528                 }
529
530                 if (dsgp == NULL) {
531                         if (tsgp)
532                                 sgp = tsgp;
533                         else
534                                 sgp = 0;
535                 } else if (tsgp == NULL)
536                         sgp = dsgp;
537                 else if (dsgp->sg_phdr.p_vaddr > tsgp->sg_phdr.p_vaddr)
538                         sgp = dsgp;
539                 else if (dsgp->sg_phdr.p_vaddr < tsgp->sg_phdr.p_vaddr)
540                         sgp = tsgp;
541                 else {
542                         /*
543                          * One of the segments must be of zero size.
544                          */
545                         if (tsgp->sg_phdr.p_memsz)
546                                 sgp = tsgp;
547                         else
548                                 sgp = dsgp;
549                 }
550
551                 if (esgp && (esgp->sg_phdr.p_vaddr > sgp->sg_phdr.p_vaddr))
552                         sgp = esgp;
553
554                 if (sgp) {
555                         end = sgp->sg_phdr.p_vaddr + sgp->sg_phdr.p_memsz;
556
557                         /*
558                          * If the last loadable segment is a read-only segment,
559                          * then the application which uses the symbol _end to
560                          * find the beginning of writable heap area may cause
561                          * segmentation violation. We adjust the value of the
562                          * _end to skip to the next page boundary.
563                          *
564                          * 6401812 System interface which returs beginning
565                          *         heap would be nice.
566                          * When the above RFE is implemented, the changes below
567                          * could be changed in a better way.
568                          */
569                         if ((sgp->sg_phdr.p_flags & PF_W) == 0)
570                                 end = (Addr)S_ROUND(end, sysconf(_SC_PAGESIZE));
571
572                         /*
573                          * If we're dealing with a memory reservation there are
574                          * no sections to establish an index for _end, so assign
575                          * it as an absolute.
576                          */
577                         if (sgp->sg_osdescs != NULL) {
578                                 /*
579                                  * Determine the last section for this segment.
580                                  */
581                                 Os_desc *osp = sgp->sg_osdescs->apl_data
582                                     [sgp->sg_osdescs->apl_nitems - 1];
583
584                                 /* LINTED */
585                                 end_ndx = elf_ndxscn(osp->os_scn);
586                         } else {
```



```
587                                 end_ndx = SHN_ABS;
588                                 end_abs = 1;
589                         }
590                 } else {
591                         end = (Addr) 0;
592                         end_ndx = SHN_ABS;
593                         end_abs = 1;
594                         ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_UPD_NOSEG));
595                 }
596         }
597
598         /*
599          * Initialize the scoped symbol table entry point.  This is for all
600          * the global symbols that have been scoped to locals and will be
601          * filled in during global symbol processing so that we don't have
602          * to traverse the globals symbol hash array more than once.
603          */
604         if (symtab) {
605                 scopesym_bndx = symtab_ndx;
606                 scopesym_ndx = scopesym_bndx;
607                 symtab_ndx += ofl->ofl_scopecnt;
608         }
609
610         /*
611          * If expanding partially expanded symbols under '-z nopartial',
612          * prepare to do that.
613          */
614         if (ofl->ofl_isparexpn) {
615                 osp = ofl->ofl_isparexpn->is_osdesc;
616                 parexpnbase = parexpnaddr = (Addr)(osp->os_shdr->sh_addr +
617                     ofl->ofl_isparexpn->is_indata->d_off);
618                 /* LINTED */
619                 parexpnndx = elf_ndxscn(osp->os_scn);
620                 ofl->ofl_parexpnndx = osp->os_identndx;
621         }
622
623         /*
624          * If we are generating a .symtab collect all the local symbols,
625          * assigning a new virtual address or displacement (value).
626          */
627         for (APLIST_TRAVERSE(ofl->ofl_objs, idx1, ifl)) {
628                 Xword           lndx, local = ifl->ifl_locscnt;
629                 Cap_desc        *cdp = ifl->ifl_caps;
630
631                 for (lndx = 1; lndx < local; lndx++) {
632                         Gotndx          *gnp;
633                         uchar_t         type;
634                         Word            *_symshndx;
635                         int             enter_in_symtab, enter_in_ldynsym;
636                         int             update_done;
637
638                         sdp = ifl->ifl_oldndx[lndx];
639                         sym = sdp->sd_sym;
640
641                         /*
642                          * Assign a got offset if necessary.
643                          */
644                         if ((ld_targ.t_mr.mr_assign_got != NULL) &&
645                             (*ld_targ.t_mr.mr_assign_got)(ofl, sdp) == S_ERROR)
646                                 return ((Addr)S_ERROR);
647
648                         if (DBG_ENABLED) {
649                                 Aliste  idx2;
650
651                                 for (ALIST_TRAVERSE(sdp->sd_GOTndxs,
652                                     idx2, gnp)) {
```

```
653                                     gottable->gt_sym = sdp;
654                                     gottable->gt_gndx.gn_gotndx =
655                                         gnp->gn_gotndx;
656                                     gottable->gt_gndx.gn_addend =
657                                         gnp->gn_addend;
658                                     gottable++;
659                             }
660                     }

