```
**********************************************************
   43331 Sat Apr 26 08:41:44 2014
new/usr/src/cmd/file/file.c
4732 /usr/bin/file should provide -b option  for compatiblity with GNU/BSD file
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
Reviewed by: Serghei Samsi <sscdvp@gmail.com>
Reviewed by: Alexander Pyhalov <alp@rsu.ru>
Reviewed by: Garrett D’Amore <garrett@damore.org>
**********************************************************
```

```
/*
 * 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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#define _LARGEFILE64_SOURCE

/* Get definitions for the relocation types supported. */
#define ELF_TARGET_ALL

#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <stdio.h>
#include <libelf.h>
#include <stdlib.h>
#include <limits.h>
#include <locale.h>
#include <wctype.h>
#include <string.h>
#include <errno.h>
#include <door.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/mkdev.h>
#include <sys/stat.h>
#include <sys/elf.h>
#include <procfs.h>
#include <sys/core.h>
```

```
#include <sys/dumphdr.h>
#include <netinet/in.h>
#include <gelf.h>
#include <elfcap.h>
#include <sgsrtcid.h>
#include "file.h"
#include "elf_read.h"

/*
 *      Misc
 */

#define FBSZ            512
#define MLIST_SZ        12

/*
 * The 0x8FCA0102 magic string was used in crash dumps generated by releases
 * prior to Solaris 7.
 */
#define OLD_DUMP_MAGIC  0x8FCA0102

#if defined(__sparc)
#define NATIVE_ISA      "SPARC"
#define OTHER_ISA       "Intel"
#else
#define NATIVE_ISA      "Intel"
#define OTHER_ISA       "SPARC"
#endif

/* Assembly language comment char */
#ifdef pdp11
#define ASCOMCHAR ’/’
#else
#define ASCOMCHAR ’!’
#endif

#pragma align   16(fbuf)
static char     fbuf[FBSZ];

/*
 * Magic file variables
 */
static intmax_t maxmagicoffset;
static intmax_t tmpmax;
static char     *magicbuf;

static char     *dfile;
static char     *troff[] = {    /* new troff intermediate lang */
                "x", "T", "res", "init", "font", "202", "V0", "p1", 0};

static char     *fort[] = {                     /* FORTRAN */
                "function", "subroutine", "common", "dimension", "block",
                "integer", "real", "data", "double",
                "FUNCTION", "SUBROUTINE", "COMMON", "DIMENSION", "BLOCK",
                "INTEGER", "REAL", "DATA", "DOUBLE", 0};

static char     *asc[] = {              /* Assembler Commands */
                "sys", "mov", "tst", "clr", "jmp", "cmp", "set", "inc",
                "dec", 0};

static char     *c[] = {                        /* C Language */
                "int", "char", "float", "double", "short", "long", "unsigned",
                "register", "static", "struct", "extern", 0};

static char     *as[] = {       /* Assembler Pseudo Ops, prepended with ’.’ */
                "globl", "global", "ident", "file", "byte", "even",
                "text", "data", "bss", "comm", 0};

/*
 * The line and debug section names are used by the strip command.
 * Any changes in the strip implementation need to be reflected here.
 */
static char     *debug_sections[] = { /* Debug sections in a ELF file */
                ".debug", ".stab", ".dwarf", ".line", NULL};

/* start for MB env */
static wchar_t  wchar;
static int      length;
static int      IS_ascii;
static int      Max;
/* end for MB env */
static int      i;      /* global index into first 'fbsz' bytes of file */
static int      fbsz;
static int      ifd = -1;
static int      elffd = -1;
static int      tret;
static int      hflg;
static int      dflg;
static int      mflg;
static int      M_flg;
static int      iflg;
static struct stat64    mbuf;

static char     **mlist1;       /* 1st ordered list of magic files */
static char     **mlist2;       /* 2nd ordered list of magic files */
static size_t   mlist1_sz;      /* number of ptrs allocated for mlist1 */
static size_t   mlist2_sz;      /* number of ptrs allocated for mlist2 */
static char     **mlist1p;      /* next entry in mlist1 */
static char     **mlist2p;      /* next entry in mlist2 */

static ssize_t  mread;

static void ar_coff_or_aout(int ifd);
static int type(char *file);
static int def_position_tests(char *file);
static void def_context_tests(void);
static int troffint(char *bp, int n);
static int lookup(char **tab);
static int ccom(void);
static int ascom(void);
static int sccs(void);
static int english(char *bp, int n);
static int shellscript(char buf[], struct stat64 *sb);
static int elf_check(char *file);
static int get_door_target(char *, char *, size_t);
static int zipfile(char *, int);
static int is_crash_dump(const char *, int);
static void print_dumphdr(const int, const dumphdr_t *, uint32_t (*)(uint32_t),
    const char *);
static uint32_t swap_uint32(uint32_t);
static uint32_t return_uint32(uint32_t);
static void usage(void);
static void default_magic(void);
static void add_to_mlist(char *, int);
static void fd_cleanup(void);
static int is_rtld_config(void);

/* from elf_read.c */
int elf_read32(int elffd, Elf_Info *EInfo);
int elf_read64(int elffd, Elf_Info *EInfo);

#ifdef XPG4
```

```
        /* SUSv3 requires a single <space> after the colon */
#define prf(x)  (void) printf("%s: ", x);
#else   /* !XPG4 */
#define prf(x)  (void) printf("%s:%s", x, (int)strlen(x) > 6 ? "\t" : "\t\t");
#endif  /* XPG4 */

/*
 * Static program identifier - used to prevent localization of the name "file"
 * within individual error messages.
 */
const char *File = "file";

int
main(int argc, char **argv)
{
        char    *p;
        int     ch;
        FILE    *fl;
        int     bflg = 0;
#endif /* ! codereview */
        int     cflg = 0;
        int     eflg = 0;
        int     fflg = 0;
        char    *ap = NULL;
        int     pathlen;
        char    **filep;

        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
#define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
#endif
        (void) textdomain(TEXT_DOMAIN);

        while ((ch = getopt(argc, argv, "M:bcdf:him:")) != EOF) {
        while ((ch = getopt(argc, argv, "M:cdf:him:")) != EOF) {
                switch (ch) {

                case 'M':
                        add_to_mlist(optarg, !dflg);
                        M_flg++;
                        break;

                case 'b':
                        bflg++;
                        break;

#endif /* ! codereview */
                case 'c':
                        cflg++;
                        break;

                case 'd':
                        if (!dflg) {
                                default_magic();
                                add_to_mlist(dfile, 0);
                                dflg++;
                        }
                        break;

                case 'f':
                        fflg++;
                        errno = 0;
                        if ((fl = fopen(optarg, "r")) == NULL) {
                                int err = errno;
                                (void) fprintf(stderr, gettext("%s: cannot "
                                    "open file %s: %s\n"), File, optarg,
                                    err ? strerror(err) : "");
                                usage();
                        }
                        pathlen = pathconf("/", _PC_PATH_MAX);
                        if (pathlen == -1) {
                                int err = errno;
                                (void) fprintf(stderr, gettext("%s: cannot "
                                    "determine maximum path length: %s\n"),
                                    File, strerror(err));
                                exit(1);
                        }
                        pathlen += 2; /* for null and newline in fgets */
                        if ((ap = malloc(pathlen * sizeof (char))) == NULL) {
                                int err = errno;
                                (void) fprintf(stderr, gettext("%s: malloc "
                                    "failed: %s\n"), File, strerror(err));
                                exit(2);
                        }
                        break;

                case 'h':
                        hflg++;
                        break;

                case 'i':
                        iflg++;
                        break;

                case 'm':
                        add_to_mlist(optarg, !dflg);
                        mflg++;
                        break;

                case '?':
                        eflg++;
                        break;
                }
        }
        if (!cflg && !fflg && (eflg || optind == argc))
                usage();
        if (iflg && (dflg || mflg || M_flg)) {
                usage();
        }
        if ((iflg && cflg) || (cflg && bflg)) {
        if (iflg && cflg) {
                usage();
        }

        if (!dflg && !mflg && !M_flg && !iflg) {
        /* no -d, -m, nor -M option; also -i option doesn't need magic  */
                default_magic();
                if (f_mkmtab(dfile, cflg, 0) == -1) {
                        exit(2);
                }
        }

        else if (mflg && !M_flg && !dflg) {
        /* -m specified without -d nor -M */

#ifdef XPG4     /* For SUSv3 only */

                /*
                 * The default position-dependent magic file tests
                 * in /etc/magic will follow all the -m magic tests.
