

```

*****
43331 Sat Apr 12 09:17:29 2014
new/usr/src/cmd/file/file.c
4732 /usr/bin/file should provide -b option for compatiblity with GNU/BSD file
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
22 /*      All Rights Reserved      */

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

28 /*
29 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
30 * Use is subject to license terms.
31 */

33 #define _LARGEFILE64_SOURCE

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

38 #include <ctype.h>
39 #include <unistd.h>
40 #include <fcntl.h>
41 #include <signal.h>
42 #include <stdio.h>
43 #include <libelf.h>
44 #include <stdlib.h>
45 #include <limits.h>
46 #include <locale.h>
47 #include <wctype.h>
48 #include <string.h>
49 #include <errno.h>
50 #include <door.h>
51 #include <sys/param.h>
52 #include <sys/types.h>
53 #include <sys/mkdev.h>
54 #include <sys/stat.h>
55 #include <sys/elf.h>
56 #include <procfs.h>
57 #include <sys/core.h>
58 #include <sys/dumphdr.h>
59 #include <netinet/in.h>
60 #include <gelf.h>
61 #include <elfcap.h>

```

```

62 #include <sgsrtcid.h>
63 #include "file.h"
64 #include "elf_read.h"

66 /*
67  *      Misc
68 */

70 #define FBSZ          512
71 #define MLIST_SZ     12

73 /*
74  * The 0x8FCA0102 magic string was used in crash dumps generated by releases
75  * prior to Solaris 7.
76 */
77 #define OLD_DUMP_MAGIC 0x8FCA0102

79 #if defined(__sparc)
80 #define NATIVE_ISA      "SPARC"
81 #define OTHER_ISA     "Intel"
82 #else
83 #define NATIVE_ISA     "Intel"
84 #define OTHER_ISA     "SPARC"
85 #endif

87 /* Assembly language comment char */
88 #ifdef pdp11
89 #define ASCOMCHAR '/'
90 #else
91 #define ASCOMCHAR '!'
92 #endif

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

97 /*
98  * Magic file variables
99 */
100 static intmax_t maxmagicoffset;
101 static intmax_t tmpmax;
102 static char *magicbuf;

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

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

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

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

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

126 /*
127  * The line and debug section names are used by the strip command.

```

```

128 * Any changes in the strip implementation need to be reflected here.
129 */
130 static char *debug_sections[] = { /* Debug sections in a ELF file */
131     ".debug", ".stab", ".dwarf", ".line", NULL};

133 /* start for MB env */
134 static wchar_t wchar;
135 static int length;
136 static int IS_ascii;
137 static int Max;
138 /* end for MB env */
139 static int i; /* global index into first 'fbsz' bytes of file */
140 static int fbsz;
141 static int ifd = -1;
142 static int elffd = -1;
143 static int tret;
144 static int hflg;
145 static int dflg;
146 static int mflg;
147 static int M_flg;
148 static int iflg;
149 static struct stat64 mbuf;

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

158 static ssize_t mread;

160 static void ar_coff_or_aout(int ifd);
161 static int type(char *file);
162 static int def_position_tests(char *file);
163 static void def_context_tests(void);
164 static int troffint(char *bp, int n);
165 static int lookup(char **tab);
166 static int ccom(void);
167 static int ascom(void);
168 static int sccs(void);
169 static int english(char *bp, int n);
170 static int shellscrip(char buf[], struct stat64 *sb);
171 static int elf_check(char *file);
172 static int get_door_target(char *, char *, size_t);
173 static int zipfile(char *, int);
174 static int is_crash_dump(const char *, int);
175 static void print_dumphdr(const int, const dumphdr_t *, uint32_t (*)(uint32_t),
176     const char *);
177 static uint32_t swap_uint32(uint32_t);
178 static uint32_t return_uint32(uint32_t);
179 static void usage(void);
180 static void default_magic(void);
181 static void add_to_mlist(char *, int);
182 static void fd_cleanup(void);
183 static int is_rtlld_confid(void);

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

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

```

```

194 #endif /* XPG4 */

196 /*
197 * Static program identifier - used to prevent localization of the name "file"
198 * within individual error messages.
199 */
200 const char *File = "file";

202 int
203 main(int argc, char **argv)
204 {
205     char *p;
206     int ch;
207     FILE *fl;
208     int bflg = 0;
209 #endif /* !codereview */
210     int cflg = 0;
211     int eflg = 0;
212     int fflg = 0;
213     char *ap = NULL;
214     int pathlen;
215     char **filep;

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

223     while ((ch = getopt(argc, argv, "M:bcdf:him:")) != EOF) {
224         while ((ch = getopt(argc, argv, "M:cdf:him:")) != EOF) {
225             switch (ch) {
226                 case 'M':
227                     add_to_mlist(optarg, !dflg);
228                     M_flg++;
229                     break;

231                 case 'b':
232                     bflg++;
233                     break;

235 #endif /* !codereview */
236                 case 'c':
237                     cflg++;
238                     break;

240                 case 'd':
241                     if (!dflg) {
242                         default_magic();
243                         add_to_mlist(dfile, 0);
244                         dflg++;
245                     }
246                     break;

248                 case 'f':
249                     fflg++;
250                     errno = 0;
251                     if ((fl = fopen(optarg, "r")) == NULL) {
252                         int err = errno;
253                         (void) fprintf(stderr, gettext("%s: cannot "
254                             "open file %s: %s\n"), File, optarg,
255                             err ? strerror(err) : "");
256                         usage();
257                     }
258                     pathlen = pathconf("/", _PC_PATH_MAX);