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

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

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

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

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

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

715                             /*
716                              * Provided this isn't an unnamed register
717                              * symbol, update its name.
718                              */
```



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

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

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

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

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

779                     /*
780                      * If we are expanding the locally bound partially
781                      * initialized symbols, then update the address here.
782                      */
783                     if (ofl->ofl_isparexpn &&
784                         (sdp->sd_flags & FLG_SY_PAREXPN) && !update_done) {
```

```
785                             sym->st_shndx = parexpnndx;
786                             sdp->sd_isc = ofl->ofl_isparexpn;
787                             sym->st_value = parexpnaddr;
788                             parexpnaddr += sym->st_size;
789                             if ((flags & FLG_OF_RELOBJ) == 0)
790                                     sym->st_value -= parexpnbase;
791                     }
793                     /*
794                      * If this isn't an UNDEF symbol (ie. an input section
795                      * is associated), update the symbols value and index.
796                      */
797                     if (((isc = sdp->sd_isc) != NULL) && !update_done) {
798                             Word    sectndx;
800                             osp = isc->is_osdesc;
801                             /* LINTED */
802                             sym->st_value +=
803                                 (Off)_elf_getxoff(isc->is_indata);
804                             if ((flags & FLG_OF_RELOBJ) == 0) {
805                                     sym->st_value += osp->os_shdr->sh_addr;
806                                     /*
807                                      * TLS symbols are relative to
808                                      * the TLS segment.
809                                      */
810                                     if ((type == STT_TLS) &&
811                                         (ofl->ofl_tlsphdr)) {
812                                             sym->st_value -=
813                                                 ofl->ofl_tlsphdr->p_vaddr;
814                                     }
815                             }
816                             /* LINTED */
817                             if ((sdp->sd_shndx = sectndx =
818                                 elf_ndxscn(osp->os_scn)) >= SHN_LORESERVE) {
819                                     if (_symshndx) {
820                                             *_symshndx = sectndx;
821                                     }
822                                     sym->st_shndx = SHN_XINDEX;
823                             } else {
824                                     /* LINTED */
825                                     sym->st_shndx = sectndx;
826                             }
827                     }
829                     /*
830                      * If entering the symbol in both the symtab and the
831                      * ldynsym, then the one in symtab needs to be
832                      * copied to ldynsym. If it is only in the ldynsym,
833                      * then the code above already set it up and we have
834                      * nothing more to do here.
835                      */
836                     if (enter_in_symtab && enter_in_ldynsym) {
837                             ldynsym[ldynsym_ndx] = *sym;
838                             (void) st_setstring(dynstr, sdp->sd_name,
839                                 &stoff);
840                             ldynsym[ldynsym_ndx].st_name = stoff;
842                             if (_symshndx && ldynshndx)
843                                     ldynshndx[ldynsym_ndx] = *_symshndx;
845                             /* Add it to sort section if it qualifies */
846                             ADD_TO_DYNSORT(sdp, sym, type, ldynsym_ndx);
848                             ldynsym_ndx++;
849                     }
850             }
```



```
852             /*
853              * If this input file has undergone object to symbol
854              * capabilities conversion, supply any new capabilities symbols.
855              * These symbols are copies of the original global symbols, and
856              * follow the existing local symbols that are supplied from this
857              * input file (which are identified with a preceding STT_FILE).
858              */
859             if (symtab && cdp && cdp->ca_syms) {
860                     Aliste          idx2;
861                     Cap_sym         *csp;
863                     for (APLIST_TRAVERSE(cdp->ca_syms, idx2, csp)) {
864                             Is_desc *isp;
866                             sdp = csp->cs_sdp;
867                             sym = sdp->sd_sym;
869                             if ((isp = sdp->sd_isc) != NULL) {
870                                     Os_desc *osp = isp->is_osdesc;
872                                     /*
873                                      * Update the symbols value.
874                                      */
875                                     /* LINTED */
876                                     sym->st_value +=
877                                         (Off)_elf_getxoff(isp->is_indata);
878                                     if ((flags & FLG_OF_RELOBJ) == 0)
879                                             sym->st_value +=
880                                                 osp->os_shdr->sh_addr;
882                                     /*
883                                      * Update the symbols section index.
884                                      */
885                                     sdp->sd_shndx = sym->st_shndx =
886                                         elf_ndxscn(osp->os_scn);
887                             }
889                             symtab[symtab_ndx] = *sym;
890                             (void) st_setstring(strtab, sdp->sd_name,
891                                 &stoff);
892                             symtab[symtab_ndx].st_name = stoff;
893                             sdp->sd_symndx = symtab_ndx++;
894                     }
895             }
896     }
898     symtab_gbl_bndx = symtab_ndx;   /* .symtab index of 1st global entry */
900     /*
901      * Two special symbols are '_init' and '_fini'.  If these are supplied
902      * by crti.o then they are used to represent the total concatenation of
903      * the '.init' and '.fini' sections.
904      *
905      * Determine whether any .init or .fini sections exist.  If these
906      * sections exist and a dynamic object is being built, but no '_init'
907      * or '_fini' symbols are found, then the user is probably building
908      * this object directly from ld(1) rather than using a compiler driver
909      * that provides the symbols via crt's.
910      *
911      * If the .init or .fini section exist, and their associated symbols,
912      * determine the size of the sections and updated the symbols value
913      * accordingly.
914      */
915     if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_INIT_U), SYM_NOHASH, 0,
916         ofl)) != NULL) && (sdp->sd_ref == REF_REL_NEED) && sdp->sd_isc &&
```

```
917             (sdp->sd_isc->is_osdesc == iosp)) {
918                 if (ld_sym_copy(sdp) == S_ERROR)
919                         return ((Addr)S_ERROR);
920                 sdp->sd_sym->st_size = sdp->sd_isc->is_osdesc->os_shdr->sh_size;
921
922         } else if (iosp && !(flags & FLG_OF_RELOBJ)) {
923                 ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_NOCRT),
924                     MSG_ORIG(MSG_SYM_INIT_U), MSG_ORIG(MSG_SCN_INIT));
925         }
926
927         if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_FINI_U), SYM_NOHASH, 0,
928             ofl)) != NULL) && (sdp->sd_ref == REF_REL_NEED) && sdp->sd_isc &&
929             (sdp->sd_isc->is_osdesc == fosp)) {
930                 if (ld_sym_copy(sdp) == S_ERROR)
931                         return ((Addr)S_ERROR);
932                 sdp->sd_sym->st_size = sdp->sd_isc->is_osdesc->os_shdr->sh_size;
933
934         } else if (fosp && !(flags & FLG_OF_RELOBJ)) {
935                 ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_NOCRT),
936                     MSG_ORIG(MSG_SYM_FINI_U), MSG_ORIG(MSG_SCN_FINI));
937         }
938
939         /*
940          * Assign .bss information for use with updating COMMON symbols.
941          */
942         if (ofl->ofl_isbss) {
943                 isc = ofl->ofl_isbss;
944                 osp = isc->is_osdesc;
945
946                 bssaddr = osp->os_shdr->sh_addr +
947                     (Off)_elf_getxoff(isc->is_indata);
948                 /* LINTED */
949                 bssndx = elf_ndxscn(osp->os_scn);
950         }
951
952 #if     defined(_ELF64)
953         /*
954          * For amd64 target, assign .lbss information for use
955          * with updating LCOMMON symbols.
956          */
957         if ((ld_targ.t_m.m_mach == EM_AMD64) && ofl->ofl_islbss) {
958                 osp = ofl->ofl_islbss->is_osdesc;
959
960                 lbssaddr = osp->os_shdr->sh_addr +
961                     (Off)_elf_getxoff(ofl->ofl_islbss->is_indata);
962                 /* LINTED */
963                 lbssndx = elf_ndxscn(osp->os_scn);
964         }
965 #endif
966         /*
967          * Assign .tlsbss information for use with updating COMMON symbols.
968          */
969         if (ofl->ofl_istlsbss) {
970                 osp = ofl->ofl_istlsbss->is_osdesc;
971                 tlsbssaddr = osp->os_shdr->sh_addr +
972                     (Off)_elf_getxoff(ofl->ofl_istlsbss->is_indata);
973                 /* LINTED */
974                 tlsbssndx = elf_ndxscn(osp->os_scn);
975         }
976
977         if ((sorted_syms = libld_calloc(ofl->ofl_globcnt +
978             ofl->ofl_elimcnt + ofl->ofl_scopecnt,
979             sizeof (*sorted_syms))) == NULL)
980                 return ((Addr)S_ERROR);
981
982         scndx = 0;
```



```
983         ssndx = ofl->ofl_scopecnt + ofl->ofl_elimcnt;
984
985         DBG_CALL(Dbg_syms_up_title(ofl->ofl_lml));
986
987         /*
988          * Traverse the internal symbol table updating global symbol information
989          * and allocating common.
990          */
991         for (sav = avl_first(&ofl->ofl_symavl); sav;
992             sav = AVL_NEXT(&ofl->ofl_symavl, sav)) {
993                 Sym     *symptr;
994                 int     local;
995                 int     restore;
996
997                 sdp = sav->sav_sdp;
998
999                 /*
1000                 * Ignore any symbols that have been marked as invalid during
1001                 * input processing.  Providing these aren't used for
1002                 * relocation, they will be dropped from the output image.
1003                 */
1004                if (sdp->sd_flags & FLG_SY_INVALID) {
1005                        DBG_CALL(Dbg_syms_old(ofl, sdp));
1006                        DBG_CALL(Dbg_syms_ignore(ofl, sdp));
1007                        continue;
1008                }
1009
1010                /*
1011                 * Only needed symbols are copied to the output symbol table.
1012                 */
1013                if (sdp->sd_ref == REF_DYN_SEEN)
1014                        continue;
1015
1016                if (ld_sym_reducable(ofl, sdp))
1016                if (SYM_IS_HIDDEN(sdp) && (flags & FLG_OF_PROCRED))
1017                        local = 1;
1018                else
1019                        local = 0;
1020
1021                if (local || (ofl->ofl_hashbkts == 0)) {
1022                        sorted_syms[scndx++].sl_sdp = sdp;
1023                } else {
1024                        sorted_syms[ssndx].sl_hval = sdp->sd_aux->sa_hash %
1025                            ofl->ofl_hashbkts;
1026                        sorted_syms[ssndx].sl_sdp = sdp;
1027                        ssndx++;
1028                }
1029
1030                /*
1031                 * Note - expand the COMMON symbols here because an address
1032                 * must be assigned to them in the same order that space was
1033                 * calculated in sym_validate().  If this ordering isn't
1034                 * followed differing alignment requirements can throw us all
1035                 * out of whack.
1036                 *
1037                 * The expanded .bss global symbol is handled here as well.
1038                 *
1039                 * The actual adding entries into the symbol table still occurs
1040                 * below in hashbucket order.
1041                 */
1042                symptr = sdp->sd_sym;
1043                restore = 0;
1044                if ((sdp->sd_flags & FLG_SY_PAREXPN) ||
1045                    ((sdp->sd_flags & FLG_SY_SPECSEC) &&
1046                    (sdp->sd_shndx = symptr->st_shndx) == SHN_COMMON)) {
```

```
1048                        /*
1049                         * An expanded symbol goes to a special .data section
1050                         * prepared for that purpose (ofl->ofl_isparexpn).
1051                         * Assign COMMON allocations to .bss.
1052                         * Otherwise leave it as is.
1053                         */
1054                        if (sdp->sd_flags & FLG_SY_PAREXPN) {
1055                                restore = 1;
1056                                sdp->sd_shndx = parexpnndx;
1057                                sdp->sd_flags &= ~FLG_SY_SPECSEC;
1058                                symptr->st_value = (Xword) S_ROUND(
1059                                    parexpnaddr, symptr->st_value);
1060                                parexpnaddr = symptr->st_value +
1061                                    symptr->st_size;
1062                                sdp->sd_isc = ofl->ofl_isparexpn;
1063                                sdp->sd_flags |= FLG_SY_COMMEXP;
1064
1065                        } else if (ELF_ST_TYPE(symptr->st_info) != STT_TLS &&
1066                            (local || !(flags & FLG_OF_RELOBJ))) {
1067                                restore = 1;
1068                                sdp->sd_shndx = bssndx;
1069                                sdp->sd_flags &= ~FLG_SY_SPECSEC;
1070                                symptr->st_value = (Xword)S_ROUND(bssaddr,
1071                                    symptr->st_value);
1072                                bssaddr = symptr->st_value + symptr->st_size;
1073                                sdp->sd_isc = ofl->ofl_isbss;
1074                                sdp->sd_flags |= FLG_SY_COMMEXP;
1075
1076                        } else if (ELF_ST_TYPE(symptr->st_info) == STT_TLS &&
1077                            (local || !(flags & FLG_OF_RELOBJ))) {
1078                                restore = 1;
1079                                sdp->sd_shndx = tlsbssndx;
1080                                sdp->sd_flags &= ~FLG_SY_SPECSEC;
1081                                symptr->st_value = (Xword)S_ROUND(tlsbssaddr,
1082                                    symptr->st_value);
1083                                tlsbssaddr = symptr->st_value + symptr->st_size;
1084                                sdp->sd_isc = ofl->ofl_istlsbss;
1085                                sdp->sd_flags |= FLG_SY_COMMEXP;
1086                                /*
1087                                 * TLS symbols are relative to the TLS segment.
1088                                 */
1089                                symptr->st_value -= ofl->ofl_tlsphdr->p_vaddr;
1090                        }
1091 #if     defined(_ELF64)
1092                } else if ((ld_targ.t_m.m_mach == EM_AMD64) &&
1093                    (sdp->sd_flags & FLG_SY_SPECSEC) &&
1094                    ((sdp->sd_shndx = symptr->st_shndx) ==
1095                    SHN_X86_64_LCOMMON) &&
1096                    ((local || !(flags & FLG_OF_RELOBJ)))) {
1097                        restore = 1;
1098                        sdp->sd_shndx = lbssndx;
1099                        sdp->sd_flags &= ~FLG_SY_SPECSEC;
1100                        symptr->st_value = (Xword)S_ROUND(lbssaddr,
1101                            symptr->st_value);
1102                        lbssaddr = symptr->st_value + symptr->st_size;
1103                        sdp->sd_isc = ofl->ofl_islbss;
1104                        sdp->sd_flags |= FLG_SY_COMMEXP;
1105 #endif
1106                }
1107
1108                if (restore != 0) {
1109                        uchar_t         type, bind;
1110
1111                        /*
1112                         * Make sure this COMMON symbol is returned to the same
1113                         * binding as was defined in the original relocatable
```



```
1114                         * object reference.
1115                         */
1116                        type = ELF_ST_TYPE(symptr->st_info);
1117                        if (sdp->sd_flags & FLG_SY_GLOBREF)
1118                                bind = STB_GLOBAL;
1119                        else
1120                                bind = STB_WEAK;
1121
1122                        symptr->st_info = ELF_ST_INFO(bind, type);
1123                }
1124        }
1125
1126        /*
1127         * If this is a dynamic object then add any local capabilities symbols.
1128         */
1129        if (dynsym && ofl->ofl_capfamilies) {
1130                Cap_avlnode     *cav;
1131
1132                for (cav = avl_first(ofl->ofl_capfamilies); cav;
1133                    cav = AVL_NEXT(ofl->ofl_capfamilies, cav)) {
1134                        Cap_sym         *csp;
1135                        Aliste          idx;
1136
1137                        for (APLIST_TRAVERSE(cav->cn_members, idx, csp)) {
1138                                sdp = csp->cs_sdp;
1139
1140                                DBG_CALL(Dbg_syms_created(ofl->ofl_lml,
1141                                    sdp->sd_name));
1142                                DBG_CALL(Dbg_syms_entered(ofl, sdp->sd_sym,
1143                                    sdp));
1144
1145                                dynsym[dynsym_ndx] = *sdp->sd_sym;
1146
1147                                (void) st_setstring(dynstr, sdp->sd_name,
1148                                    &stoff);
1149                                dynsym[dynsym_ndx].st_name = stoff;
1150
1151                                sdp->sd_sym = &dynsym[dynsym_ndx];
1152                                sdp->sd_symndx = dynsym_ndx;
1153
1154                                /*
1155                                 * Indicate that this is a capabilities symbol.
1156                                 * Note, that this identification only provides
1157                                 * information regarding the symbol that is
1158                                 * visible from elfdump(1) -y.  The association
1159                                 * of a symbol to its capabilities is derived
1160                                 * from a .SUNW_capinfo entry.
1161                                 */
1162                                if (syminfo) {
1163                                        syminfo[dynsym_ndx].si_flags |=
1164                                            SYMINFO_FLG_CAP;
1165                                }
1166
1167                                dynsym_ndx++;
1168                        }
1169                }
1170        }
1171
1172        if (ofl->ofl_hashbkts) {
1173                qsort(sorted_syms + ofl->ofl_scopecnt + ofl->ofl_elimcnt,
1174                    ofl->ofl_globcnt, sizeof (Sym_s_list),
1175                    (int (*)(const void *, const void *))sym_hash_compare);
1176        }
1177
1178        for (ssndx = 0; ssndx < (ofl->ofl_elimcnt + ofl->ofl_scopecnt +
1179            ofl->ofl_globcnt); ssndx++) {
```

```
1180                const char      *name;
1181                Sym             *sym;
1182                Sym_aux         *sap;
1183                Half            spec;
1184                int             local = 0, dynlocal = 0, enter_in_symtab;
1185                Gotndx          *gnp;
1186                Word            sectndx;

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

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

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

1203                if (DBG_ENABLED) {
1204                        Aliste  idx2;

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

1604                        continue;
1605                }
1606        }

1608        DBG_CALL(Dbg_syms_old(ofl, sdp));

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

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



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

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

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

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

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

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

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

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