                 */
```

```
                for (filep = mlist1; filep < mlist1p; filep++) {
                        if (f_mkmtab(*filep, cflg, 1) == -1) {
                                exit(2);
                        }
                }
                default_magic();
                if (f_mkmtab(dfile, cflg, 0) == -1) {
                        exit(2);
                }
#else   /* !XPG4 */
                /*
                 * Retain Solaris file behavior for -m before SUSv3,
                 * when the new -d and -M options are not specified.
                 * Use the -m file specified in place of the default
                 * /etc/magic file.  Solaris file will
                 * now allow more than one magic file to be specified
                 * with multiple -m options, for consistency with
                 * other behavior.
                 *
                 * Put the magic table(s) specified by -m into
                 * the second magic table instead of the first
                 * (as indicated by the last argument to f_mkmtab()),
                 * since they replace the /etc/magic tests and
                 * must be executed alongside the default
                 * position-sensitive tests.
                 */

                for (filep = mlist1; filep < mlist1p; filep++) {
                        if (f_mkmtab(*filep, cflg, 0) == -1) {
                                exit(2);
                        }
                }
#endif /* XPG4 */
        } else {
                /*
                 * For any other combination of -d, -m, and -M,
                 * use the magic files in command-line order.
                 * Store the entries from the two separate lists of magic
                 * files, if any, into two separate magic file tables.
                 * mlist1: magic tests executed before default magic tests
                 * mlist2: default magic tests and after
                 */
                for (filep = mlist1; filep && (filep < mlist1p); filep++) {
                        if (f_mkmtab(*filep, cflg, 1) == -1) {
                                exit(2);
                        }
                }
                for (filep = mlist2; filep && (filep < mlist2p); filep++) {
                        if (f_mkmtab(*filep, cflg, 0) == -1) {
                                exit(2);
                        }
                }
        }

        /* Initialize the magic file variables; check both magic tables */
        tmpmax = f_getmaxoffset(1);
        maxmagicoffset = f_getmaxoffset(0);
        if (maxmagicoffset < tmpmax) {
                maxmagicoffset = tmpmax;
        }
        if (maxmagicoffset < (intmax_t)FBSZ)
                maxmagicoffset = (intmax_t)FBSZ;
        if ((magicbuf = malloc(maxmagicoffset)) == NULL) {
                int err = errno;
                (void) fprintf(stderr, gettext("%s: malloc failed: %s\n"),
                    File, strerror(err));
                exit(2);
        }

        if (cflg) {
                f_prtmtab();
                if (ferror(stdout) != 0) {
                        (void) fprintf(stderr, gettext("%s: error writing to "
                            "stdout\n"), File);
                        exit(1);
                }
                if (fclose(stdout) != 0) {
                        int err = errno;
                        (void) fprintf(stderr, gettext("%s: fclose "
                            "failed: %s\n"), File, strerror(err));
                        exit(1);
                }
                exit(0);
        }

        for (; fflg || optind < argc; optind += !fflg) {
                register int    l;

                if (fflg) {
                        if ((p = fgets(ap, pathlen, fl)) == NULL) {
                                fflg = 0;
                                optind--;
                                continue;
                        }
                        l = strlen(p);
                        if (l > 0)
                                p[l - 1] = '\0';
                } else
                        p = argv[optind];

                if (!bflg)
#endif /* ! codereview */
                        prf(p);         /* print "file_name:<tab>" */

                if (type(p))
                        tret = 1;
        }
        if (ap != NULL)
                free(ap);
        if (tret != 0)
                exit(tret);

        if (ferror(stdout) != 0) {
                (void) fprintf(stderr, gettext("%s: error writing to "
                    "stdout\n"), File);
                exit(1);
        }
        if (fclose(stdout) != 0) {
                int err = errno;
                (void) fprintf(stderr, gettext("%s: fclose failed: %s\n"),
                    File, strerror(err));
                exit(1);
        }
        return (0);
}

static int
type(char *file)
{
        int     cc;
        char    buf[BUFSIZ];
        int     (*statf)() = hflg ? lstat64 : stat64;
```

```
        i = 0;          /* reset index to beginning of file */
        ifd = -1;
        if ((*statf)(file, &mbuf) < 0) {
                if (statf == lstat64 || lstat64(file, &mbuf) < 0) {
                        int err = errno;
                        (void) printf(gettext("cannot open: %s\n"),
                            strerror(err));
                        return (0);             /* POSIX.2 */
                }
        }
        switch (mbuf.st_mode & S_IFMT) {
        case S_IFREG:
                if (iflg) {
                        (void) printf(gettext("regular file\n"));
                        return (0);
                }
                break;
        case S_IFCHR:
                (void) printf(gettext("character"));
                goto spcl;

        case S_IFDIR:
                (void) printf(gettext("directory\n"));
                return (0);

        case S_IFIFO:
                (void) printf(gettext("fifo\n"));
                return (0);

        case S_IFLNK:
                if ((cc = readlink(file, buf, BUFSIZ)) < 0) {
                        int err = errno;
                        (void) printf(gettext("readlink error: %s\n"),
                            strerror(err));
                        return (1);
                }
                buf[cc] = '\0';
                (void) printf(gettext("symbolic link to %s\n"), buf);
                return (0);

        case S_IFBLK:
                (void) printf(gettext("block"));
                                        /* major and minor, see sys/mkdev.h */
spcl:
                (void) printf(gettext(" special (%d/%d)\n"),
                    major(mbuf.st_rdev), minor(mbuf.st_rdev));
                return (0);

        case S_IFSOCK:
                (void) printf("socket\n");
                /* FIXME, should open and try to getsockname. */
                return (0);

        case S_IFDOOR:
                if (get_door_target(file, buf, sizeof (buf)) == 0)
                        (void) printf(gettext("door to %s\n"), buf);
                else
                        (void) printf(gettext("door\n"));
                return (0);

        }

        if (elf_version(EV_CURRENT) == EV_NONE) {
                (void) printf(gettext("libelf is out of date\n"));
                return (1);
```

```
	}

	ifd = open64(file, O_RDONLY);
	if (ifd < 0) {
		int err = errno;
		(void) printf(gettext("cannot open: %s\n"), strerror(err));
		return (0);			/* POSIX.2 */
	}

	/* need another fd for elf, since we might want to read the file too */
	elffd = open64(file, O_RDONLY);
	if (elffd < 0) {
		int err = errno;
		(void) printf(gettext("cannot open: %s\n"), strerror(err));
		(void) close(ifd);
		ifd = -1;
		return (0);			/* POSIX.2 */
	}
	if ((fbsz = read(ifd, fbuf, FBSZ)) == -1) {
		int err = errno;
		(void) printf(gettext("cannot read: %s\n"), strerror(err));
		(void) close(ifd);
		ifd = -1;
		return (0);			/* POSIX.2 */
	}
	if (fbsz == 0) {
		(void) printf(gettext("empty file\n"));
		fd_cleanup();
		return (0);
	}

	/*
	 * First try user-specified position-dependent magic tests, if any,
	 * which need to execute before the default tests.