```

```

259         if (pathlen == -1) {
260             int err = errno;
261             (void) fprintf(stderr, gettext("%s: cannot "
262             "determine maximum path length: %s\n"),
263             File, strerror(err));
264             exit(1);
265         }
266         pathlen += 2; /* for null and newline in fgets */
267         if ((ap = malloc(pathlen * sizeof(char))) == NULL) {
268             int err = errno;
269             (void) fprintf(stderr, gettext("%s: malloc "
270             "failed: %s\n"), File, strerror(err));
271             exit(2);
272         }
273         break;
274
275     case 'h':
276         hflg++;
277         break;
278
279     case 'i':
280         iflg++;
281         break;
282
283     case 'm':
284         add_to_mlist(optarg, !dflg);
285         mflg++;
286         break;
287
288     case '?':
289         eflg++;
290         break;
291     }
292     if (!cflg && !fflg && (eflg || optind == argc))
293         usage();
294     if (iflg && (dflg || mflg || M_flg)) {
295         usage();
296     }
297     if ((iflg && cflg) || (cflg && bflg)) {
298         if (iflg && cflg) {
299             usage();
300         }
301
302         if (!dflg && !mflg && !M_flg && !iflg) {
303             /* no -d, -m, nor -M option; also -i option doesn't need magic */
304             default_magic();
305             if (f_mkmtab(dfile, cflg, 0) == -1) {
306                 exit(2);
307             }
308         }
309
310         else if (mflg && !M_flg && !dflg) {
311             /* -m specified without -d nor -M */
312
313 #ifdef XPG4
314             /* For SUSv3 only */
315
316             /*
317             * The default position-dependent magic file tests
318             * in /etc/magic will follow all the -m magic tests.
319             */
320
321             for (filep = mlist1; filep < mlist1p; filep++) {
322                 if (f_mkmtab(*filep, cflg, 1) == -1) {
323                     exit(2);
324                 }
325             }
326         }
327     }

```

```

324         }
325         default_magic();
326         if (f_mkmtab(dfile, cflg, 0) == -1) {
327             exit(2);
328         }
329 #else /* !XPG4 */
330 /*
331  * Retain Solaris file behavior for -m before SUSv3,
332  * when the new -d and -M options are not specified.
333  * Use the -m file specified in place of the default
334  * /etc/magic file. Solaris file will
335  * now allow more than one magic file to be specified
336  * with multiple -m options, for consistency with
337  * other behavior.
338  *
339  * Put the magic table(s) specified by -m into
340  * the second magic table instead of the first
341  * (as indicated by the last argument to f_mkmtab()),
342  * since they replace the /etc/magic tests and
343  * must be executed alongside the default
344  * position-sensitive tests.
345  */
346
347     for (filep = mlist1; filep < mlist1p; filep++) {
348         if (f_mkmtab(*filep, cflg, 0) == -1) {
349             exit(2);
350         }
351     }
352 #endif /* XPG4 */
353     } else {
354         /*
355          * For any other combination of -d, -m, and -M,
356          * use the magic files in command-line order.
357          * Store the entries from the two separate lists of magic
358          * files, if any, into two separate magic file tables.
359          * mlist1: magic tests executed before default magic tests
360          * mlist2: default magic tests and after
361          */
362         for (filep = mlist1; filep && (filep < mlist1p); filep++) {
363             if (f_mkmtab(*filep, cflg, 1) == -1) {
364                 exit(2);
365             }
366         }
367         for (filep = mlist2; filep && (filep < mlist2p); filep++) {
368             if (f_mkmtab(*filep, cflg, 0) == -1) {
369                 exit(2);
370             }
371         }
372     }
373
374     /* Initialize the magic file variables; check both magic tables */
375     tmpmax = f_getmaxoffset(1);
376     maxmagicoffset = f_getmaxoffset(0);
377     if (maxmagicoffset < tmpmax) {
378         maxmagicoffset = tmpmax;
379     }
380     if (maxmagicoffset < (intmax_t)FBSZ)
381         maxmagicoffset = (intmax_t)FBSZ;
382     if ((magicbuf = malloc(maxmagicoffset)) == NULL) {
383         int err = errno;
384         (void) fprintf(stderr, gettext("%s: malloc failed: %s\n"),
385         File, strerror(err));
386         exit(2);
387     }
388
389     if (cflg) {

```

```

390     f_prtmtab();
391     if (ferror(stdout) != 0) {
392         (void) fprintf(stderr, gettext("%s: error writing to "
393             "stdout\n"), File);
394         exit(1);
395     }
396     if (fclose(stdout) != 0) {
397         int err = errno;
398         (void) fprintf(stderr, gettext("%s: fclose "
399             "failed: %s\n"), File, strerror(err));
400         exit(1);
401     }
402     exit(0);
403 }
404
405 for (; fflg || optind < argc; optind += !fflg) {
406     register int    l;
407
408     if (fflg) {
409         if ((p = fgets(ap, pathlen, fl)) == NULL) {
410             fflg = 0;
411             optind--;
412             continue;
413         }
414         l = strlen(p);
415         if (l > 0)
416             p[l - 1] = '\\0';
417     } else
418         p = argv[optind];
419
420     if (!bflg)
421 #endif /* !codereview */
422         prf(p);          /* print "file_name:<tab>" */
423
424     if (type(p))
425         tret = 1;
426 }
427 if (ap != NULL)
428     free(ap);
429 if (tret != 0)
430     exit(tret);
431
432 if (ferror(stdout) != 0) {
433     (void) fprintf(stderr, gettext("%s: error writing to "
434         "stdout\n"), File);
435     exit(1);
436 }
437 if (fclose(stdout) != 0) {
438     int err = errno;
439     (void) fprintf(stderr, gettext("%s: fclose failed: %s\n"),
440         File, strerror(err));
441     exit(1);
442 }
443 return (0);
444 }
445
446 static int
447 type(char *file)
448 {
449     int    cc;
450     char  buf[BUFSIZ];
451     int    (*statf)() = hflg ? lstat64 : stat64;
452
453     i = 0;          /* reset index to beginning of file */
454     ifd = -1;
455     if ((*statf)(file, &mbuf) < 0) {