```
1707                if (spec) {
1708                        switch (spec) {
1709                        case SDAUX_ID_ETEXT:
1710                                sym->st_value = etext;
1711                                sectndx = etext_ndx;
1712                                if (etext_abs)
1713                                        sdp->sd_flags |= FLG_SY_SPECSEC;
1714                                else
1715                                        sdp->sd_flags &= ~FLG_SY_SPECSEC;
1716                                break;
1717                        case SDAUX_ID_EDATA:
1718                                sym->st_value = edata;
1719                                sectndx = edata_ndx;
1720                                if (edata_abs)
1721                                        sdp->sd_flags |= FLG_SY_SPECSEC;
1722                                else
1723                                        sdp->sd_flags &= ~FLG_SY_SPECSEC;
1724                                break;
1725                        case SDAUX_ID_END:
1726                                sym->st_value = end;
1727                                sectndx = end_ndx;
1728                                if (end_abs)
1729                                        sdp->sd_flags |= FLG_SY_SPECSEC;
1730                                else
1731                                        sdp->sd_flags &= ~FLG_SY_SPECSEC;
1732                                break;
1733                        case SDAUX_ID_START:
1734                                sym->st_value = start;
1735                                sectndx = start_ndx;
1736                                sdp->sd_flags &= ~FLG_SY_SPECSEC;
1737                                break;
1738                        case SDAUX_ID_DYN:
1739                                if (flags & FLG_OF_DYNAMIC) {
1740                                        sym->st_value = ofl->
1741                                            ofl_osdynamic->os_shdr->sh_addr;
1742                                        /* LINTED */
1743                                        sectndx = elf_ndxscn(
1744                                            ofl->ofl_osdynamic->os_scn);
1745                                        sdp->sd_flags &= ~FLG_SY_SPECSEC;
1746                                }
1747                                break;
1748                        case SDAUX_ID_PLT:
1749                                if (ofl->ofl_osplt) {
1750                                        sym->st_value = ofl->
1751                                            ofl_osplt->os_shdr->sh_addr;
1752                                        /* LINTED */
1753                                        sectndx = elf_ndxscn(
1754                                            ofl->ofl_osplt->os_scn);
1755                                        sdp->sd_flags &= ~FLG_SY_SPECSEC;
1756                                }
1757                                break;
1758                        case SDAUX_ID_GOT:
1759                                /*
1760                                 * Symbol bias for negative growing tables is
1761                                 * stored in symbol's value during
1762                                 * allocate_got().
1763                                 */
1764                                sym->st_value += ofl->
1765                                    ofl_osgot->os_shdr->sh_addr;
1766                                /* LINTED */
1767                                sectndx = elf_ndxscn(ofl->
1768                                    ofl_osgot->os_scn);
1769                                sdp->sd_flags &= ~FLG_SY_SPECSEC;
1770                                break;
1771                        default:
1772                                /* NOTHING */
```