	 */
	if ((mread = pread(ifd, (void*)magicbuf, (size_t)maxmagicoffset,
	    (off_t)0)) == -1) {
		int err = errno;
		(void) printf(gettext("cannot read: %s\n"), strerror(err));
		fd_cleanup();
		return (0);
	}

	/*
	 * ChecK against Magic Table entries.
	 * Check first magic table for magic tests to be applied
	 * before default tests.
	 * If no default tests are to be applied, all magic tests
	 * should occur in this magic table.
	 */
	switch (f_ckmtab(magicbuf, mread, 1)) {
		case -1:	/* Error */
			exit(2);
			break;
		case 0:		/* Not magic */
			break;
		default:	/* Switch is magic index */
			(void) putchar('\n');
			fd_cleanup();
			return (0);
			/* NOTREACHED */
			break;
	}

	if (dflg || !M_flg) {
		/*
```

```
		 * default position-dependent tests,
		 * plus non-default magic tests, if any
		 */
		switch (def_position_tests(file)) {
			case -1:	/* error */
				fd_cleanup();
				return (1);
			case 1:	/* matching type found */
				fd_cleanup();
				return (0);
				/* NOTREACHED */
				break;
			case 0:		/* no matching type found */
				break;
		}
		/* default context-sensitive tests */
		def_context_tests();
	} else {
		/* no more tests to apply; no match was found */
		(void) printf(gettext("data\n"));
	}
	fd_cleanup();
	return (0);
}

/*
 * def_position_tests() - applies default position-sensitive tests,
 *	looking for values in specific positions in the file.
 *	These are followed by default (followed by possibly some
 *	non-default) magic file tests.
 *
 *	All position-sensitive tests, default or otherwise, must
 *	be applied before context-sensitive tests, to avoid
 *	false context-sensitive matches.
 *
 *	Returns -1 on error which should result in error (non-zero)
 *	exit status for the file utility.
 *	Returns 0 if no matching file type found.
 *	Returns 1 if matching file type found.
 */

static int
def_position_tests(char *file)
{
	if (sccs()) {	/* look for "1hddddd" where d is a digit */
		(void) printf("sccs \n");
		return (1);
	}
	if (fbuf[0] == '#' && fbuf[1] == '!' && shellscript(fbuf+2, &mbuf))
		return (1);

	if (elf_check(file) == 0) {
		(void) putchar('\n');
		return (1);
	/* LINTED: pointer cast may result in improper alignment */
	} else if (*(int *)fbuf == CORE_MAGIC) {
		/* LINTED: pointer cast may result in improper alignment */
		struct core *corep = (struct core *)fbuf;

		(void) printf("a.out core file");

		if (*(corep->c_cmdname) != '\0')
			(void) printf(" from '%s'", corep->c_cmdname);
		(void) putchar('\n');
		return (1);
	}
```

```
        /*
         * Runtime linker (ld.so.1) configuration file.
         */
        if (is_rtld_config())
                return (1);

        /*
         * ZIP files, JAR files, and Java executables
         */
        if (zipfile(fbuf, ifd))
                return (1);

        if (is_crash_dump(fbuf, ifd))
                return (1);

        /*
         * ChecK against Magic Table entries.
         * The magic entries checked here always start with default
         * magic tests and may be followed by other, non-default magic
         * tests.  If no default tests are to be executed, all the
         * magic tests should have been in the first magic table.
         */
        switch (f_ckmtab(magicbuf, mread, 0)) {
                case -1:        /* Error */
                        exit(2);
                        break;
                case 0:         /* Not magic */
                        return (0);
                        /* NOTREACHED */
                        break;
                default:        /* Switch is magic index */

                        /*
                         * f_ckmtab recognizes file type,
                         * check if it is PostScript.
                         * if not, check if elf or a.out
                         */
                        if (magicbuf[0] == '%' && magicbuf[1] == '!') {
                                (void) putchar('\n');
                        } else {

                                /*
                                 * Check that the file is executable (dynamic
                                 * objects must be executable to be exec'ed,
                                 * shared objects need not be, but by convention
                                 * should be executable).
                                 *
                                 * Note that we should already have processed
                                 * the file if it was an ELF file.
                                 */
                                ar_coff_or_aout(elffd);
                                (void) putchar('\n');
                        }
                        return (1);
                        /* NOTREACHED */
                        break;
        }

        return (0);     /* file was not identified */
}

/*
 * def_context_tests() - default context-sensitive tests.
 *      These are the last tests to be applied.
 *      If no match is found, prints out "data".
```

```
 */

static void
def_context_tests(void)
{
        int     j;
        int     nl;
        char    ch;
        int     len;

        if (ccom() == 0)
                goto notc;
        while (fbuf[i] == '#') {
                j = i;
                while (fbuf[i++] != '\n') {
                        if (i - j > 255) {
                                (void) printf(gettext("data\n"));
                                return;
                        }
                        if (i >= fbsz)
                                goto notc;
                }
                if (ccom() == 0)
                        goto notc;
        }
check:
        if (lookup(c) == 1) {
                while ((ch = fbuf[i]) != ';' && ch != '{') {
                        if ((len = mblen(&fbuf[i], MB_CUR_MAX)) <= 0)
                                len = 1;
                        i += len;
                        if (i >= fbsz)
                                goto notc;
                }
                (void) printf(gettext("c program text"));
                goto outa;
        }
        nl = 0;
        while (fbuf[i] != '(') {
                if (fbuf[i] <= 0)
                        goto notas;
                if (fbuf[i] == ';') {
                        i++;
                        goto check;
                }
                if (fbuf[i++] == '\n')
                        if (nl++ > 6)
                                goto notc;
                if (i >= fbsz)
                        goto notc;
        }
        while (fbuf[i] != ')') {
                if (fbuf[i++] == '\n')
                        if (nl++ > 6)
                                goto notc;
                if (i >= fbsz)
                        goto notc;
        }
        while (fbuf[i] != '{') {
                if ((len = mblen(&fbuf[i], MB_CUR_MAX)) <= 0)
                        len = 1;
                if (fbuf[i] == '\n')
                        if (nl++ > 6)
                                goto notc;
                i += len;
                if (i >= fbsz)
```

```
			goto notc;
	}
	(void) printf(gettext("c program text"));
	goto outa;
notc:
	i = 0;			/* reset to begining of file again */
	while (fbuf[i] == 'c' || fbuf[i] == 'C'|| fbuf[i] == '!' ||
	    fbuf[i] == '*' || fbuf[i] == '\n') {
		while (fbuf[i++] != '\n')
			if (i >= fbsz)
				goto notfort;
	}
	if (lookup(fort) == 1) {
		(void) printf(gettext("fortran program text"));
		goto outa;
	}
notfort:			/* looking for assembler program */
	i = 0;			/* reset to beginning of file again */
	if (ccom() == 0)	/* assembler programs may contain */
				/* c-style comments */
		goto notas;
	if (ascom() == 0)
		goto notas;
	j = i - 1;
	if (fbuf[i] == '.') {
		i++;
		if (lookup(as) == 1) {
			(void) printf(gettext("assembler program text"));
			goto outa;
		} else if (j != -1 && fbuf[j] == '\n' && isalpha(fbuf[j + 2])) {
			(void) printf(
			    gettext("[nt]roff, tbl, or eqn input text"));
			goto outa;
		}
	}
	while (lookup(asc) == 0) {
		if (ccom() == 0)
			goto notas;
		if (ascom() == 0)
			goto notas;
		while (fbuf[i] != '\n' && fbuf[i++] != ':') {
			if (i >= fbsz)
				goto notas;
		}
		while (fbuf[i] == '\n' || fbuf[i] == ' ' || fbuf[i] == '\t')
			if (i++ >= fbsz)
				goto notas;
		j = i - 1;
		if (fbuf[i] == '.') {
			i++;
			if (lookup(as) == 1) {
				(void) printf(
				    gettext("assembler program text"));
				goto outa;
			} else if (fbuf[j] == '\n' && isalpha(fbuf[j+2])) {
				(void) printf(
				    gettext("[nt]roff, tbl, or eqn input "
				    "text"));
				goto outa;
			}
		}
	}
	(void) printf(gettext("assembler program text"));
	goto outa;
notas:
	/* start modification for multibyte env */
```

```
	IS_ascii = 1;
	if (fbsz < FBSZ)
		Max = fbsz;
	else
		Max = FBSZ - MB_LEN_MAX; /* prevent cut of wchar read */
	/* end modification for multibyte env */

	for (i = 0; i < Max; /* null */)
		if (fbuf[i] & 0200) {
			IS_ascii = 0;
			if (fbuf[0] == '\100' && fbuf[1] == '\357') {
				(void) printf(gettext("troff output\n"));
				return;
			}
		/* start modification for multibyte env */
			if ((length = mbtowc(&wchar, &fbuf[i], MB_CUR_MAX))
			    <= 0 || !iswprint(wchar)) {
				(void) printf(gettext("data\n"));
				return;
			}
			i += length;
		}
		else
			i++;
	i = fbsz;
		/* end modification for multibyte env */
	if (mbuf.st_mode&(S_IXUSR|S_IXGRP|S_IXOTH))
		(void) printf(gettext("commands text"));
	else if (troffint(fbuf, fbsz))
		(void) printf(gettext("troff intermediate output text"));
	else if (english(fbuf, fbsz))
		(void) printf(gettext("English text"));
	else if (IS_ascii)
		(void) printf(gettext("ascii text"));
	else
		(void) printf(gettext("text")); /* for multibyte env */
outa:
	/*
	 * This code is to make sure that no MB char is cut in half
	 * while still being used.