```

```

456         if (statf == lstat64 || lstat64(file, &mbuf) < 0) {
457             int err = errno;
458             (void) printf(gettext("cannot open: %s\n"),
459                 strerror(err));
460             return (0);          /* POSIX.2 */
461         }
462     }
463     switch (mbuf.st_mode & S_IFMT) {
464     case S_IFREG:
465         if (iflg) {
466             (void) printf(gettext("regular file\n"));
467             return (0);
468         }
469         break;
470     case S_IFCHR:
471         (void) printf(gettext("character"));
472         goto spcl;
473
474     case S_IFDIR:
475         (void) printf(gettext("directory\n"));
476         return (0);
477
478     case S_IFIFO:
479         (void) printf(gettext("fifo\n"));
480         return (0);
481
482     case S_IFLNK:
483         if ((cc = readlink(file, buf, BUFSIZ)) < 0) {
484             int err = errno;
485             (void) printf(gettext("readlink error: %s\n"),
486                 strerror(err));
487             return (1);
488         }
489         buf[cc] = '\\0';
490         (void) printf(gettext("symbolic link to %s\n"), buf);
491         return (0);
492
493     case S_IFBLK:
494         (void) printf(gettext("block"));
495         /* major and minor, see sys/mkdev.h */
496     spcl:
497         (void) printf(gettext(" special (%d/%d)\n"),
498             major(mbuf.st_rdev), minor(mbuf.st_rdev));
499         return (0);
500
501     case S_IFSOCK:
502         (void) printf("socket\n");
503         /* FIXME, should open and try to getsockname. */
504         return (0);
505
506     case S_IFDOOR:
507         if (get_door_target(file, buf, sizeof (buf)) == 0)
508             (void) printf(gettext("door to %s\n"), buf);
509         else
510             (void) printf(gettext("door\n"));
511         return (0);
512     }
513
514     if (elf_version(EV_CURRENT) == EV_NONE) {
515         (void) printf(gettext("libelf is out of date\n"));
516         return (1);
517     }
518
519     ifd = open64(file, O_RDONLY);
520     if (ifd < 0) {

```

```

522         int err = errno;
523         (void) printf(gettext("cannot open: %s\n"), strerror(err));
524         return (0);          /* POSIX.2 */
525     }

527     /* need another fd for elf, since we might want to read the file too */
528     elffd = open64(file, O_RDONLY);
529     if (elffd < 0) {
530         int err = errno;
531         (void) printf(gettext("cannot open: %s\n"), strerror(err));
532         (void) close(elffd);
533         ifd = -1;
534         return (0);          /* POSIX.2 */
535     }
536     if ((fbsz = read(ifd, fbuf, FBSZ)) == -1) {
537         int err = errno;
538         (void) printf(gettext("cannot read: %s\n"), strerror(err));
539         (void) close(ifd);
540         ifd = -1;
541         return (0);          /* POSIX.2 */
542     }
543     if (fbsz == 0) {
544         (void) printf(gettext("empty file\n"));
545         fd_cleanup();
546         return (0);
547     }

549     /*
550     * First try user-specified position-dependent magic tests, if any,
551     * which need to execute before the default tests.
552     */
553     if ((mread = pread(ifd, (void*)magicbuf, (size_t)maxmagicoffset,
554         (off_t)0)) == -1) {
555         int err = errno;
556         (void) printf(gettext("cannot read: %s\n"), strerror(err));
557         fd_cleanup();
558         return (0);
559     }

561     /*
562     * Check against Magic Table entries.
563     * Check first magic table for magic tests to be applied
564     * before default tests.
565     * If no default tests are to be applied, all magic tests
566     * should occur in this magic table.
567     */
568     switch (f_ckmtab(magicbuf, mread, 1)) {
569     case -1:          /* Error */
570         exit(2);
571         break;
572     case 0:          /* Not magic */
573         break;
574     default:        /* Switch is magic index */
575         (void) putchar('\n');
576         fd_cleanup();
577         return (0);
578         /* NOTREACHED */
579         break;
580     }

582     if (dflg || !M_flg) {
583         /*
584         * default position-dependent tests,
585         * plus non-default magic tests, if any
586         */
587         switch (def_position_tests(file)) {

```

```

588         case -1:      /* error */
589             fd_cleanup();
590             return (1);
591         case 1:      /* matching type found */
592             fd_cleanup();
593             return (0);
594             /* NOTREACHED */
595             break;
596         case 0:      /* no matching type found */
597             break;
598     }
599     /* default context-sensitive tests */
600     def_context_tests();
601 } else {
602     /* no more tests to apply; no match was found */
603     (void) printf(gettext("data\n"));
604 }
605 fd_cleanup();
606 return (0);
607 }

609 /*
610 * def_position_tests() - applies default position-sensitive tests,
611 * looking for values in specific positions in the file.
612 * These are followed by default (followed by possibly some
613 * non-default) magic file tests.
614 *
615 * All position-sensitive tests, default or otherwise, must
616 * be applied before context-sensitive tests, to avoid
617 * false context-sensitive matches.
618 *
619 * Returns -1 on error which should result in error (non-zero)
620 * exit status for the file utility.
621 * Returns 0 if no matching file type found.
622 * Returns 1 if matching file type found.
623 */

625 static int
626 def_position_tests(char *file)
627 {
628     if (sccs()) { /* look for "lhdddd" where d is a digit */
629         (void) printf("sccs \n");
630         return (1);
631     }
632     if (fbuf[0] == '#' && fbuf[1] == '!' && shellsript(fbuf+2, &mbuf))
633         return (1);

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

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

645         if (corep->c_cmdname != '\0')
646             (void) printf(" from '%s'", corep->c_cmdname);
647         (void) putchar('\n');
648         return (1);
649     }

651     /*
652     * Runtime linker (ld.so.1) configuration file.
653     */