```
1773                                    ;
1774                            }
1775                    }

1777                    /*
1778                     * If a plt index has been assigned to an undefined function,
1779                     * update the symbols value to the appropriate .plt address.
1780                     */
1781                    if ((flags & FLG_OF_DYNAMIC) && (flags & FLG_OF_EXEC) &&
1782                        (sdp->sd_file) &&
1783                        (sdp->sd_file->ifl_ehdr->e_type == ET_DYN) &&
1784                        (ELF_ST_TYPE(sym->st_info) == STT_FUNC) &&
1785                        !(flags & FLG_OF_BFLAG)) {
1786                            if (sap->sa_PLTndx)
1787                                    sym->st_value =
1788                                        (*ld_targ.t_mr.mr_calc_plt_addr)(sdp, ofl);
1789                    }

1791                    /*
1792                     * Finish updating the symbols.
1793                     */

1795                    /*
1796                     * Sym Update: if scoped local - set local binding
1797                     */
1798                    if (local)
1799                            sym->st_info = ELF_ST_INFO(STB_LOCAL,
1800                                ELF_ST_TYPE(sym->st_info));

1802                    /*
1803                     * Sym Updated: If both the .symtab and .dynsym
1804                     * are present then we've actually updated the information in
1805                     * the .dynsym, therefore copy this same information to the
1806                     * .symtab entry.
1807                     */
1808                    sdp->sd_shndx = sectndx;
1809                    if (enter_in_symtab && dynsym && (!local || dynlocal)) {
1810                            Word _symndx = dynlocal ? scopesym_ndx : symtab_ndx;

1812                            symtab[_symndx].st_value = sym->st_value;
1813                            symtab[_symndx].st_size = sym->st_size;
1814                            symtab[_symndx].st_info = sym->st_info;
1815                            symtab[_symndx].st_other = sym->st_other;
1816                    }

1818                    if (enter_in_symtab) {
1819                            Word    _symndx;

1821                            if (local)
1822                                    _symndx = scopesym_ndx++;
1823                            else
1824                                    _symndx = symtab_ndx++;
1825                            if (((sdp->sd_flags & FLG_SY_SPECSEC) == 0) &&
1826                                (sectndx >= SHN_LORESERVE)) {
1827                                    assert(symshndx != NULL);
1828                                    symshndx[_symndx] = sectndx;
1829                                    symtab[_symndx].st_shndx = SHN_XINDEX;
1830                            } else {
1831                                    /* LINTED */
1832                                    symtab[_symndx].st_shndx = (Half)sectndx;
1833                            }
1834                    }

