```
**********************************************************
   15100 Tue Oct 23 15:45:56 2012
new/usr/src/cmd/file/elf_read.c
3299 file should only care about object capabilities
3300 file should care about all object capabilities
**********************************************************
```

```
/*
 * 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) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */


/*      Copyright (c) 1987, 1988 Microsoft Corporation  */
/*        All Rights Reserved   */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * ELF files can exceed 2GB in size. A standard 32-bit program
 * like 'file' cannot read past 2GB, and will be unable to see
 * the ELF section headers that typically are at the end of the
 * object. The simplest solution to this problem would be to make
 * the 'file' command a 64-bit application. However, as a matter of
 * policy, we do not want to require this. A simple command like
 * 'file' should not carry such a requirement, especially as we
 * support 32-bit only hardware.
 *
 * An alternative solution is to build this code as 32-bit
 * large file aware. The usual way to do this is to define a pair
 * of preprocessor definitions:
 *
 *      _LARGEFILE64_SOURCE
 *              Map standard I/O routines to their largefile aware versions.
 *
 *      _FILE_OFFSET_BITS=64
 *              Map off_t to off64_t
 *
 * The problem with this solution is that libelf is not large file capable,
 * and the libelf header file will prevent compilation if
 * _FILE_OFFSET_BITS is set to 64.
 *
 * So, the solution used in this code is to define _LARGEFILE64_SOURCE
 * to get access to the 64-bit APIs, not to define _FILE_OFFSET_BITS, and to
```



```
 * use our own types in place of off_t, and size_t. We read all the file
 * data directly using pread64(), and avoid the use of libelf for anything
 * other than the xlate functionality.
 */
#define _LARGEFILE64_SOURCE
#define FILE_ELF_OFF_T  off64_t
#define FILE_ELF_SIZE_T uint64_t

#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <libelf.h>
#include <stdlib.h>
#include <limits.h>
#include <locale.h>
#include <string.h>
#include <errno.h>
#include <procfs.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/elf.h>
#include <elfcap.h>
#include "file.h"
#include "elf_read.h"

extern const char *File;

static int get_class(void);
static int get_version(void);
static int get_format(void);
static int process_shdr(Elf_Info *);
static int process_phdr(Elf_Info *);
static int file_xlatetom(Elf_Type, char *);
static int xlatetom_nhdr(Elf_Nhdr *);
static int get_phdr(Elf_Info *, int);
static int get_shdr(Elf_Info *, int);

static Elf_Ehdr EI_Ehdr;                /* Elf_Ehdr to be stored */
static Elf_Word EI_Ehdr_shnum;          /* # section headers */
static Elf_Word EI_Ehdr_phnum;          /* # program headers */
static Elf_Word EI_Ehdr_shstrndx;       /* Index of section hdr string table */
static Elf_Shdr EI_Shdr;                /* recent Elf_Shdr to be stored */
static Elf_Phdr EI_Phdr;                /* recent Elf_Phdr to be stored */


static int
get_class(void)
{
        return (EI_Ehdr.e_ident[EI_CLASS]);
}
______unchanged_portion_omitted_

/*
 * process_shdr:        Read Section Headers to attempt to get HW/SW
 *                      capabilities by looking at the SUNW_cap
 *                      section and set string in Elf_Info.
 *                      Also look for symbol tables and debug
 *                      information sections. Set the "stripped" field
 *                      in Elf_Info with corresponding flags.
