```
**********************************************************
    7014 Sat Jan  5 00:24:56 2013
new/usr/src/cmd/sgs/libelf/common/getarsym.c
3451 archive libraries with no symbols shouldn't require a string table
**********************************************************
/*
 * 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) 1990, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*      Copyright (c) 1988 AT&T */
/*        All Rights Reserved   */

#include <stdlib.h>
#include <errno.h>
#include <libelf.h>
#include "decl.h"
#include "msg.h"


/*
 * Convert archive symbol table to memory format
 *
 * This takes a pointer to file's archive symbol table, alignment
 * unconstrained.  Returns null terminated vector of Elf_Arsym
 * structures. Elf_Arsym uses size_t to represent offsets, which
 * will be 32-bit in 32-bit versions, and 64-bits otherwise.
 *
 * There are two forms of archive symbol table, the original 32-bit
 * form, and a 64-bit form originally found in IRIX64. The two formats
 * differ only in the width of the integer word:
 *
 *              # offsets       4/8-byte word
 *              offset[0...]    4/8-byte word each
 *              strings         null-terminated, for offset[x]
 *
 * By default, the 64-bit form is only used when the archive exceeds
 * the limits of 32-bits (4GB) in size. However, this is not required,
 * and the ar -S option can be used to create a 64-bit symbol table in
 * an archive that is under 4GB.
 *
 * Both 32 and 64-bit versions of libelf can read the 32-bit format
 * without loss of information. Similarly, a 64-bit version of libelf
 * will have no problem reading a 64-bit symbol table. This leaves the
 * case where a 32-bit libelf reads a 64-bit symbol table, which requires
 * some explanation. The offsets in a 64-bit symbol table will have zeros
```

```
 * in the upper half of the words until the size of the archive exceeds 4GB.
 * However, 32-bit libelf is unable to read any files larger than 2GB
 * (see comments in update.c). As such, any archive that the 32-bit version
 * of this code will encounter will be under 4GB in size. The upper 4
 * bytes of each word will be zero, and can be safely ignored.
 */


/*
 * Offsets in archive headers are written in MSB (large endian) order
 * on all platforms, regardless of native byte order. These macros read
 * 4 and 8 byte values from unaligned memory.
 *
 * note:
 * -    The get8() macro for 32-bit code can ignore the first 4 bytes of
 *      of the word, because they are known to be 0.
 *
 * -    The inner most value in these macros is cast to an unsigned integer
 *      of the final width in order to prevent the C comilier from doing
 *      unwanted sign extension when the topmost bit of a byte is set.
 */
#define get4(p) ((((((uint32_t)p[0]<<8)+p[1])<<8)+p[2])<<8)+p[3])

#ifdef _LP64
#define get8(p) (((((((((((((((uint64_t)p[0]<<8)+p[1])<<8)+p[2])<<8)+   \
    p[3])<<8)+p[4])<<8)+p[5])<<8)+p[6])<<8)+p[7])
#else
#define get8(p) ((((((uint64_t)p[4]<<8)+p[5])<<8)+p[6])<<8)+p[7])
#endif


static Elf_Void *
arsym(Byte *off, size_t sz, size_t *e, int is64)
{
        char            *endstr = (char *)off + sz;
        register char   *str;
        Byte            *endoff;
        Elf_Void        *oas;
        size_t          eltsize = is64 ? 8 : 4;

        {
                register size_t n;

                if (is64) {
                        if (sz < 8 || (sz - 8) / 8 < (n = get8(off))) {
                                _elf_seterr(EFMT_ARSYMSZ, 0);
                                return (NULL);
                                return (0);
                        }
                } else {
                        if (sz < 4 || (sz - 4) / 4 < (n = get4(off))) {
                                _elf_seterr(EFMT_ARSYMSZ, 0);
                                return (NULL);
                                return (0);
                        }
                }
                off += eltsize;
                endoff = off + n * eltsize;

                /*
                 * If there are symbols in the symbol table, a
                 * string table must be present and NULL terminated.
                 *
                 * The format dictates that the string table must always be
                 * present, however in the case of an archive containing no
                 * symbols GNU ar will not create one.  We are permissive for
                 * the sake of compatibility.
                 * string table must be present, null terminated
                 */
                if ((n > 0) && (((str = (char *)endoff) >= endstr) ||
                    (*(endstr - 1) != '\0'))) {

                if (((str = (char *)endoff) >= endstr) ||
                    (*(endstr - 1) != '\0')) {
                        _elf_seterr(EFMT_ARSYM, 0);
                        return (NULL);
                        return (0);
                }

                /*
                 * There is always at least one entry returned if a symtab
                 * exists since the table's last entry is an artificial one
                 * with a NULL as_name, but is included in the count.
                 *
#endif /* ! codereview */
                 * overflow can occur here, but not likely
                 */

                *e = n + 1;
                if ((oas = calloc(n + 1, sizeof (Elf_Arsym))) == NULL) {
                n = sizeof (Elf_Arsym) * (n + 1);
                if ((oas = malloc(n)) == 0) {
                        _elf_seterr(EMEM_ARSYM, errno);
                        return (NULL);
                        return (0);
                }
        }
        {
                register Elf_Arsym      *as = (Elf_Arsym *)oas;

                while (off < endoff) {
                        if (str >= endstr) {
                                _elf_seterr(EFMT_ARSYMSTR, 0);
                                free(oas);
                                return (NULL);
                                return (0);
                        }
                        if (is64)
                                as->as_off = get8(off);
                        else
                                as->as_off = get4(off);
                        as->as_name = str;
                        as->as_hash = elf_hash(str);
                        ++as;
                        off += eltsize;
                        while (*str++ != '\0')
                                /* LINTED */
                                ;
                }
                as->as_name = NULL;
                as->as_name = 0;
                as->as_off = 0;
                as->as_hash = ~(unsigned long)0L;
        }
        return (oas);
}
```

______*unchanged_portion_omitted_*