1836                    if (dynsym && (!local || dynlocal)) {
1837                            /*
1838                             * dynsym and ldynsym are distinct tables, so
```

```
1839                         * we use indirection to access the right one
1840                         * and the related extended section index array.
1841                         */
1842                        Word    _symndx;
1843                        Sym     *_dynsym;
1844                        Word    *_dynshndx;

1846                        if (!local) {
1847                                _symndx = dynsym_ndx++;
1848                                _dynsym = dynsym;
1849                                _dynshndx = dynshndx;
1850                        } else {
1851                                _symndx = ldynscopesym_ndx++;
1852                                _dynsym = ldynsym;
1853                                _dynshndx = ldynshndx;
1854                        }
1855                        if (((sdp->sd_flags & FLG_SY_SPECSEC) == 0) &&
1856                            (sectndx >= SHN_LORESERVE)) {
1857                                assert(_dynshndx != NULL);
1858                                _dynshndx[_symndx] = sectndx;
1859                                _dynsym[_symndx].st_shndx = SHN_XINDEX;
1860                        } else {
1861                                /* LINTED */
1862                                _dynsym[_symndx].st_shndx = (Half)sectndx;
1863                        }
1864                }

1866                DBG_CALL(Dbg_syms_new(ofl, sym, sdp));
1867        }

1869        /*
1870         * Now that all the symbols have been processed update any weak symbols
1871         * information (ie. copy all information except 'st_name').  As both
1872         * symbols will be represented in the output, return the weak symbol to
1873         * its correct type.
1874         */
1875        for (ALIST_TRAVERSE(weak, idx1, wkp)) {
1876                Sym_desc        *sdp, *_sdp;
1877                Sym             *sym, *_sym, *__sym;
1878                uchar_t         bind;

1880                sdp = wkp->wk_weak;
1881                _sdp = wkp->wk_alias;
1882                _sym = __sym = _sdp->sd_sym;

1884                sdp->sd_flags |= FLG_SY_WEAKDEF;

1886                /*
1887                 * If the symbol definition has been scoped then assign it to
1888                 * be local, otherwise if it's from a shared object then we need
1889                 * to maintain the binding of the original reference.
1890                 */
1891                if (SYM_IS_HIDDEN(sdp)) {
1892                        if (ld_sym_reducable(ofl, sdp))
1892                        if (flags & FLG_OF_PROCRED)
1893                                bind = STB_LOCAL;
1894                        else
1895                                bind = STB_WEAK;
1896                } else if ((sdp->sd_ref == REF_DYN_NEED) &&
1897                    (sdp->sd_flags & FLG_SY_GLOBREF))
1898                        bind = STB_GLOBAL;
1899                else
1900                        bind = STB_WEAK;

1902                DBG_CALL(Dbg_syms_old(ofl, sdp));
1903                if ((sym = wkp->wk_symtab) != NULL) {
```



```
1904                        sym->st_value = _sym->st_value;
1905                        sym->st_size = _sym->st_size;
1906                        sym->st_other = _sym->st_other;
1907                        sym->st_shndx = _sym->st_shndx;
1908                        sym->st_info = ELF_ST_INFO(bind,
1909                            ELF_ST_TYPE(sym->st_info));
1910                        __sym = sym;
1911                }
1912                if ((sym = wkp->wk_dynsym) != NULL) {
1913                        sym->st_value = _sym->st_value;
1914                        sym->st_size = _sym->st_size;
1915                        sym->st_other = _sym->st_other;
1916                        sym->st_shndx = _sym->st_shndx;
1917                        sym->st_info = ELF_ST_INFO(bind,
1918                            ELF_ST_TYPE(sym->st_info));
1919                        __sym = sym;
1920                }
1921                DBG_CALL(Dbg_syms_new(ofl, __sym, sdp));
1922        }

1924        /*
1925         * Now display GOT debugging information if required.
1926         */
1927        DBG_CALL(Dbg_got_display(ofl, 0, 0,
1928            ld_targ.t_m.m_got_xnumber, ld_targ.t_m.m_got_entsize));

1930        /*
1931         * Update the section headers information. sh_info is
1932         * supposed to contain the offset at which the first
1933         * global symbol resides in the symbol table, while
1934         * sh_link contains the section index of the associated
1935         * string table.
1936         */
1937        if (symtab) {
1938                Shdr    *shdr = ofl->ofl_ossymtab->os_shdr;

1940                shdr->sh_info = symtab_gbl_bndx;
1941                /* LINTED */
1942                shdr->sh_link = (Word)elf_ndxscn(ofl->ofl_osstrtab->os_scn);
1943                if (symshndx)
1944                        ofl->ofl_ossymshndx->os_shdr->sh_link =
1945                            (Word)elf_ndxscn(ofl->ofl_ossymtab->os_scn);

1947                /*
1948                 * Ensure that the expected number of symbols
1949                 * were entered into the right spots:
1950                 *      - Scoped symbols in the right range
1951                 *      - Globals start at the right spot
1952                 *              (correct number of locals entered)
1953                 *      - The table is exactly filled
1954                 *              (correct number of globals entered)
1955                 */
1956                assert((scopesym_bndx + ofl->ofl_scopecnt) == scopesym_ndx);
1957                assert(shdr->sh_info == SYMTAB_LOC_CNT(ofl));
1958                assert((shdr->sh_info + ofl->ofl_globcnt) == symtab_ndx);
1959        }
1960        if (dynsym) {
1961                Shdr    *shdr = ofl->ofl_osdynsym->os_shdr;

1963                shdr->sh_info = DYNSYM_LOC_CNT(ofl);
1964                /* LINTED */
1965                shdr->sh_link = (Word)elf_ndxscn(ofl->ofl_osdynstr->os_scn);

1967                ofl->ofl_oshash->os_shdr->sh_link =
1968                    /* LINTED */
1969                    (Word)elf_ndxscn(ofl->ofl_osdynsym->os_scn);
```

```
1970                if (dynshndx) {
1971                        shdr = ofl->ofl_osdynshndx->os_shdr;
1972                        shdr->sh_link =
1973                            (Word)elf_ndxscn(ofl->ofl_osdynsym->os_scn);
1974                }
1975        }
1976        if (ldynsym) {
1977                Shdr    *shdr = ofl->ofl_osldynsym->os_shdr;

1979                /* ldynsym has no globals, so give index one past the end */
1980                shdr->sh_info = ldynsym_ndx;

1982                /*
1983                 * The ldynsym and dynsym must be adjacent. The
1984                 * idea is that rtld should be able to start with
1985                 * the ldynsym and march straight through the end
1986                 * of dynsym, seeing them as a single symbol table,
1987                 * despite the fact that they are in distinct sections.
1988                 * Ensure that this happened correctly.
1989                 *
1990                 * Note that I use ldynsym_ndx here instead of the
1991                 * computation I used to set the section size
1992                 * (found in ldynsym_cnt). The two will agree, unless
1993                 * we somehow miscounted symbols or failed to insert them
1994                 * all. Using ldynsym_ndx here catches that error in
1995                 * addition to checking for adjacency.
1996                 */
1997                assert(dynsym == (ldynsym + ldynsym_ndx));