 */
static int
process_shdr(Elf_Info *EI)
{
        int             capn, mac;
```

```
419         int             i, j, idx;
420         FILE_ELF_OFF_T  cap_off;
421         FILE_ELF_SIZE_T csize;
422         char            *section_name;
423         Elf_Cap         Chdr;
424         Elf_Shdr        *shdr = &EI_Shdr;


427         csize = sizeof (Elf_Cap);
428         mac = EI_Ehdr.e_machine;

430         /* if there are no sections, return success anyway */
431         if (EI_Ehdr.e_shoff == 0 && EI_Ehdr_shnum == 0)
432                 return (ELF_READ_OKAY);

434         /* read section names from String Section */
435         if (get_shdr(EI, EI_Ehdr_shstrndx) == ELF_READ_FAIL)
436                 return (ELF_READ_FAIL);

438         if ((section_name = malloc(shdr->sh_size)) == NULL)
439                 return (ELF_READ_FAIL);

441         if (pread64(EI->elffd, section_name, shdr->sh_size, shdr->sh_offset)
442             != shdr->sh_size)
443                 return (ELF_READ_FAIL);

445         /* read all the sections and process them */
446         for (idx = 1, i = 0; i < EI_Ehdr_shnum; idx++, i++) {
447                 char *str;

449                 if (get_shdr(EI, i) == ELF_READ_FAIL)
450                         return (ELF_READ_FAIL);

452                 if (shdr->sh_type == SHT_NULL) {
453                         idx--;
454                         continue;
455                 }

457                 cap_off = shdr->sh_offset;
458                 if (shdr->sh_type == SHT_SUNW_cap) {
459                         char capstr[128];

461 #endif /* ! codereview */
462                         if (shdr->sh_size == 0 || shdr->sh_entsize == 0) {
463                                 (void) fprintf(stderr, ELF_ERR_ELFCAP1,
464                                     File, EI->file);
465                                 return (ELF_READ_FAIL);
466                         }
467                         capn = (shdr->sh_size / shdr->sh_entsize);
468                         for (j = 0; j < capn; j++) {
469                                 /*
470                                  * read cap and xlate the values
471                                  */
472                                 if (pread64(EI->elffd, &Chdr, csize, cap_off)
473                                     != csize ||
474                                     file_xlatetom(ELF_T_CAP, (char *)&Chdr)
475                                     == 0) {
476                                         (void) fprintf(stderr, ELF_ERR_ELFCAP2,
477                                             File, EI->file);
478                                         return (ELF_READ_FAIL);
479                                 }

461                                 if (Chdr.c_tag != CA_SUNW_NULL) {
462                                         (void) elfcap_tag_to_str(
463                                             ELFCAP_STYLE_UC, Chdr.c_tag,
464                                             Chdr.c_un.c_val, EI->cap_str,
```



```
465                                             sizeof (EI->cap_str),
466                                             ELFCAP_FMT_SNGSPACE, mac);
467                                 }
481                                 cap_off += csize;

483                                 /*
484                                  * Each capatibility group is terminated with
485                                  * CA_SUNW_NULL.  Groups other than the first
486                                  * represent symbol capabilities, and aren't
487                                  * interesting here.
488                                  */
489                                 if (Chdr.c_tag == CA_SUNW_NULL)
490                                         break;

492                                 (void) elfcap_tag_to_str(ELFCAP_STYLE_UC,
493                                     Chdr.c_tag, Chdr.c_un.c_val, capstr,
494                                     sizeof (capstr), ELFCAP_FMT_SNGSPACE,
495                                     mac);

497                                 if ((*EI->cap_str != '\0') && (*capstr != '\0'))
498                                         (void) strlcat(EI->cap_str, " ",
499                                             sizeof (EI->cap_str));

501                                 (void) strlcat(EI->cap_str, capstr,
502                                     sizeof (EI->cap_str));
503 #endif /* ! codereview */
504                                 }
505                 }

507                 /*
508                  * Definition time:
509                  *      - "not stripped" means that an executable file
510                  *      contains a Symbol Table (.symtab)
511                  *      - "stripped" means that an executable file
512                  *      does not contain a Symbol Table.
513                  * When strip -l or strip -x is run, it strips the
514                  * debugging information (.line section name (strip -l),
515                  * .line, .debug*, .stabs*, .dwarf* section names
516                  * and SHT_SUNW_DEBUGSTR and SHT_SUNW_DEBUG
517                  * section types (strip -x), however the Symbol
518                  * Table will still be present.
519                  * Therefore, if
520                  *      - No Symbol Table present, then report
521                  *              "stripped"
522                  *      - Symbol Table present with debugging
523                  *      information (line number or debug section names,
524                  *      or SHT_SUNW_DEBUGSTR or SHT_SUNW_DEBUG section
525                  *      types) then report:
526                  *              "not stripped"
527                  *      - Symbol Table present with no debugging
528                  *      information (line number or debug section names,
529                  *      or SHT_SUNW_DEBUGSTR or SHT_SUNW_DEBUG section
530                  *      types) then report:
531                  *              "not stripped, no debugging information
532                  *              available"
533                  */
534                 if ((EI->stripped & E_NOSTRIP) == E_NOSTRIP)
535                         continue;

537                 if (!(EI->stripped & E_SYMTAB) &&
538                     (shdr->sh_type == SHT_SYMTAB)) {
539                         EI->stripped |= E_SYMTAB;
540                         continue;
541                 }

543                 str = &section_name[shdr->sh_name];
```

```
545                 if (!(EI->stripped & E_DBGINF) &&
546                     ((shdr->sh_type == SHT_SUNW_DEBUG) ||
547                     (shdr->sh_type == SHT_SUNW_DEBUGSTR) ||
548                     (is_in_list(str)))) {
549                         EI->stripped |= E_DBGINF;
550                 }
551         }
552         free(section_name);

554         return (ELF_READ_OKAY);
555 }
```