	 */
	fbsz = (fbsz < FBSZ ? fbsz : fbsz - MB_CUR_MAX + 1);
	while (i < fbsz) {
		if (isascii(fbuf[i])) {
			i++;
			continue;
		} else {
			if ((length = mbtowc(&wchar, &fbuf[i], MB_CUR_MAX))
			    <= 0 || !iswprint(wchar)) {
				(void) printf(gettext(" with garbage\n"));
				return;
			}
			i = i + length;
		}
	}
	(void) printf("\n");
}

static int
troffint(char *bp, int n)
{
	int k;

	i = 0;
	for (k = 0; k < 6; k++) {
		if (lookup(troff) == 0)
```

```
                        return (0);
                if (lookup(troff) == 0)
                        return (0);
                while (i < n && bp[i] != '\n')
                        i++;
                if (i++ >= n)
                        return (0);
        }
        return (1);
}

static void
ar_coff_or_aout(int elffd)
{
        Elf *elf;

        /*
         * Get the files elf descriptor and process it as an elf or
         * a.out (4.x) file.
         */

        elf = elf_begin(elffd, ELF_C_READ, (Elf *)0);
        switch (elf_kind(elf)) {
                case ELF_K_AR :
                        (void) printf(gettext(", not a dynamic executable "
                            "or shared object"));
                        break;
                case ELF_K_COFF:
                        (void) printf(gettext(", unsupported or unknown "
                            "file type"));
                        break;
                default:
                        /*
                         * This is either an unknown file or an aout format
                         * At this time, we don't print dynamic/stripped
                         * info. on a.out or non-Elf binaries.
                         */
                        break;
        }
        (void) elf_end(elf);
}


static void
print_elf_type(Elf_Info EI)
{
        switch (EI.type) {
        case ET_NONE:
                (void) printf(" %s", gettext("unknown type"));
                break;
        case ET_REL:
                (void) printf(" %s", gettext("relocatable"));
                break;
        case ET_EXEC:
                (void) printf(" %s", gettext("executable"));
                break;
        case ET_DYN:
                (void) printf(" %s", gettext("dynamic lib"));
                break;
        default:
                break;
        }
}

static void
print_elf_machine(int machine)
```

```
{
        /*
         * This table must be kept in sync with the EM_ constants
         * in /usr/include/sys/elf.h.
         */
        static const char *mach_str[EM_NUM] = {
                "unknown machine",              /* 0 - EM_NONE */
                "WE32100",                      /* 1 - EM_M32 */
                "SPARC",                        /* 2 - EM_SPARC */
                "80386",                        /* 3 - EM_386 */
                "M68000",                       /* 4 - EM_68K */
                "M88000",                       /* 5 - EM_88K */
                "80486",                        /* 6 - EM_486 */
                "i860",                         /* 7 - EM_860 */
                "MIPS RS3000 Big-Endian",       /* 8 - EM_MIPS */
                "S/370",                        /* 9 - EM_S370 */
                "MIPS RS3000 Little-Endian",    /* 10 - EM_MIPS_RS3_LE */
                "MIPS RS6000",                  /* 11 - EM_RS6000 */
                NULL,                           /* 12 - EM_UNKNOWN12 */
                NULL,                           /* 13 - EM_UNKNOWN13 */
                NULL,                           /* 14 - EM_UNKNOWN14 */
                "PA-RISC",                      /* 15 - EM_PA_RISC */
                "nCUBE",                        /* 16 - EM_nCUBE */
                "VPP500",                       /* 17 - EM_VPP500 */
                "SPARC32PLUS",                  /* 18 - EM_SPARC32PLUS */
                "i960",                         /* 19 - EM_960 */
                "PowerPC",                      /* 20 - EM_PPC */
                "PowerPC64",                    /* 21 - EM_PPC64 */
                "S/390",                        /* 22 - EM_S390 */
                NULL,                           /* 23 - EM_UNKNOWN23 */
                NULL,                           /* 24 - EM_UNKNOWN24 */
                NULL,                           /* 25 - EM_UNKNOWN25 */
                NULL,                           /* 26 - EM_UNKNOWN26 */
                NULL,                           /* 27 - EM_UNKNOWN27 */
                NULL,                           /* 28 - EM_UNKNOWN28 */
                NULL,                           /* 29 - EM_UNKNOWN29 */
                NULL,                           /* 30 - EM_UNKNOWN30 */
                NULL,                           /* 31 - EM_UNKNOWN31 */
                NULL,                           /* 32 - EM_UNKNOWN32 */
                NULL,                           /* 33 - EM_UNKNOWN33 */
                NULL,                           /* 34 - EM_UNKNOWN34 */
                NULL,                           /* 35 - EM_UNKNOWN35 */
                "V800",                         /* 36 - EM_V800 */
                "FR20",                         /* 37 - EM_FR20 */
                "RH32",                         /* 38 - EM_RH32 */
                "RCE",                          /* 39 - EM_RCE */
                "ARM",                          /* 40 - EM_ARM */
                "Alpha",                        /* 41 - EM_ALPHA */
                "S/390",                        /* 42 - EM_SH */
                "SPARCV9",                      /* 43 - EM_SPARCV9 */
                "Tricore",                      /* 44 - EM_TRICORE */
                "ARC",                          /* 45 - EM_ARC */
                "H8/300",                       /* 46 - EM_H8_300 */
                "H8/300H",                      /* 47 - EM_H8_300H */
                "H8S",                          /* 48 - EM_H8S */
                "H8/500",                       /* 49 - EM_H8_500 */
                "IA64",                         /* 50 - EM_IA_64 */
                "MIPS-X",                       /* 51 - EM_MIPS_X */
                "Coldfire",                     /* 52 - EM_COLDFIRE */
                "M68HC12",                      /* 53 - EM_68HC12 */
                "MMA",                          /* 54 - EM_MMA */
                "PCP",                          /* 55 - EM_PCP */
                "nCPU",                         /* 56 - EM_NCPU */
                "NDR1",                         /* 57 - EM_NDR1 */
                "Starcore",                     /* 58 - EM_STARCORE */
                "ME16",                         /* 59 - EM_ME16 */
```

```
		"ST100",			/* 60 - EM_ST100 */
		"TINYJ",			/* 61 - EM_TINYJ */
		"AMD64",			/* 62 - EM_AMD64 */
		"PDSP",				/* 63 - EM_PDSP */
		NULL,				/* 64 - EM_UNKNOWN64 */