2000                /* LINTED */
2001                shdr->sh_link = (Word)elf_ndxscn(ofl->ofl_osdynstr->os_scn);

2003                if (ldynshndx) {
2004                        shdr = ofl->ofl_osldynshndx->os_shdr;
2005                        shdr->sh_link =
2006                            (Word)elf_ndxscn(ofl->ofl_osldynsym->os_scn);
2007                }

2009                /*
2010                 * The presence of .SUNW_ldynsym means that there may be
2011                 * associated sort sections, one for regular symbols
2012                 * and the other for TLS. Each sort section needs the
2013                 * following done:
2014                 *      - Section header link references .SUNW_ldynsym
2015                 *      - Should have received the expected # of items
2016                 *      - Sorted by increasing address
2017                 */
2018                if (ofl->ofl_osdynsymsort) {    /* .SUNW_dynsymsort */
2019                        ofl->ofl_osdynsymsort->os_shdr->sh_link =
2020                            (Word)elf_ndxscn(ofl->ofl_osldynsym->os_scn);
2021                        assert(ofl->ofl_dynsymsortcnt == dynsymsort_ndx);

2023                        if (dynsymsort_ndx > 1) {
2024                                dynsort_compare_syms = ldynsym;
2025                                qsort(dynsymsort, dynsymsort_ndx,
2026                                    sizeof (*dynsymsort), dynsort_compare);
2027                                dynsort_dupwarn(ofl, ldynsym,
2028                                    st_getstrbuf(dynstr),
2029                                    dynsymsort, dynsymsort_ndx,
2030                                    MSG_ORIG(MSG_SCN_DYNSYMSORT));
2031                        }
2032                }
2033                if (ofl->ofl_osdyntlssort) {    /* .SUNW_dyntlssort */
2034                        ofl->ofl_osdyntlssort->os_shdr->sh_link =
2035                            (Word)elf_ndxscn(ofl->ofl_osldynsym->os_scn);
```



```
2036                        assert(ofl->ofl_dyntlssortcnt == dyntlssort_ndx);

2038                        if (dyntlssort_ndx > 1) {
2039                                dynsort_compare_syms = ldynsym;
2040                                qsort(dyntlssort, dyntlssort_ndx,
2041                                    sizeof (*dyntlssort), dynsort_compare);
2042                                dynsort_dupwarn(ofl, ldynsym,
2043                                    st_getstrbuf(dynstr),
2044                                    dyntlssort, dyntlssort_ndx,
2045                                    MSG_ORIG(MSG_SCN_DYNTLSSORT));
2046                        }
2047                }
2048        }

2050        /*
2051         * Used by ld.so.1 only.
2052         */
2053        return (etext);

2055 #undef ADD_TO_DYNSORT
2056 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   88782 Thu Feb  7 01:35:57 2019
new/usr/src/cmd/sgs/packages/common/SUNWonld-README
10346 ld(1) should not reduce symbol visibility of COMDAT symbols when producing
**********************************************************
```

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

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

  General link-editor information can be found:

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

  Comments and Questions:

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

  Warnings:

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

  Patches:

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

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



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

        /net/sunsoftpatch/patches/temporary

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

        http://sunsoftpatch.eng

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

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


  History:

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

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

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

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

---------
Solaris 8
---------
Bugid   Risk Synopsis
================================================================================
4225937 i386 linker emits sparc specific warning messages
4215164 shf_order flag handling broken by fix for 4194028.
4215587 using ld and the -r option on solaris 7 with compiler option -xarch=v9
        causes link errors.
4234657 103627-08 breaks purify 4.2 (plt padding should not be enabled for
        32-bit)
4235241 dbx no longer gets dlclose notification.
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-05  (never released)
        Solaris/SunOS 5.7_x86           patch 106951-05  (never released)
        Solaris/SunOS 5.6_sparc         patch 107733-02  (never released)
        Solaris/SunOS 5.6_x86           patch 107734-02
--------------------------------------------------------------------------------
4248290 inetd dumps core upon bootup - failure in dlclose() logic.
4238071 dlopen() leaks while descriptors under low memory conditions
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.7_sparc         patch 106950-06
        Solaris/SunOS 5.7_x86           patch 106951-06
        Solaris/SunOS 5.6_sparc         patch 107733-03  (never released)
        Solaris/SunOS 5.6_x86           patch 107734-03
--------------------------------------------------------------------------------
4267980 INITFIRST flag of the shard object could be ignored.
--------------------------------------------------------------------------------
All the above changes plus:
    4238973 fix for 4121152 affects linking of Ada objects
    4158744 patch 103627-02 causes core when RPATH has blank entry and
            dlopen/dlclose is used
are incorporated in the following patches:
        Solaris/SunOS 5.5.1_sparc       patch 103627-12  (never released)
        Solaris/SunOS 5.5.1_x86         patch 103628-11
```

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



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

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

231 ----------------------------------
232 Solaris 8 900 (2nd Q-update - s28u2)
233 ----------------------------------
234 Bugid   Risk Synopsis
235 ==============================================================================
236 4324775 non-PIC code & -zcombreloc don't mix very well...
237 4327653 run-time linker should preload tables it will process (madvise)
238 4324324 shared object code can be referenced before .init has fired
239 4321634 .init firing of multiple INITFIRST objects can fail
240 ------------------------------------------------------------------------------
241 All the above changes are incorporated in the following patches:
242         Solaris/SunOS 5.8_sparc         patch 109147-03
243         Solaris/SunOS 5.8_x86           patch 109148-03
244         Solaris/SunOS 5.7_sparc         patch 106950-11
245         Solaris/SunOS 5.7_x86           patch 106951-11
246         Solaris/SunOS 5.6_sparc         patch 107733-08
247         Solaris/SunOS 5.6_x86           patch 107734-08
248 ------------------------------------------------------------------------------
249 4338812 crle(1) omits entries in the directory cache
250 4341496 RFE: provide a static version of /usr/bin/crle
251 4340878 rtld should treat $ORIGIN like LD_LIBRARY_PATH in security issues
252 ------------------------------------------------------------------------------
253 All the above changes are incorporated in the following patches:
254         Solaris/SunOS 5.8_sparc         patch 109147-04
255         Solaris/SunOS 5.8_x86           patch 109148-04
256         Solaris/SunOS 5.7_sparc         patch 106950-12
257         Solaris/SunOS 5.7_x86           patch 106951-12
258 ------------------------------------------------------------------------------
259 4349563 auxiliary filter error handling regression introduced in 4165487
```

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

-----------------------------------
Solaris 8 101 (3rd Q-update - s28u3)
-----------------------------------
Bugid   Risk Synopsis
================================================================================
4346144 link-auditing: plt_tracing fails if LA_SYMB_NOPLTENTER given after
        being bound
4346001 The ld should support mapfile syntax to generate PT_SUNWSTACK segment
4349137 rtld_db: A third fallback method for locating the linkmap
4343417 dladdr interface information inadequate
4343801 RFE: crle(1): provide option for updating configuration files
4346615 ld.so.1 attempting to open a directory gives: No such device
4352233 crle should not honor umask
4352330 LD_PRELOAD cannot use absolute path for privileged program
4357805 RFE: man page for ld(1) does not document all -z or -B options in
        Solaris 8 9/00
4358751 ld.so.1: LD_XXX environ variables and LD_FLAGS should be synchronized.
4358862 link editors should reference "64" symlinks instead of sparcv9 (ia64).
4356879 PLTs could use faster code sequences in some cases
4367118 new fast baplt's fail when traversed twice in threaded application
```