```

```

654     if (is_rtld_config())
655         return (1);

657     /*
658     * ZIP files, JAR files, and Java executables
659     */
660     if (zipfile(fbuf, ifd))
661         return (1);

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

666     /*
667     * Check against Magic Table entries.
668     * The magic entries checked here always start with default
669     * magic tests and may be followed by other, non-default magic
670     * tests.  If no default tests are to be executed, all the
671     * magic tests should have been in the first magic table.
672     */
673     switch (f_ckmtab(magicbuf, mread, 0)) {
674     case -1: /* Error */
675         exit(2);
676         break;
677     case 0: /* Not magic */
678         return (0);
679         /* NOTREACHED */
680         break;
681     default: /* Switch is magic index */

683         /*
684         * f_ckmtab recognizes file type,
685         * check if it is PostScript.
686         * if not, check if elf or a.out
687         */
688         if (magicbuf[0] == '%' && magicbuf[1] == '!') {
689             (void) putchar('\n');
690         } else {

692             /*
693             * Check that the file is executable (dynamic
694             * objects must be executable to be exec'ed,
695             * shared objects need not be, but by convention
696             * should be executable).
697             *
698             * Note that we should already have processed
699             * the file if it was an ELF file.
700             */
701             ar_coff_or_aout(elffd);
702             (void) putchar('\n');
703         }
704         return (1);
705         /* NOTREACHED */
706         break;
707     }

709     return (0); /* file was not identified */
710 }

712 /*
713 * def_context_tests() - default context-sensitive tests.
714 * These are the last tests to be applied.
715 * If no match is found, prints out "data".
716 */

718 static void
719 def_context_tests(void)

```

```

720 {
721     int    j;
722     int    nl;
723     char   ch;
724     int    len;

726     if (ccom() == 0)
727         goto notc;
728     while (fbuf[i] == '#') {
729         j = i;
730         while (fbuf[i++] != '\n') {
731             if (i - j > 255) {
732                 (void) printf(gettext("data\n"));
733                 return;
734             }
735             if (i >= fbsz)
736                 goto notc;
737         }
738         if (ccom() == 0)
739             goto notc;
740     }
741     check:
742     if (lookup(c) == 1) {
743         while ((ch = fbuf[i]) != ';' && ch != '{') {
744             if ((len = mblen(&fbuf[i], MB_CUR_MAX)) <= 0)
745                 len = 1;
746             i += len;
747             if (i >= fbsz)
748                 goto notc;
749         }
750         (void) printf(gettext("c program text"));
751         goto outa;
752     }
753     nl = 0;
754     while (fbuf[i] != '(') {
755         if (fbuf[i] <= 0)
756             goto notas;
757         if (fbuf[i] == ';'') {
758             i++;
759             goto check;
760         }
761         if (fbuf[i++] == '\n')
762             if (nl++ > 6)
763                 goto notc;
764         if (i >= fbsz)
765             goto notc;
766     }
767     while (fbuf[i] != ')') {
768         if (fbuf[i++] == '\n')
769             if (nl++ > 6)
770                 goto notc;
771         if (i >= fbsz)
772             goto notc;
773     }
774     while (fbuf[i] != '{') {
775         if ((len = mblen(&fbuf[i], MB_CUR_MAX)) <= 0)
776             len = 1;
777         if (fbuf[i] == '\n')
778             if (nl++ > 6)
779                 goto notc;
780         i += len;
781         if (i >= fbsz)
782             goto notc;
783     }
784     (void) printf(gettext("c program text"));
785     goto outa;

```

```

786 notc:
787     i = 0; /* reset to begining of file again */
788     while (fbuf[i] == 'c' || fbuf[i] == 'C' || fbuf[i] == '!' ||
789           fbuf[i] == '*' || fbuf[i] == '\n') {
790         while (fbuf[i++] != '\n')
791             if (i >= fbsz)
792                 goto notfort;
793     }
794     if (lookup(fort) == 1) {
795         (void) printf(gettext("fortran program text"));
796         goto outa;
797     }
798 notfort: /* looking for assembler program */
799     i = 0; /* reset to beginning of file again */
800     if (ccom() == 0) /* assembler programs may contain */
801         /* c-style comments */
802         goto notas;
803     if (ascom() == 0)
804         goto notas;
805     j = i - 1;
806     if (fbuf[i] == '.') {
807         i++;
808         if (lookup(as) == 1) {
809             (void) printf(gettext("assembler program text"));
810             goto outa;
811         } else if (j != -1 && fbuf[j] == '\n' && isalpha(fbuf[j + 2])) {
812             (void) printf(
813                 gettext("[nt]roff, tbl, or eqn input text"));
814             goto outa;
815         }
816     }
817     while (lookup(asc) == 0) {
818         if (ccom() == 0)
819             goto notas;
820         if (ascom() == 0)
821             goto notas;
822         while (fbuf[i] != '\n' && fbuf[i++] != ':') {
823             if (i >= fbsz)
824                 goto notas;
825         }
826         while (fbuf[i] == '\n' || fbuf[i] == ' ' || fbuf[i] == '\t')
827             if (i++ >= fbsz)
828                 goto notas;
829         j = i - 1;
830         if (fbuf[i] == '.') {
831             i++;
832             if (lookup(as) == 1) {
833                 (void) printf(
834                     gettext("assembler program text"));
835                 goto outa;
836             } else if (fbuf[j] == '\n' && isalpha(fbuf[j+2])) {
837                 (void) printf(
838                     gettext("[nt]roff, tbl, or eqn input "
839                         "text"));
840                 goto outa;
841             }
842         }
843     }
844     (void) printf(gettext("assembler program text"));
845     goto outa;
846 notas: /* start modification for multibyte env */
847     IS_ascii = 1;
848     if (fbsz < FBSZ)
849         Max = fbsz;
850     else