		NULL,				/* 65 - EM_UNKNOWN65 */
		"FX66",				/* 66 - EM_FX66 */
		"ST9 PLUS",			/* 67 - EM_ST9PLUS */
		"ST7",				/* 68 - EM_ST7 */
		"68HC16",			/* 69 - EM_68HC16 */
		"68HC11",			/* 70 - EM_68HC11 */
		"68H08",			/* 71 - EM_68HC08 */
		"68HC05",			/* 72 - EM_68HC05 */
		"SVX",				/* 73 - EM_SVX */
		"ST19",				/* 74 - EM_ST19 */
		"VAX",				/* 75 - EM_VAX */
		"CRIS",				/* 76 - EM_CRIS */
		"Javelin",			/* 77 - EM_JAVELIN */
		"Firepath",			/* 78 - EM_FIREPATH */
		"ZSP",				/* 79 - EM_ZSP */
		"MMIX",				/* 80 - EM_MMIX */
		"HUANY",			/* 81 - EM_HUANY */
		"Prism",			/* 82 - EM_PRISM */
		"AVR",				/* 83 - EM_AVR */
		"FR30",				/* 84 - EM_FR30 */
		"D10V",				/* 85 - EM_D10V */
		"D30V",				/* 86 - EM_D30V */
		"V850",				/* 87 - EM_V850 */
		"M32R",				/* 88 - EM_M32R */
		"MN10300",			/* 89 - EM_MN10300 */
		"MN10200",			/* 90 - EM_MN10200 */
		"picoJava",			/* 91 - EM_PJ */
		"OpenRISC",			/* 92 - EM_OPENRISC */
		"Tangent-A5",			/* 93 - EM_ARC_A5 */
		"Xtensa"			/* 94 - EM_XTENSA */
	};
	/* If new machine is added, refuse to compile until we're updated */
#if EM_NUM != 95
#error "Number of known ELF machine constants has changed"
#endif

	const char *str;

	if ((machine < EM_NONE) || (machine >= EM_NUM))
		machine = EM_NONE;

	str = mach_str[machine];
	if (str)
		(void) printf(" %s", str);
}

static void
print_elf_datatype(int datatype)
{
	switch (datatype) {
	case ELFDATA2LSB:
		(void) printf(" LSB");
		break;
	case ELFDATA2MSB:
		(void) printf(" MSB");
		break;
	default:
		break;
	}
}
```

```
static void
print_elf_class(int class)
{
	switch (class) {
	case ELFCLASS32:
		(void) printf(" %s", gettext("32-bit"));
		break;
	case ELFCLASS64:
		(void) printf(" %s", gettext("64-bit"));
		break;
	default:
		break;
	}
}

static void
print_elf_flags(Elf_Info EI)
{
	unsigned int flags;

	flags = EI.flags;
	switch (EI.machine) {
	case EM_SPARCV9:
		if (flags & EF_SPARC_EXT_MASK) {
			if (flags & EF_SPARC_SUN_US3) {
				(void) printf("%s", gettext(
				    ", UltraSPARC3 Extensions Required"));
			} else if (flags & EF_SPARC_SUN_US1) {
				(void) printf("%s", gettext(
				    ", UltraSPARC1 Extensions Required"));
			}
			if (flags & EF_SPARC_HAL_R1)
				(void) printf("%s", gettext(
				    ", HaL R1 Extensions Required"));
		}
		break;
	case EM_SPARC32PLUS:
		if (flags & EF_SPARC_32PLUS)
			(void) printf("%s", gettext(", V8+ Required"));
		if (flags & EF_SPARC_SUN_US3) {
			(void) printf("%s",
			    gettext(", UltraSPARC3 Extensions Required"));
		} else if (flags & EF_SPARC_SUN_US1) {
			(void) printf("%s",
			    gettext(", UltraSPARC1 Extensions Required"));
		}
		if (flags & EF_SPARC_HAL_R1)
			(void) printf("%s",
			    gettext(", HaL R1 Extensions Required"));
		break;
	default:
		break;
	}
}

/*
 * check_ident:	checks the ident field of the presumeably
 *		elf file. If check fails, this is not an
 *		elf file.
 */
static int
check_ident(unsigned char *ident, int fd)
{
	int class;
	if (pread64(fd, ident, EI_NIDENT, 0) != EI_NIDENT)
		return (ELF_READ_FAIL);
```

```
        class = ident[EI_CLASS];
        if (class != ELFCLASS32 && class != ELFCLASS64)
                return (ELF_READ_FAIL);
        if (ident[EI_MAG0] != ELFMAG0 || ident[EI_MAG1] != ELFMAG1 ||
            ident[EI_MAG2] != ELFMAG2 || ident[EI_MAG3] != ELFMAG3)
                return (ELF_READ_FAIL);

        return (ELF_READ_OKAY);
}

static int
elf_check(char *file)
{
        Elf_Info EInfo;
        int class, version, format;
        unsigned char ident[EI_NIDENT];

        (void) memset(&EInfo, 0, sizeof (Elf_Info));
        EInfo.file = file;

        /*
         * Verify information in file indentifier.
         * Return quietly if not elf; Different type of file.
         */
        if (check_ident(ident, elffd) == ELF_READ_FAIL)
                return (1);

        /*
         * Read the elf headers for processing and get the
         * get the needed information in Elf_Info struct.
         */
        class = ident[EI_CLASS];
        if (class == ELFCLASS32) {
                if (elf_read32(elffd, &EInfo) == ELF_READ_FAIL) {
                        (void) fprintf(stderr, gettext("%s: %s: can't "
                            "read ELF header\n"), File, file);
                        return (1);
                }
        } else if (class == ELFCLASS64) {
                if (elf_read64(elffd, &EInfo) == ELF_READ_FAIL) {
                        (void) fprintf(stderr, gettext("%s: %s: can't "
                            "read ELF header\n"), File, file);
                        return (1);
                }
        } else {
                /* something wrong */
                return (1);
        }

        /* version not in ident then 1 */
        version = ident[EI_VERSION] ? ident[EI_VERSION] : 1;

        format = ident[EI_DATA];
        (void) printf("%s", gettext("ELF"));
        print_elf_class(class);
        print_elf_datatype(format);
        print_elf_type(EInfo);

        if (EInfo.core_type != EC_NOTCORE) {
                /* Print what kind of core is this */
                if (EInfo.core_type == EC_OLDCORE)
                        (void) printf(" %s", gettext("pre-2.6 core file"));
                else
                        (void) printf(" %s", gettext("core file"));
        }
```

```
        /* Print machine info */
        print_elf_machine(EInfo.machine);

        /* Print Version */
        if (version == 1)
                (void) printf(" %s %d", gettext("Version"), version);

        /* Print Flags */
        print_elf_flags(EInfo);

        /* Last bit, if it is a core */
        if (EInfo.core_type != EC_NOTCORE) {
                /* Print the program name that dumped this core */
                (void) printf(gettext(", from '%s'"), EInfo.fname);
                return (0);
        }

        /* Print Capabilities */
        if (EInfo.cap_str[0] != '\0')
                (void) printf(" [%s]", EInfo.cap_str);

        if ((EInfo.type != ET_EXEC) && (EInfo.type != ET_DYN))
                return (0);

        /* Print if it is dynamically linked */
        if (EInfo.dynamic)
                (void) printf(gettext(", dynamically linked"));
        else
                (void) printf(gettext(", statically linked"));

        /* Printf it it is stripped */
        if (EInfo.stripped & E_SYMTAB) {
                (void) printf(gettext(", not stripped"));
                if (!(EInfo.stripped & E_DBGINF)) {
                        (void) printf(gettext(
                            ", no debugging information available"));
                }
        } else {
                (void) printf(gettext(", stripped"));
        }

        return (0);
}

/*
 * is_rtld_config - If file is a runtime linker config file, prints
 * the description and returns True (1). Otherwise, silently returns
 * False (0).