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

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

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

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

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

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



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

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

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

588 ----------------------------------
589 Solaris 9 403 (3nd Q-update - s9u3)
```



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

635 ----------------------------------
636 Solaris 9 803 (4th Q-update - s9u4)
637 ----------------------------------
638 Bugid   Risk Synopsis
639 ================================================================================
640 4730110 ld.so.1 list implementation could scale better
641 4728822 restrict the objects dlsym() searches.
642         PSARC/2002/478 New dlopen(3dl) flag - RTLD_FIRST
643 4714146 crle: 64-bit secure pathname is incorrect.
644 4504895 dlclose() does not remove all objects
645 4698800 Wrong comments in /usr/lib/ld/sparcv9/map.*
646 4745129 dldump is inconsistent with .dynamic processing errors.
647 4753066 LD_SIGNAL isn't very useful in a threaded environment
648         PSARC/2002/569 New dlinfo(3dl) flag - RTLD_DI_SIGNAL
649 4765536 crle: symbolic links can confuse alternative object configuration info
650 4766815 ld -r of object the TLS data fails
651 4770484 elfdump can not handle stripped archive file
652 4770494 The ld command gives improper error message handling broken archive
653 4775738 overwriting output relocation table when 'ld -zignore' is used
654 4778247 elfdump -e of core files fails
655 4779976 elfdump dies on bad relocation entries
```

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

-------------------------------------
Solaris 9 1203 (5th Q-update - s9u5)
-------------------------------------
Bugid   Risk Synopsis
================================================================================
4830584 mmap for the padding region doesn't get freed after dlclose
4831650 ld.so.1 can walk off the end of it's call_init() array...
4831544 ldd using .so modules compiled with FD7 compiler caused a core dump
4834784 Accessing members in a TLS structure causes a core dump in Oracle
4824026 segv when -z combreloc is used with -xlinkopt
4825296 typo in elfdump
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-08
        Solaris/SunOS 5.9_x86           patch T113986-04
        Solaris/SunOS 5.8_sparc         patch T109147-24
        Solaris/SunOS 5.8_x86           patch T109148-24
--------------------------------------------------------------------------------
4470917 Solaris Process Model Unification (link-editor components only)
        PSARC/2002/117 Solaris Process Model Unification
4744411 Bloomberg wants a faster linker.
4811969 64-bit links can be much slower than 32-bit.
4825065 ld(1) should ignore consecutive empty sections.
4838226 unrelocated shared objects may be erroneously collected for init firing
4830889 TLS: testcase coredumps with -xarch=v9 and -g
4845764 filter removal can leave dangling filtee pointer
4811093 apptrace -F libc date core dumps
4826315 Link editors need to be pre- and post- Unified Process Model aware
4868300 interposing on direct bindings can fail
4872634 Large LD_PRELOAD values can cause SEGV of process
```



```
--------------------------------------------------------------------------------
All the above changes are incorporated in the following patches:
        Solaris/SunOS 5.9_sparc         patch T112963-09
        Solaris/SunOS 5.9_x86           patch T113986-05
        Solaris/SunOS 5.8_sparc         patch T109147-25
        Solaris/SunOS 5.8_x86           patch T109148-25
--------------------------------------------------------------------------------

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

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

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

839 ----------
840 Solaris 10
841 ----------
842 Bugid   Risk Synopsis
843 ============================================================================
844 5044797 ld.so.1: secure directory testing is being skipped during filtee
845         processing
846 4963676 Remove remaining static libraries
847 5021541 unnecessary PT_SUNWBSS segment may be created
848 5031495 elfdump complains about bad symbol entries in core files
849 5012172 Need error when creating shared object with .o compiled
850         -xarch=v9 -xcode=abs44
851 4994738 rd_plt_resolution() resolves ebx-relative PLT entries incorrectly
852 5023493 ld -m output with patch 109147-25 missing .o information
853 ----------------------------------------------------------------------------
```



```
854 All the above changes are incorporated in the following patches:
855         Solaris/SunOS 5.9_sparc         patch T112963-15
856         Solaris/SunOS 5.9_x86           patch T113986-11
857         Solaris/SunOS 5.8_sparc         patch T109147-30
858         Solaris/SunOS 5.8_x86           patch T109148-30
859 ----------------------------------------------------------------------------
860 5071614 109147-29 & -30 break the build of on28-patch on Solaris 8 2/04
861 5029830 crle: provide for optional alternative dependencies.
862 5034652 ld.so.1 should save, and print, more error messages
863 5036561 ld.so.1 outputs non-fatal fatal message about auxiliary filter libraries
864 5042713 4866170 broke ld.so's ::setenv
865 5047082 ld can core dump on bad gcc objects
866 5047612 ld.so.1: secure pathname verification is flawed with filter use
867 5047235 elfdump can core dump printing PT_INTERP section
868 4798376 nits in demo code
869 5041446 gelf_update_*() functions inconsistently return NULL or 0
870 5032364 M_ID_TLSBSS and M_ID_UNKNOWN have the same value
871 4707030 Empty LD_PRELOAD_64 doesn't override LD_PRELOAD
872 4968618 symbolic linkage causes core dump
873 5062313 dladdr() can cause deadlock in MT apps.
874 5056867 $ISALIST/$HWCAP expansion should be more flexible.
875 4918303 0@0.so.1 should not use compiler-supplied crt*.o files
876 5058415 whocalls cannot take more than 10 arguments
877 5067518 The fix for 4918303 breaks the build if a new work space is used.
878 ----------------------------------------------------------------------------
879 All the above changes are incorporated in the following patches:
880         Solaris/SunOS 5.9_sparc         patch T112963-16
881         Solaris/SunOS 5.9_x86           patch T113986-12
882         Solaris/SunOS 5.8_sparc         patch T109147-31
883         Solaris/SunOS 5.8_x86           patch T109148-31
884 ----------------------------------------------------------------------------
885 5013759 *file* should report hardware/software capabilities (link-editor
886         components only)
887 5063580 libldstab: file /tmp/posto..: .stab[.index|.sbfocus] found with no
888         matching stri
889 5076838 elfdump(1) is built with a CTF section (the wrong one)
890 5080344 Hardware capabilities are not enforced for a.out
891 5079061 RTLD_DEFAULT can be expensive
892         PSARC/2004/747 New dlsym(3c) Handle - RTLD_PROBE
893 5064973 allow normal relocs against TLS symbols for some sections
894 5085792 LD_XXXX_64 should override LD_XXXX
895 5096272 every executable or library has a .SUNW_dof section
896 5094135 Bloomberg wants a faster ldd.
897 5086352 libld.so.3 should be built with a .SUNW_ctf ELF section, ready for CR
898 5098205 elfdump gives wrong section name for the global offset table
899 5092414 Linker patch 109147-29 makes Broadvison One-To-One server v4.1
900         installation fail
901 5080256 dump(1) doesn't list ELF hardware capabilities
902 5097347 recursive read lock in gelf_getsym()
903 ----------------------------------------------------------------------------
904 All the above changes are incorporated in the following patches:
905         Solaris/SunOS 5.9_sparc         patch T112963-17
906         Solaris/SunOS 5.9_x86           patch T113986-13
907         Solaris/SunOS 5.8_sparc         patch T109147-32
908         Solaris/SunOS 5.8_x86           patch T109148-32
909 ----------------------------------------------------------------------------
910 5106206 ld.so.1 fail to run a Solaris9 program that has libc linked with
911         -z lazyload
912 5102601 ON should deliver a 64-bit operating system for Opteron systems
913         (link-editor components only)
914 6173852 enable link_auditing technology for amd64
915 6174599 linker does not create .eh_frame_hdr sections for eh_frame sections
916         with SHF_LINK_ORDER
917 6175609 amd64 run-time linker has a corrupted note section
918 6175843 amd64 rdb_demo files not installed
919 6182293 ld.so.1 can repeatedly relocate object .plts (RTLD_NOW).
```

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

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