```

```

852     Max = FBSZ - MB_LEN_MAX; /* prevent cut of wchar read */
853     /* end modification for multibyte env */
854
855     for (i = 0; i < Max; /* null */)
856         if (fbuf[i] & 0200) {
857             IS_ascii = 0;
858             if (fbuf[0] == '\100' && fbuf[1] == '\357') {
859                 (void) printf(gettext("troff output\n"));
860                 return;
861             }
862             /* start modification for multibyte env */
863             if ((length = mbtowc(&wchar, &fbuf[i], MB_CUR_MAX))
864                 <= 0 || !iswprint(wchar)) {
865                 (void) printf(gettext("data\n"));
866                 return;
867             }
868             i += length;
869         }
870     else
871         i++;
872
873     i = fbsz; /* end modification for multibyte env */
874     if (mbuf.st_mode & (S_IXUSR|S_IXGRP|S_IXOTH))
875         (void) printf(gettext("commands text"));
876     else if (troffint(fbuf, fbsz))
877         (void) printf(gettext("troff intermediate output text"));
878     else if (english(fbuf, fbsz))
879         (void) printf(gettext("English text"));
880     else if (IS_ascii)
881         (void) printf(gettext("ascii text"));
882     else
883         (void) printf(gettext("text")); /* for multibyte env */
884 outa:
885     /*
886     * This code is to make sure that no MB char is cut in half
887     * while still being used.
888     */
889     fbsz = (fbsz < FBSZ ? fbsz : fbsz - MB_CUR_MAX + 1);
890     while (i < fbsz) {
891         if (isascii(fbuf[i])) {
892             i++;
893             continue;
894         } else {
895             if ((length = mbtowc(&wchar, &fbuf[i], MB_CUR_MAX))
896                 <= 0 || !iswprint(wchar)) {
897                 (void) printf(gettext(" with garbage\n"));
898                 return;
899             }
900             i = i + length;
901         }
902     }
903     (void) printf("\n");
904 }
905
906 static int
907 troffint(char *bp, int n)
908 {
909     int k;
910
911     i = 0;
912     for (k = 0; k < 6; k++) {
913         if (lookup(troff) == 0)
914             return (0);
915         if (lookup(troff) == 0)
916             return (0);
917         while (i < n && bp[i] != '\n')

```

```

918         i++;
919         if (i++ >= n)
920             return (0);
921     }
922     return (1);
923 }

925 static void
926 ar_coff_or_aout(int elffd)
927 {
928     Elf *elf;

930     /*
931      * Get the files elf descriptor and process it as an elf or
932      * a.out (4.x) file.
933      */

935     elf = elf_begin(elffd, ELF_C_READ, (Elf *)0);
936     switch (elf_kind(elf)) {
937     case ELF_K_AR :
938         (void) printf(gettext(", not a dynamic executable "
939             "or shared object"));
940         break;
941     case ELF_K_COFF:
942         (void) printf(gettext(", unsupported or unknown "
943             "file type"));
944         break;
945     default:
946         /*
947          * This is either an unknown file or an aout format
948          * At this time, we don't print dynamic/striped
949          * info. on a.out or non-Elf binaries.
950          */
951         break;
952     }
953     (void) elf_end(elf);
954 }

957 static void
958 print_elf_type(Elf_Info EI)
959 {
960     switch (EI.type) {
961     case ET_NONE:
962         (void) printf(" %s", gettext("unknown type"));
963         break;
964     case ET_REL:
965         (void) printf(" %s", gettext("relocatable"));
966         break;
967     case ET_EXEC:
968         (void) printf(" %s", gettext("executable"));
969         break;
970     case ET_DYN:
971         (void) printf(" %s", gettext("dynamic lib"));
972         break;
973     default:
974         break;
975     }
976 }

978 static void
979 print_elf_machine(int machine)
980 {
981     /*
982      * This table must be kept in sync with the EM_ constants
983      * in /usr/include/sys/elf.h.

```

```

984     */
985     static const char *mach_str[EM_NUM] = {
986         "unknown machine", /* 0 - EM_NONE */
987         "WE32100", /* 1 - EM_M32 */
988         "SPARC", /* 2 - EM_SPARC */
989         "80386", /* 3 - EM_386 */
990         "M68000", /* 4 - EM_68K */
991         "M88000", /* 5 - EM_88K */
992         "80486", /* 6 - EM_486 */
993         "i860", /* 7 - EM_860 */
994         "MIPS RS3000 Big-Endian", /* 8 - EM_MIPS */
995         "S/370", /* 9 - EM_S370 */
996         "MIPS RS3000 Little-Endian", /* 10 - EM_MIPS_RS3_LE */
997         "MIPS RS6000", /* 11 - EM_RS6000 */
998         NULL, /* 12 - EM_UNKNOWN12 */
999         NULL, /* 13 - EM_UNKNOWN13 */
1000        NULL, /* 14 - EM_UNKNOWN14 */
1001        "PA-RISC", /* 15 - EM_PA_RISC */
1002        "nCUBE", /* 16 - EM_nCUBE */
1003        "VPP500", /* 17 - EM_VPP500 */
1004        "SPARC32PLUS", /* 18 - EM_SPARC32PLUS */
1005        "i960", /* 19 - EM_960 */
1006        "PowerPC", /* 20 - EM_PPC */
1007        "PowerPC64", /* 21 - EM_PPC64 */
1008        "S/390", /* 22 - EM_S390 */
1009        NULL, /* 23 - EM_UNKNOWN23 */
1010        NULL, /* 24 - EM_UNKNOWN24 */
1011        NULL, /* 25 - EM_UNKNOWN25 */
1012        NULL, /* 26 - EM_UNKNOWN26 */
1013        NULL, /* 27 - EM_UNKNOWN27 */
1014        NULL, /* 28 - EM_UNKNOWN28 */
1015        NULL, /* 29 - EM_UNKNOWN29 */
1016        NULL, /* 30 - EM_UNKNOWN30 */
1017        NULL, /* 31 - EM_UNKNOWN31 */
1018        NULL, /* 32 - EM_UNKNOWN32 */
1019        NULL, /* 33 - EM_UNKNOWN33 */
1020        NULL, /* 34 - EM_UNKNOWN34 */
1021        NULL, /* 35 - EM_UNKNOWN35 */
1022        "V800", /* 36 - EM_V800 */
1023        "FR20", /* 37 - EM_FR20 */
1024        "RH32", /* 38 - EM_RH32 */
1025        "RCE", /* 39 - EM_RCE */
1026        "ARM", /* 40 - EM_ARM */
1027        "Alpha", /* 41 - EM_ALPHA */
1028        "S/390", /* 42 - EM_SH */
1029        "SPARCV9", /* 43 - EM_SPARCV9 */
1030        "Tricore", /* 44 - EM_TRICORE */
1031        "ARC", /* 45 - EM_ARC */
1032        "H8/300", /* 46 - EM_H8_300 */
1033        "H8/300H", /* 47 - EM_H8_300H */
1034        "H8S", /* 48 - EM_H8S */
1035        "H8/500", /* 49 - EM_H8_500 */
1036        "IA64", /* 50 - EM_IA_64 */
1037        "MIPS-X", /* 51 - EM_MIPS_X */
1038        "Coldfire", /* 52 - EM_COLDFIRE */
1039        "M68HC12", /* 53 - EM_68HC12 */
1040        "MMA", /* 54 - EM_MMA */
1041        "PCP", /* 55 - EM_PCP */
1042        "nCPU", /* 56 - EM_NCPU */
1043        "NDR1", /* 57 - EM_NDR1 */
1044        "Starcore", /* 58 - EM_STARCORE */
1045        "ME16", /* 59 - EM_ME16 */
1046        "ST100", /* 60 - EM_ST100 */
1047        "TINYJ", /* 61 - EM_TINYJ */
1048        "AMD64", /* 62 - EM_AMD64 */
1049        "PDSP", /* 63 - EM_PDSP */