 */
int
is_rtld_config(void)
{
        Rtc_id *id;

        if ((fbsz >= sizeof (*id)) && RTC_ID_TEST(fbuf)) {
                (void) printf(gettext("Runtime Linking Configuration"));
                id = (Rtc_id *) fbuf;
                print_elf_class(id->id_class);
                print_elf_datatype(id->id_data);
                print_elf_machine(id->id_machine);
                (void) printf("\n");
                return (1);
        }

        return (0);
}
```

```
/*
 * lookup -
 * Attempts to match one of the strings from a list, 'tab',
 * with what is in the file, starting at the current index position 'i'.
 * Looks past any initial whitespace and expects whitespace or other
 * delimiting characters to follow the matched string.
 * A match identifies the file as being 'assembler', 'fortran', 'c', etc.
 * Returns 1 for a successful match, 0 otherwise.
 */
static int
lookup(char **tab)
{
        register char   r;
        register int    k, j, l;

        while (fbuf[i] == ' ' || fbuf[i] == '\t' || fbuf[i] == '\n')
                i++;
        for (j = 0; tab[j] != 0; j++) {
                l = 0;
                for (k = i; ((r = tab[j][l++]) == fbuf[k] && r != '\0'); k++)
                        ;
                if (r == '\0')
                        if (fbuf[k] == ' ' || fbuf[k] == '\n' ||
                            fbuf[k] == '\t' || fbuf[k] == '{' ||
                            fbuf[k] == '/') {
                                i = k;
                                return (1);
                        }
        }
        return (0);
}

/*
 * ccom -
 * Increments the current index 'i' into the file buffer 'fbuf' past any
 * whitespace lines and C-style comments found, starting at the current
 * position of 'i'.  Returns 1 as long as we don't increment i past the
 * size of fbuf (fbsz).  Otherwise, returns 0.
 */

static int
ccom(void)
{
        register char   cc;
        int             len;

        while ((cc = fbuf[i]) == ' ' || cc == '\t' || cc == '\n')
                if (i++ >= fbsz)
                        return (0);
        if (fbuf[i] == '/' && fbuf[i+1] == '*') {
                i += 2;
                while (fbuf[i] != '*' || fbuf[i+1] != '/') {
                        if (fbuf[i] == '\\')
                                i++;
                        if ((len = mblen(&fbuf[i], MB_CUR_MAX)) <= 0)
                                len = 1;
                        i += len;
                        if (i >= fbsz)
                                return (0);
                }
                if ((i += 2) >= fbsz)
                        return (0);
        }
        if (fbuf[i] == '\n')
                if (ccom() == 0)
```

```
                        return (0);
        return (1);
}

/*
 * ascom -
 * Increments the current index 'i' into the file buffer 'fbuf' past
 * consecutive assembler program comment lines starting with ASCOMCHAR,
 * starting at the current position of 'i'.
 * Returns 1 as long as we don't increment i past the
 * size of fbuf (fbsz).  Otherwise returns 0.
 */

static int
ascom(void)
{
        while (fbuf[i] == ASCOMCHAR) {
                i++;
                while (fbuf[i++] != '\n')
                        if (i >= fbsz)
                                return (0);
                while (fbuf[i] == '\n')
                        if (i++ >= fbsz)
                                return (0);
        }
        return (1);
}

static int
sccs(void)
{                               /* look for "1hddddd" where d is a digit */
        register int j;

        if (fbuf[0] == 1 && fbuf[1] == 'h') {
                for (j = 2; j <= 6; j++) {
                        if (isdigit(fbuf[j]))
                                continue;
                        else
                                return (0);
                }
        } else {
                return (0);
        }
        return (1);
}

static int
english(char *bp, int n)
{
#define NASC 128                /* number of ascii char ?? */
        register int    j, vow, freq, rare, len;
        register int    badpun = 0, punct = 0;
        int     ct[NASC];

        if (n < 50)
                return (0); /* no point in statistics on squibs */
        for (j = 0; j < NASC; j++)
                ct[j] = 0;
        for (j = 0; j < n; j += len) {
                if ((unsigned char)bp[j] < NASC)
                        ct[bp[j]|040]++;
                switch (bp[j]) {
                case '.':
                case ',':
                case ')':
                case '%':
```

```c
                case ';':
                case ':':
                case '?':
                        punct++;
                        if (j < n-1 && bp[j+1] != ' ' && bp[j+1] != '\n')
                                badpun++;
                }
                if ((len = mblen(&bp[j], MB_CUR_MAX)) <= 0)
                        len = 1;
        }
        if (badpun*5 > punct)
                return (0);
        vow = ct['a'] + ct['e'] + ct['i'] + ct['o'] + ct['u'];
        freq = ct['e'] + ct['t'] + ct['a'] + ct['i'] + ct['o'] + ct['n'];
        rare = ct['v'] + ct['j'] + ct['k'] + ct['q'] + ct['x'] + ct['z'];
        if (2*ct[';'] > ct['e'])
                return (0);
        if ((ct['>'] + ct['<'] + ct['/']) > ct['e'])
                return (0);     /* shell file test */
        return (vow * 5 >= n - ct[' '] && freq >= 10 * rare);
}


static int
shellscript(char buf[], struct stat64 *sb)
{
        char *tp, *cp, *xp, *up, *gp;

        cp = strchr(buf, '\n');
        if (cp == NULL || cp - fbuf > fbsz)
                return (0);
        for (tp = buf; tp != cp && isspace((unsigned char)*tp); tp++)
                if (!isascii(*tp))
                        return (0);
        for (xp = tp; tp != cp && !isspace((unsigned char)*tp); tp++)
                if (!isascii(*tp))
                        return (0);
        if (tp == xp)
                return (0);
        if (sb->st_mode & S_ISUID)
                up = gettext("set-uid ");
        else
                up = "";

        if (sb->st_mode & S_ISGID)
                gp = gettext("set-gid ");
        else
                gp = "";

        if (strncmp(xp, "/bin/sh", tp - xp) == 0)
                xp = gettext("shell");
        else if (strncmp(xp, "/bin/csh", tp - xp) == 0)
                xp = gettext("c-shell");
        else if (strncmp(xp, "/usr/sbin/dtrace", tp - xp) == 0)
                xp = gettext("DTrace");
        else
                *tp = '\0';
        /*
         * TRANSLATION_NOTE
         * This message is printed by file command for shell scripts.
         * The first %s is for the translation for "set-uid " (if the script
         *   has the set-uid bit set), or is for an empty string (if the
         *   script does not have the set-uid bit set).
         * Similarly, the second %s is for the translation for "set-gid ",
         *   or is for an empty string.