```
 986 6219317 Large page support is needed for mapping executables, libraries and
 987         files (link-editor components only)
 988 6244897 ld.so.1 can't run apps from commandline
 989 6251798 moe(1) returns an internal assertion failure message in some
 990         circumstances
 991 6251722 ld fails silently with exit 1 status when -z ignore passed
 992 6254364 ld won't build libgenunix.so with absolute relocations
 993 6215444 ld.so.1 caches "not there" lazy libraries, foils svc.startd(1M)'s logic
 994 6222525 dlsym(3C) trusts caller(), which may return wrong results with tail call
 995         optimization
 996 6241995 warnings in sgs should be fixed (link-editor components only)
 997 6258834 direct binding availability should be verified at runtime
 998 6260361 lari shouldn't count a.out non-zero undefined entries as interesting
 999 6260780 ldd doesn't recognize LD_NOAUXFLTR
1000 6266261 Add ld(1) -Bnodirect support (D)
1001 6261990 invalid e_flags error could be a little more friendly
1002 6261803 lari(1) should find more events uninteresting (D)
1003 6267352 libld_malloc provides inadequate alignment
1004 6268693 SHN_SUNW_IGNORE symbols should be allowed to be mulitiply defined
1005 6262789 Infosys wants a faster linker
1006 --------------------------------------------------------------------------------
1007 All the above changes are incorporated in the following patches:
1008         Solaris/SunOS 5.10_sparc        patch T117461-03
1009         Solaris/SunOS 5.10_x86          patch T118345-03
1010         Solaris/SunOS 5.9_sparc         patch T112963-21
1011         Solaris/SunOS 5.9_x86           patch T113986-17
1012         Solaris/SunOS 5.8_sparc         patch T109147-36
1013         Solaris/SunOS 5.8_x86           patch T109148-36
1014 --------------------------------------------------------------------------------
1015 6283601 The usr/src/cmd/sgs/packages/common/copyright contains old information
1016         legally problematic
1017 6276905 dlinfo gives inconsistent results (relative vs absolute linkname) (D)
1018         PSARC/2005/357 dlinfo(3c) RTLD_DI_ARGSINFO
1019 6284941 excessive link times with many groups/sections
1020 6280467 dlclose() unmaps shared library before library's _fini() has finished
1021 6291547 ld.so mishandles LD_AUDIT causing security problems.
1022 --------------------------------------------------------------------------------
1023 All the above changes are incorporated in the following patches:
1024         Solaris/SunOS 5.10_sparc        patch T117461-04
1025         Solaris/SunOS 5.10_x86          patch T118345-04
1026         Solaris/SunOS 5.9_sparc         patch T112963-22
1027         Solaris/SunOS 5.9_x86           patch T113986-18
1028         Solaris/SunOS 5.8_sparc         patch T109147-37
1029         Solaris/SunOS 5.8_x86           patch T109148-37
1030 --------------------------------------------------------------------------------
1031 6295971 UNIX98/UNIX03 *vsx* DYNL.hdr/misc/dlfcn/T.dlfcn 14 fails, auxv.h syntax
1032         error
1033 6299525 .init order failure when processing cycles
1034 6273855 gcc and sgs/crle don't get along
1035 6273864 gcc and sgs/libld don't get along
1036 6273875 gcc and sgs/rtld don't get along
1037 6272563 gcc and amd64/krtld/doreloc.c don't get along
1038 6290157 gcc and sgs/librtld_db/rdb_demo don't get along
1039 6301218 Matlab dumps core on startup when running on 112963-22 (D)
1040 --------------------------------------------------------------------------------
1041 All the above changes are incorporated in the following patches:
1042         Solaris/SunOS 5.10_sparc        patch T117461-06
1043         Solaris/SunOS 5.10_x86          patch T118345-08
1044         Solaris/SunOS 5.9_sparc         patch T112963-23
1045         Solaris/SunOS 5.9_x86           patch T113986-19
1046         Solaris/SunOS 5.8_sparc         patch T109147-38
1047         Solaris/SunOS 5.8_x86           patch T109148-38
1048 --------------------------------------------------------------------------------
1049 6314115 Checkpoint refuses to start, crashes on start, after application of
1050         linker patch 112963-22
1051 --------------------------------------------------------------------------------
```

All the above changes are incorporated in the following patches:

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

---

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

---

All the above changes are incorporated in the following patches:

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

---

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

---

All the above changes are incorporated in the following patches:

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

---

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

---

All the above changes are incorporated in the following patches:

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

---

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

Bugid Risk Synopsis

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

---

All the above changes are incorporated in the following patches:

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

---

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

---

All the above changes are incorporated in the following patches:

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

---



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

---

All the above changes are incorporated in the following patches:

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

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

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

---

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

Bugid Risk Synopsis

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

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

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



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

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

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

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

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

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



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

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

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

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

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



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

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

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

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

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



```
1646 3451    archive libraries with no symbols shouldn't require a string table
1647 3616    SHF_GROUP sections should not be discarded via other COMDAT mechanisms
1648 3709    need sloppy relocation for GNU .debug_macro
1649 3722    link-editor is over restrictive of R_AMD64_32 addends
1650 3926    multiple extern map file definitions corrupt symbol table entry
1651 3999    libld extended section handling is broken
1652 4003    dldump() can't deal with extended sections
1653 4227    ld --library-path is translated to -l-path, not -L
1654 4270    ld(1) argument error reporting is still pretty bad
1655 4383    libelf can't write extended sections when ELF_F_LAYOUT
1656 4959    completely discarded merged string sections will corrupt output objects
1657 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 10346   ld(1) should not reduce symbol visibility of COMDAT symbols when
1666         producing relocatable objects
1667 #endif /* ! codereview */
```