```



```

1050     NULL,          /* 64 - EM_UNKNOWN64 */
1051     NULL,          /* 65 - EM_UNKNOWN65 */
1052     "FX66",        /* 66 - EM_FX66 */
1053     "ST9 PLUS",    /* 67 - EM_ST9PLUS */
1054     "ST7",          /* 68 - EM_ST7 */
1055     "68HC16",      /* 69 - EM_68HC16 */
1056     "68HC11",      /* 70 - EM_68HC11 */
1057     "68HC08",      /* 71 - EM_68HC08 */
1058     "68HC05",      /* 72 - EM_68HC05 */
1059     "SVX",         /* 73 - EM_SVX */
1060     "ST19",        /* 74 - EM_ST19 */
1061     "VAX",         /* 75 - EM_VAX */
1062     "CRIS",        /* 76 - EM_CRIS */
1063     "Javelin",     /* 77 - EM_JAVELIN */
1064     "Firepath",   /* 78 - EM_FIREPATH */
1065     "ZSP",         /* 79 - EM_ZSP */
1066     "MMIX",        /* 80 - EM_MMIX */
1067     "HUANY",       /* 81 - EM_HUANY */
1068     "Prism",       /* 82 - EM_PRISM */
1069     "AVR",         /* 83 - EM_AVR */
1070     "FR30",        /* 84 - EM_FR30 */
1071     "D10V",        /* 85 - EM_D10V */
1072     "D30V",        /* 86 - EM_D30V */
1073     "V850",        /* 87 - EM_V850 */
1074     "M32R",        /* 88 - EM_M32R */
1075     "MN10300",     /* 89 - EM_MN10300 */
1076     "MN10200",     /* 90 - EM_MN10200 */
1077     "picoJava",    /* 91 - EM_PJ */
1078     "OpenRISC",    /* 92 - EM_OPENRISC */
1079     "Tangent-A5", /* 93 - EM_ARC_A5 */
1080     "Xtensa",      /* 94 - EM_XTENSA */
1081     };
1082     /* If new machine is added, refuse to compile until we're updated */
1083     #if EM_NUM != 95
1084     #error "Number of known ELF machine constants has changed"
1085     #endif

1087     const char *str;

1089     if ((machine < EM_NONE) || (machine >= EM_NUM))
1090         machine = EM_NONE;

1092     str = mach_str[machine];
1093     if (str)
1094         (void) printf(" %s", str);
1095 }

1097 static void
1098 print_elf_datatype(int datatype)
1099 {
1100     switch (datatype) {
1101     case ELFDATA2LSB:
1102         (void) printf(" LSB");
1103         break;
1104     case ELFDATA2MSB:
1105         (void) printf(" MSB");
1106         break;
1107     default:
1108         break;
1109     }
1110 }

1112 static void
1113 print_elf_class(int class)
1114 {
1115     switch (class) {

```

```

1116     case ELFCLASS32:
1117         (void) printf(" %s", gettext("32-bit"));
1118         break;
1119     case ELFCLASS64:
1120         (void) printf(" %s", gettext("64-bit"));
1121         break;
1122     default:
1123         break;
1124     }
1125 }

1127 static void
1128 print_elf_flags(Elf_Info EI)
1129 {
1130     unsigned int flags;

1132     flags = EI.flags;
1133     switch (EI.machine) {
1134     case EM_SPARCV9:
1135         if (flags & EF_SPARC_EXT_MASK) {
1136             if (flags & EF_SPARC_SUN_US3) {
1137                 (void) printf("%s", gettext(
1138                     " UltraSPARC3 Extensions Required"));
1139             } else if (flags & EF_SPARC_SUN_US1) {
1140                 (void) printf("%s", gettext(
1141                     " UltraSPARC1 Extensions Required"));
1142             }
1143             if (flags & EF_SPARC_HAL_R1)
1144                 (void) printf("%s", gettext(
1145                     " HaL R1 Extensions Required"));
1146         }
1147         break;
1148     case EM_SPARC32PLUS:
1149         if (flags & EF_SPARC_32PLUS)
1150             (void) printf("%s", gettext(" V8+ Required"));
1151         if (flags & EF_SPARC_SUN_US3) {
1152             (void) printf("%s",
1153                 gettext(" UltraSPARC3 Extensions Required"));
1154         } else if (flags & EF_SPARC_SUN_US1) {
1155             (void) printf("%s",
1156                 gettext(" UltraSPARC1 Extensions Required"));
1157         }
1158         if (flags & EF_SPARC_HAL_R1)
1159             (void) printf("%s",
1160                 gettext(" HaL R1 Extensions Required"));
1161         break;
1162     default:
1163         break;
1164     }
1165 }

1167 /*
1168 * check_ident: checks the ident field of the presumably
1169 * elf file. If check fails, this is not an
1170 * elf file.
1171 */
1172 static int
1173 check_ident(unsigned char *ident, int fd)
1174 {
1175     int class;
1176     if (pread64(fd, ident, EI_NIDENT, 0) != EI_NIDENT)
1177         return (ELF_READ_FAIL);
1178     class = ident[EI_CLASS];
1179     if (class != ELFCLASS32 && class != ELFCLASS64)
1180         return (ELF_READ_FAIL);
1181     if (ident[EI_MAG0] != ELF_MAG0 || ident[EI_MAG1] != ELF_MAG1 ||