         * The third %s is for the translation for either: "shell", "c-shell",
```

```c
         *   or "DTrace", or is for the pathname of the program the script
         *   executes.
         */
        (void) printf(gettext("%s%sexecutable %s script\n"), up, gp, xp);
        return (1);
}

static int
get_door_target(char *file, char *buf, size_t bufsize)
{
        int fd;
        door_info_t di;
        psinfo_t psinfo;

        if ((fd = open64(file, O_RDONLY)) < 0 ||
            door_info(fd, &di) != 0) {
                if (fd >= 0)
                        (void) close(fd);
                return (-1);
        }
        (void) close(fd);

        (void) sprintf(buf, "/proc/%ld/psinfo", di.di_target);
        if ((fd = open64(buf, O_RDONLY)) < 0 ||
            read(fd, &psinfo, sizeof (psinfo)) != sizeof (psinfo)) {
                if (fd >= 0)
                        (void) close(fd);
                return (-1);
        }
        (void) close(fd);

        (void) snprintf(buf, bufsize, "%s[%ld]", psinfo.pr_fname, di.di_target);
        return (0);
}

/*
 * ZIP file header information
 */
#define SIGSIZ          4
#define LOCSIG          "PK\003\004"
#define LOCHDRSIZ       30

#define CH(b, n)        (((unsigned char *)(b))[n])
#define SH(b, n)        (CH(b, n) | (CH(b, n+1) << 8))
#define LG(b, n)        (SH(b, n) | (SH(b, n+2) << 16))

#define LOCNAM(b)       (SH(b, 26))     /* filename size */
#define LOCEXT(b)       (SH(b, 28))     /* extra field size */

#define XFHSIZ          4               /* header id, data size */
#define XFHID(b)        (SH(b, 0))      /* extract field header id */
#define XFDATASIZ(b)    (SH(b, 2))      /* extract field data size */
#define XFJAVASIG       0xcafe          /* java executables */

static int
zipfile(char *fbuf, int fd)
{
        off_t xoff, xoff_end;

        if (strncmp(fbuf, LOCSIG, SIGSIZ) != 0)
                return (0);

        xoff = LOCHDRSIZ + LOCNAM(fbuf);
        xoff_end = xoff + LOCEXT(fbuf);

        while (xoff < xoff_end) {
```

```
		char xfhdr[XFHSIZ];

		if (pread(fd, xfhdr, XFHSIZ, xoff) != XFHSIZ)
			break;

		if (XFHID(xfhdr) == XFJAVASIG) {
			(void) printf("%s\n", gettext("java archive file"));
			return (1);
		}
		xoff += sizeof (xfhdr) + XFDATASIZ(xfhdr);
	}

	/*
	 * We could just print "ZIP archive" here.
	 *
	 * However, customers may be using their own entries in
	 * /etc/magic to distinguish one kind of ZIP file from another, so
	 * let’s defer the printing of "ZIP archive" to there.
	 */
	return (0);
}

static int
is_crash_dump(const char *buf, int fd)
{
	/* LINTED: pointer cast may result in improper alignment */
	const dumphdr_t *dhp = (const dumphdr_t *)buf;

	/*
	 * The current DUMP_MAGIC string covers Solaris 7 and later releases.
	 * The utsname struct is only present in dumphdr_t’s with dump_version
	 * greater than or equal to 9.
	 */
	if (dhp->dump_magic == DUMP_MAGIC) {
		print_dumphdr(fd, dhp, return_uint32, NATIVE_ISA);

	} else if (dhp->dump_magic == swap_uint32(DUMP_MAGIC)) {
		print_dumphdr(fd, dhp, swap_uint32, OTHER_ISA);

	} else if (dhp->dump_magic == OLD_DUMP_MAGIC ||
	    dhp->dump_magic == swap_uint32(OLD_DUMP_MAGIC)) {
		char *isa = (dhp->dump_magic == OLD_DUMP_MAGIC ?
		    NATIVE_ISA : OTHER_ISA);
		(void) printf(gettext("SunOS 32-bit %s crash dump\n"), isa);

	} else {
		return (0);
	}

	return (1);
}

static void
print_dumphdr(const int fd, const dumphdr_t *dhp, uint32_t (*swap)(uint32_t),
    const char *isa)
{
	dumphdr_t dh;

	/*
	 * A dumphdr_t is bigger than FBSZ, so we have to manually read the
	 * rest of it.
	 */
	if (swap(dhp->dump_version) > 8 && pread(fd, &dh, sizeof (dumphdr_t),
	    (off_t)0) == sizeof (dumphdr_t)) {
		const char *c = swap(dh.dump_flags) & DF_COMPRESSED ?
		    "compressed " : "";
```

```
		const char *l = swap(dh.dump_flags) & DF_LIVE ?
		    "live" : "crash";

		(void) printf(gettext(
		    "%s %s %s %u-bit %s %s%s dump from ’%s’\n"),
		    dh.dump_utsname.sysname, dh.dump_utsname.release,
		    dh.dump_utsname.version, swap(dh.dump_wordsize), isa,
		    c, l, dh.dump_utsname.nodename);
	} else {
		(void) printf(gettext("SunOS %u-bit %s crash dump\n"),
		    swap(dhp->dump_wordsize), isa);
	}
}

static void
usage(void)
{
	(void) fprintf(stderr, gettext(
	    "usage: file [-bdh] [-M mfile] [-m mfile] [-f ffile] file ...\n"
	    "       file [-bdh] [-M mfile] [-m mfile] -f ffile\n"
	    "       file -i [-bh] [-f ffile] file ...\n"
	    "       file -i [-bh] -f ffile\n"
	    "usage: file [-dh] [-M mfile] [-m mfile] [-f ffile] file ...\n"
	    "       file [-dh] [-M mfile] [-m mfile] -f ffile\n"
	    "       file -i [-h] [-f ffile] file ...\n"
	    "       file -i [-h] -f ffile\n"
	    "       file -c [-d] [-M mfile] [-m mfile]\n"));
	exit(2);
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
    8986 Sat Apr 26 08:41:45 2014
new/usr/src/man/man1/file.1
4732 /usr/bin/file should provide -b option  for compatiblity with GNU/BSD file
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
Reviewed by: Serghei Samsi <sscdvp@gmail.com>
Reviewed by: Alexander Pyhalov <alp@rsu.ru>
Reviewed by: Garrett D'Amore <garrett@damore.org>
**********************************************************
'\" te
.\" Copyright 1989 AT&T Copyright (c) 1992, X/Open Company Limited All Rights Re
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\"  This notice shall appear on any product containing this material.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH FILE 1 "Apr 11, 2014"
.TH FILE 1 "Sep 10, 2013"
.SH NAME
file \- determine file type
.SH SYNOPSIS
.LP
.nf
\fB/usr/bin/file\fR [\fB-bdh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\fR] [
\fB/usr/bin/file\fR [\fB-dh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\fR] [\
.fi

.LP
.nf
\fB/usr/bin/file\fR [\fB-bdh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\fR] \
\fB/usr/bin/file\fR [\fB-dh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\fR] \f
.fi

.LP
.nf
\fB/usr/bin/file\fR \fB-i\fR [\fB-bh\fR] [\fB-f\fR \fIffile\fR] \fIfile\fR...
\fB/usr/bin/file\fR \fB-i\fR [\fB-h\fR] [\fB-f\fR \fIffile\fR] \fIfile\fR...
