```
**********************************************************
   97256 Mon Feb 18 17:20:00 2019
new/usr/src/cmd/sgs/libld/common/syms.c
code review
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 *      Copyright (c) 1988 AT&T
 *        All Rights Reserved
 *
 *
 * Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Symbol table management routines
 */

#define ELF_TARGET_AMD64

#include        <stdio.h>
#include        <string.h>
#include        <debug.h>
#include        <alloca.h>
#include        "msg.h"
#include        "_libld.h"

/*
 * AVL tree comparator function:
 *
 * The primary key is the symbol name hash with a secondary key of the symbol
 * name itself.
 */
int
ld_sym_avl_comp(const void *elem1, const void *elem2)
{
        Sym_avlnode     *sav1 = (Sym_avlnode *)elem1;
        Sym_avlnode     *sav2 = (Sym_avlnode *)elem2;
        int             res;

        res = sav1->sav_hash - sav2->sav_hash;

        if (res < 0)
                return (-1);
        if (res > 0)
                return (1);
```

```
        /*
         * Hash is equal - now compare name
         */
        res = strcmp(sav1->sav_name, sav2->sav_name);
        if (res == 0)
                return (0);
        if (res > 0)
                return (1);
        return (-1);
}
______unchanged_portion_omitted_

/*
 * If an undef symbol exists naming a bound for the output section,
 * turn it into a defined symbol with the correct value.
 *
 * We set an arbitrary 1KB limit on the resulting symbol names.
 */
static void
sym_add_bounds(Ofl_desc *ofl, Os_desc *osp, Word bound)
{
        Sym_desc *bsdp;
        char symn[1024];
        size_t nsz;

        switch (bound) {
        case SDAUX_ID_SECBOUND_START:
                nsz = snprintf(symn, sizeof (symn), "%s%s",
                    MSG_ORIG(MSG_SYM_SECBOUND_START), osp->os_name + 1);
                if (nsz >= sizeof (symn))
                if (nsz > sizeof (symn))
                        return;
                break;
        case SDAUX_ID_SECBOUND_STOP:
                nsz = snprintf(symn, sizeof (symn), "%s%s",
                    MSG_ORIG(MSG_SYM_SECBOUND_STOP), osp->os_name + 1);
                if (nsz >= sizeof (symn))
                if (nsz > sizeof (symn))
                        return;
                break;
        default:
                assert(0);
        }

        if ((bsdp = ld_sym_find(symn, SYM_NOHASH, NULL, ofl)) != NULL) {
                if ((bsdp->sd_shndx != SHN_UNDEF) &&
                    (bsdp->sd_ref == REF_REL_NEED)) {
                        ld_eprintf(ofl, ERR_WARNING, MSG_INTL(MSG_SYM_RESERVE),
                            symn, bsdp->sd_file->ifl_name);
                        return;
                }

                DBG_CALL(Dbg_syms_updated(ofl, bsdp, symn));

                bsdp->sd_aux->sa_symspec = bound;
                bsdp->sd_aux->sa_boundsec = osp;
                bsdp->sd_flags |= FLG_SY_SPECSEC;
                bsdp->sd_ref = REF_REL_NEED;
                bsdp->sd_sym->st_info = ELF_ST_INFO(STB_GLOBAL, STT_NOTYPE);
                bsdp->sd_sym->st_other = STV_PROTECTED;
                bsdp->sd_isc = NULL;
                bsdp->sd_sym->st_size = 0;
                bsdp->sd_sym->st_value = 0;
                bsdp->sd_shndx = bsdp->sd_sym->st_shndx = SHN_ABS;
        }
```

```
}

/*
 * At this point all symbol input processing has been completed, therefore
 * complete the symbol table entries by generating any necessary internal
 * symbols.
 */
uintptr_t
ld_sym_spec(Ofl_desc *ofl)
{
        Sym_desc        *sdp;
        Sg_desc         *sgp;
        Aliste          idx1;

        if (ofl->ofl_flags & FLG_OF_RELOBJ)
                return (1);

        DBG_CALL(Dbg_syms_spec_title(ofl->ofl_lml));

        /*
         * For each section in the output file, look for symbols named for the
         * __start/__stop patterns.  If references exist, flesh the symbols to
         * be defined.
         *
         * The symbols are given values at the same time as the other special
         * the symbols are given values at the same time as the other special
         * symbols.