```

```

1182         ident[EI_MAG2] != ELF_MAG2 || ident[EI_MAG3] != ELF_MAG3)
1183         return (ELF_READ_FAIL);

1185     return (ELF_READ_OKAY);
1186 }

1188 static int
1189 elf_check(char *file)
1190 {
1191     Elf_Info EInfo;
1192     int class, version, format;
1193     unsigned char ident[EI_NIDENT];

1195     (void) memset(&EInfo, 0, sizeof (Elf_Info));
1196     EInfo.file = file;

1198     /*
1199     * Verify information in file identifier.
1200     * Return quietly if not elf; Different type of file.
1201     */
1202     if (check_ident(ident, elffd) == ELF_READ_FAIL)
1203         return (1);

1205     /*
1206     * Read the elf headers for processing and get the
1207     * get the needed information in Elf_Info struct.
1208     */
1209     class = ident[EI_CLASS];
1210     if (class == ELFCLASS32) {
1211         if (elf_read32(elffd, &EInfo) == ELF_READ_FAIL) {
1212             (void) fprintf(stderr, gettext("%s: %s: can't "
1213             "read ELF header\n"), File, file);
1214             return (1);
1215         }
1216     } else if (class == ELFCLASS64) {
1217         if (elf_read64(elffd, &EInfo) == ELF_READ_FAIL) {
1218             (void) fprintf(stderr, gettext("%s: %s: can't "
1219             "read ELF header\n"), File, file);
1220             return (1);
1221         }
1222     } else {
1223         /* something wrong */
1224         return (1);
1225     }

1227     /* version not in ident then 1 */
1228     version = ident[EI_VERSION] ? ident[EI_VERSION] : 1;

1230     format = ident[EI_DATA];
1231     (void) printf("%s", gettext("ELF"));
1232     print_elf_class(class);
1233     print_elf_datatype(format);
1234     print_elf_type(EInfo);

1236     if (EInfo.core_type != EC_NOTCORE) {
1237         /* Print what kind of core is this */
1238         if (EInfo.core_type == EC_OLDCORE)
1239             (void) printf(" %s", gettext("pre-2.6 core file"));
1240         else
1241             (void) printf(" %s", gettext("core file"));
1242     }

1244     /* Print machine info */
1245     print_elf_machine(EInfo.machine);

1247     /* Print Version */

```

```

1248     if (version == 1)
1249         (void) printf(" %s %d", gettext("Version"), version);

1251     /* Print Flags */
1252     print_elf_flags(EInfo);

1254     /* Last bit, if it is a core */
1255     if (EInfo.core_type != EC_NOTCORE) {
1256         /* Print the program name that dumped this core */
1257         (void) printf(gettext(", from '%s'"), EInfo.fname);
1258         return (0);
1259     }

1261     /* Print Capabilities */
1262     if (EInfo.cap_str[0] != '\0')
1263         (void) printf(" [%s]", EInfo.cap_str);

1265     if ((EInfo.type != ET_EXEC) && (EInfo.type != ET_DYN))
1266         return (0);

1268     /* Print if it is dynamically linked */
1269     if (EInfo.dynamic)
1270         (void) printf(gettext(", dynamically linked"));
1271     else
1272         (void) printf(gettext(", statically linked"));

1274     /* Print if it is stripped */
1275     if (EInfo.stripped & E_SYMTAB) {
1276         (void) printf(gettext(", not stripped"));
1277         if (!(EInfo.stripped & E_DBGINF)) {
1278             (void) printf(gettext(
1279             ", no debugging information available"));
1280         }
1281     } else {
1282         (void) printf(gettext(", stripped"));
1283     }

1285     return (0);
1286 }

1288 /*
1289 * is_rtld_config - If file is a runtime linker config file, prints
1290 * the description and returns True (1). Otherwise, silently returns
1291 * False (0).
1292 */
1293 int
1294 is_rtld_config(void)
1295 {
1296     Rtc_id *id;

1298     if ((fbsz >= sizeof (*id)) && RTC_ID_TEST(fbuf)) {
1299         (void) printf(gettext("Runtime Linking Configuration"));
1300         id = (Rtc_id *) fbuf;
1301         print_elf_class(id->id_class);
1302         print_elf_datatype(id->id_data);
1303         print_elf_machine(id->id_machine);
1304         (void) printf("\n");
1305         return (1);
1306     }

1308     return (0);
1309 }

1311 /*
1312 * lookup -
1313 * Attempts to match one of the strings from a list, 'tab',