.fi

.LP
.nf
\fB/usr/bin/file\fR \fB-i\fR [\fB-bh\fR] \fB-f\fR \fIffile\fR
\fB/usr/bin/file\fR \fB-i\fR [\fB-h\fR] \fB-f\fR \fIffile\fR
.fi

.LP
.nf
\fB/usr/bin/file\fR \fB-c\fR [\fB-d\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfil
.fi

.LP
.nf
\fB/usr/xpg4/bin/file\fR [\fB-bdh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\
\fB/usr/xpg4/bin/file\fR [\fB-dh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\f
.fi

.LP
.nf
\fB/usr/xpg4/bin/file\fR [\fB-bdh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\
\fB/usr/xpg4/bin/file\fR [\fB-dh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\f
.fi

.LP
.nf
```

```
\fB/usr/xpg4/bin/file\fR \fB-i\fR [\fB-bh\fR] [\fB-f\fR \fIffile\fR] \fIfile\fR.
\fB/usr/xpg4/bin/file\fR \fB-i\fR [\fB-h\fR] [\fB-f\fR \fIffile\fR] \fIfile\fR..
.fi

.LP
.nf
\fB/usr/xpg4/bin/file\fR \fB-i\fR [\fB-bh\fR] \fB-f\fR \fIffile\fR
\fB/usr/xpg4/bin/file\fR \fB-i\fR [\fB-h\fR] \fB-f\fR \fIffile\fR
.fi

.LP
.nf
\fB/usr/xpg4/bin/file\fR \fB-c\fR [\fB-d\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \f
.fi

.SH DESCRIPTION
.sp
.LP
The \fBfile\fR utility performs a series of tests on each file supplied by
\fIfile\fR and, optionally, on each file listed in \fIffile\fR in an attempt to
classify it. If the file is not a regular file, its file type is identified.
The file types directory, \fBFIFO\fR, block special, and character special are
identified as such. If the file is a regular file and the file is zero-length,
it is identified as an empty file.
.sp
.LP
If \fIfile\fR appears to be a text file, \fBfile\fR examines the first 512
bytes and tries to determine its programming language. If \fIfile\fR is a
symbolic link, by default the link is followed and \fBfile\fR tests the file to
which the symbolic link refers.
.sp
.LP
If \fIfile\fR is a relocatable object, executable, or shared object, \fBfile\fR
prints out information about the file's execution requirements. This
information includes the machine class, byte-ordering, static/dynamic linkage,
and any software or hardware capability requirements. If \fIfile\fR is a
runtime linking configuration file, \fBfile\fR prints information about the
target platform, including the machine class and byte-ordering.
.sp
.LP
By default, \fBfile\fR will try to use the localized magic file
\fB/usr/lib/locale/\fIlocale\fR/LC_MESSAGES/magic\fR, if it exists, to identify
files that have a magic number. For example, in the Japanese locale, \fBfile\fR
will try to use \fB/usr/lib/locale/ja/LC_MESSAGES/magic\fR. If a localized
magic file does not exist, \fBfile\fR will utilize \fB/etc/magic\fR. A magic
number is a numeric or string constant that indicates the file type. See
\fBmagic\fR(4) for an explanation of the format of \fB/etc/magic\fR.
.sp
.LP
If \fIfile\fR does not exist, cannot be read, or its file status could not be
determined, it is not considered an error that affects the exit status. The
output will indicate that the file was processed, but that its type could not
be determined.
.SH OPTIONS
.sp
.LP
The following options are supported:
.sp
.ne 2
.na
\fB\fB-b\fR\fR
.ad
.RS 12n
Be brief, do not print leading filename.
.RE
```

```
.sp
.ne 2
.na
#endif /* ! codereview */
\fB\fB-c\fR\fR
.ad
.RS 12n
Checks the magic file for format errors. For reasons of efficiency, this
validation is normally not carried out.
.RE

.sp
.ne 2
.na
\fB\fB-d\fR\fR
.ad
.RS 12n
Applies any position-sensitive and context-sensitive default system tests to
the file.
.RE

.sp
.ne 2
.na
\fB\fB-f\fR \fIffile\fR\fR
.ad
.RS 12n
\fIffile\fR contains a list of the files to be examined.
.RE

.sp
.ne 2
.na
\fB\fB-h\fR\fR
.ad
.RS 12n
When a symbolic link is encountered, this option identifies the file as a
symbolic link. If \fB-h\fR is not specified and \fIfile\fR is a symbolic link
that refers to a non-existent file, the \fBfile\fR utility identifies the file
as a symbolic link, as if \fB-h\fR had been specified.
.RE

.sp
.ne 2
.na
\fB\fB-i\fR\fR
.ad
.RS 12n
If a file is a regular file, this option does not attempt to classify the type
of file further, but identifies the file as a "regular file".
.RE

.sp
.ne 2
.na
\fB\fB-m\fR \fImfile\fR\fR
.ad
.RS 12n
.sp
.ne 2
.na
\fB\fB/usr/bin/file\fR\fR
.ad
.RS 22n
Uses \fImfile\fR as an alternate magic file, instead of \fB/etc/magic\fR.
.RE
```

```
.sp
.ne 2
.na
\fB\fB/usr/xpg4/bin/file\fR\fR
.ad
.RS 22n
Specifies the name of a file containing position-sensitive tests that are
applied to a file in order to classify it (see \fBmagic\fR(4)). If the \fB-m\fR
option is specified without specifying the \fB-d\fR option or the \fB-M\fR
option, position-sensitive default system tests are applied after the
position-sensitive tests specified by the \fB-m\fR option.
.RE

.RE

.sp
.ne 2
.na
\fB\fB-M\fR \fIMfile\fR\fR
.ad
.RS 12n
Specifies the name of a file containing position-sensitive tests that are
applied to a file in order to classify it (see \fBmagic\fR(4)). No
position-sensitive default system tests nor context-sensitive default system
tests are applied unless the \fB-d\fR option is also specified.
.RE

.sp
.LP
If the \fB-M\fR option is specified with the \fB-d\fR option, the \fB-m\fR
option, or both, or if the \fB-m\fR option is specified with the \fB-d\fR
option, the concatenation of the position-sensitive tests specified by these
options is applied in the order specified by the appearance of these options.
.SH OPERANDS
.sp
.LP
The following operands are supported:
.sp
.ne 2
.na
\fB\fIfile\fR\fR
.ad
.RS 8n
A path name of a file to be tested.
.RE

.SH USAGE
.sp
.LP
See \fBlargefile\fR(5) for the description of the behavior of \fBfile\fR when
encountering files greater than or equal to 2 Gbyte ( 2^31 bytes).
.SH EXAMPLES
.LP
\fBExample 1 \fRDetermining if an Argument is a Binary Executable Files
.sp
.LP
The following example determine if an argument is a binary executable file:

.sp
.in +2
.nf
file "$1" | grep \(miFq executable &&
          printf "%s is executable.\en" "$1"
.fi
.in -2
```

```
.sp

.SH ENVIRONMENT VARIABLES
.sp
.LP
See \fBenviron\fR(5) for descriptions of the following environment variables
that affect the execution of \fBfile\fR: \fBLANG\fR, \fBLC_ALL\fR,
\fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
.SH EXIT STATUS
.sp
.LP
The following exit values are returned:
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.RS 6n
Successful completion.
.RE

.sp
.ne 2
.na
\fB\fB>0\fR\fR
.ad
.RS 6n
An error occurred.
.RE

.SH FILES
.sp
.ne 2
.na
\fB\fB/etc/magic\fR\fR
.ad
.RS 14n
\fBfile\fR's magic number file
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
CSI     Enabled
_
Interface Stability     Standard
.TE

.SH SEE ALSO
.sp
.LP
\fBcrle\fR(1), \fBelfdump\fR(1), \fBls\fR(1), \fBmagic\fR(4),
\fBattributes\fR(5), \fBenviron\fR(5), \fBlargefile\fR(5), \fBstandards\fR(5)
```