         */
        for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
                Os_desc *osp;
                Aliste idx2;

                for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
                        sym_add_bounds(ofl, osp, SDAUX_ID_SECBOUND_START);
                        sym_add_bounds(ofl, osp, SDAUX_ID_SECBOUND_STOP);
                }
        }

        if (sym_add_spec(MSG_ORIG(MSG_SYM_ETEXT), MSG_ORIG(MSG_SYM_ETEXT_U),
            SDAUX_ID_ETEXT, 0, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
            ofl) == S_ERROR)
                return (S_ERROR);
        if (sym_add_spec(MSG_ORIG(MSG_SYM_EDATA), MSG_ORIG(MSG_SYM_EDATA_U),
            SDAUX_ID_EDATA, 0, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
            ofl) == S_ERROR)
                return (S_ERROR);
        if (sym_add_spec(MSG_ORIG(MSG_SYM_END), MSG_ORIG(MSG_SYM_END_U),
            SDAUX_ID_END, FLG_SY_DYNSORT, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
            ofl) == S_ERROR)
                return (S_ERROR);
        if (sym_add_spec(MSG_ORIG(MSG_SYM_L_END), MSG_ORIG(MSG_SYM_L_END_U),
            SDAUX_ID_END, 0, FLG_SY_HIDDEN, ofl) == S_ERROR)
                return (S_ERROR);
        if (sym_add_spec(MSG_ORIG(MSG_SYM_L_START), MSG_ORIG(MSG_SYM_L_START_U),
            SDAUX_ID_START, 0, FLG_SY_HIDDEN, ofl) == S_ERROR)
                return (S_ERROR);

        /*
         * Historically we've always produced a _DYNAMIC symbol, even for
         * static executables (in which case its value will be 0).
         */
        if (sym_add_spec(MSG_ORIG(MSG_SYM_DYNAMIC), MSG_ORIG(MSG_SYM_DYNAMIC_U),
            SDAUX_ID_DYN, FLG_SY_DYNSORT, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
            ofl) == S_ERROR)
                return (S_ERROR);
```

```
        if (OFL_ALLOW_DYNSYM(ofl))
                if (sym_add_spec(MSG_ORIG(MSG_SYM_PLKTBL),
                    MSG_ORIG(MSG_SYM_PLKTBL_U), SDAUX_ID_PLT,
                    FLG_SY_DYNSORT, (FLG_SY_DEFAULT | FLG_SY_EXPDEF),
                    ofl) == S_ERROR)
                        return (S_ERROR);

        /*
         * A GOT reference will be accompanied by the associated GOT symbol.
         * Make sure it gets assigned the appropriate special attributes.
         */
        if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_GOFTBL_U),
            SYM_NOHASH, NULL, ofl)) != NULL) && (sdp->sd_ref != REF_DYN_SEEN)) {
                if (sym_add_spec(MSG_ORIG(MSG_SYM_GOFTBL),
                    MSG_ORIG(MSG_SYM_GOFTBL_U), SDAUX_ID_GOT, FLG_SY_DYNSORT,
                    (FLG_SY_DEFAULT | FLG_SY_EXPDEF), ofl) == S_ERROR)
                        return (S_ERROR);
        }

        return (1);
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
    1192 Mon Feb 18 17:20:00 2019
new/usr/src/test/elf-tests/tests/linker-sets/in-use-check.sh
code review
**********************************************************
#!/usr/bin/ksh
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2018, Richard Lowe.
#

# Test that existing definitions of the start/stop symbols are reported
# as conflicting with internal symbols.
# Test that a simple use of linker-sets, tat is, automatically generated start
# and end symbols for sections can be generated and used.

tmpdir=/tmp/test.$$
mkdir $tmpdir
cd $tmpdir

cleanup() {
    cd /
    rm -fr $tmpdir
}
______unchanged_portion_omitted_
```

****
    1398 Mon Feb 18 17:20:01 2019
new/usr/src/test/elf-tests/tests/linker-sets/simple.sh
code review
****

```
#!/usr/bin/ksh
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2018, Richard Lowe.
#

# Test that a simple use of linker-sets, that is, automatically generated start
# Test that a simple use of linker-sets, tat is, automatically generated start
# and end symbols for sections can be generated and used.

if [[ -z $ELF_TESTS ]]; then
    print -u2 "Don’t know where the test data is rooted";
    exit 1;
fi

tmpdir=/tmp/test.$$
mkdir $tmpdir
cd $tmpdir

cleanup() {
    cd /
    rm -fr $tmpdir
}
```

______***unchanged_portion_omitted_***