```

```

1314 * with what is in the file, starting at the current index position 'i'.
1315 * Looks past any initial whitespace and expects whitespace or other
1316 * delimiting characters to follow the matched string.
1317 * A match identifies the file as being 'assembler', 'fortran', 'c', etc.
1318 * Returns 1 for a successful match, 0 otherwise.
1319 */
1320 static int
1321 lookup(char **tab)
1322 {
1323     register char    r;
1324     register int     k, j, l;

1326     while (fbuf[i] == ' ' || fbuf[i] == '\t' || fbuf[i] == '\n')
1327         i++;
1328     for (j = 0; tab[j] != 0; j++) {
1329         l = 0;
1330         for (k = i; ((r = tab[j][l++]) == fbuf[k] && r != '\0'); k++)
1331             ;
1332         if (r == '\0')
1333             if (fbuf[k] == ' ' || fbuf[k] == '\n' ||
1334                 fbuf[k] == '\t' || fbuf[k] == '{' ||
1335                 fbuf[k] == '/') {
1336                 i = k;
1337                 return (1);
1338             }
1339     }
1340     return (0);
1341 }

1343 /*
1344 * ccom -
1345 * Increments the current index 'i' into the file buffer 'fbuf' past any
1346 * whitespace lines and C-style comments found, starting at the current
1347 * position of 'i'. Returns 1 as long as we don't increment i past the
1348 * size of fbuf (fbsz). Otherwise, returns 0.
1349 */

1351 static int
1352 ccom(void)
1353 {
1354     register char    cc;
1355     int              len;

1357     while ((cc = fbuf[i]) == ' ' || cc == '\t' || cc == '\n')
1358         if (i++ >= fbsz)
1359             return (0);
1360     if (fbuf[i] == '/' && fbuf[i+1] == '*') {
1361         i += 2;
1362         while (fbuf[i] != '*' || fbuf[i+1] != '/') {
1363             if (fbuf[i] == '\\')
1364                 i++;
1365             if ((len = mblen(&fbuf[i], MB_CUR_MAX)) <= 0)
1366                 len = 1;
1367             i += len;
1368             if (i >= fbsz)
1369                 return (0);
1370         }
1371         if ((i += 2) >= fbsz)
1372             return (0);
1373     }
1374     if (fbuf[i] == '\n')
1375         if (ccom() == 0)
1376             return (0);
1377     return (1);
1378 }

```

```

1380 /*
1381 * ascom -
1382 * Increments the current index 'i' into the file buffer 'fbuf' past
1383 * consecutive assembler program comment lines starting with ASCOMCHAR,
1384 * starting at the current position of 'i'.
1385 * Returns 1 as long as we don't increment i past the
1386 * size of fbuf (fbsz). Otherwise returns 0.
1387 */

1389 static int
1390 ascom(void)
1391 {
1392     while (fbuf[i] == ASCOMCHAR) {
1393         i++;
1394         while (fbuf[i++] != '\n')
1395             if (i >= fbsz)
1396                 return (0);
1397         while (fbuf[i] == '\n')
1398             if (i++ >= fbsz)
1399                 return (0);
1400     }
1401     return (1);
1402 }

1404 static int
1405 sccs(void)
1406 {
1407     register int j;
1408     /* look for "lhdddd" where d is a digit */

1409     if (fbuf[0] == 1 && fbuf[1] == 'h') {
1410         for (j = 2; j <= 6; j++) {
1411             if (isdigit(fbuf[j]))
1412                 continue;
1413             else
1414                 return (0);
1415         }
1416     } else {
1417         return (0);
1418     }
1419     return (1);
1420 }

1422 static int
1423 english(char *bp, int n)
1424 {
1425     #define NASC 128 /* number of ascii char ?? */
1426     register int    j, vow, freq, rare, len;
1427     register int    badpun = 0, punct = 0;
1428     int             ct[NASC];

1430     if (n < 50)
1431         return (0); /* no point in statistics on squibs */
1432     for (j = 0; j < NASC; j++)
1433         ct[j] = 0;
1434     for (j = 0; j < n; j += len) {
1435         if ((unsigned char)bp[j] < NASC)
1436             ct[bp[j]|040]++;
1437         switch (bp[j]) {
1438             case '.':
1439             case ',':
1440             case ')':
1441             case '%':
1442             case ';':
1443             case ':':
1444             case '?':
1445                 punct++;

```

```

1446         if (j < n-1 && bp[j+1] != ' ' && bp[j+1] != '\n')
1447             badpun++;
1448     }
1449     if ((len = mblen(&bp[j], MB_CUR_MAX)) <= 0)
1450         len = 1;
1451 }
1452 if (badpun*5 > punct)
1453     return (0);
1454 vow = ct['a'] + ct['e'] + ct['i'] + ct['o'] + ct['u'];
1455 freq = ct['e'] + ct['t'] + ct['a'] + ct['i'] + ct['o'] + ct['n'];
1456 rare = ct['v'] + ct['j'] + ct['k'] + ct['q'] + ct['x'] + ct['z'];
1457 if (2*ct[';'] > ct['e'])
1458     return (0);
1459 if ((ct['>'] + ct['<'] + ct['/']) > ct['e'])
1460     return (0); /* shell file test */
1461 return (vow * 5 >= n - ct[' '] && freq >= 10 * rare);
1462 }

1465 static int
1466 shellsript(char buf[], struct stat64 *sb)
1467 {
1468     char *tp, *cp, *xp, *up, *gp;
1469
1470     cp = strchr(buf, '\n');
1471     if (cp == NULL || cp - fbuf > fbsz)
1472         return (0);
1473     for (tp = buf; tp != cp && isspace((unsigned char)*tp); tp++)
1474         if (!isascii(*tp))
1475             return (0);
1476     for (xp = tp; tp != cp && !isspace((unsigned char)*tp); tp++)
1477         if (!isascii(*tp))
1478             return (0);
1479     if (tp == xp)
1480         return (0);
1481     if (sb->st_mode & S_ISUID)
1482         up = gettext("set-uid ");
1483     else
1484         up = "";
1485
1486     if (sb->st_mode & S_ISGID)
1487         gp = gettext("set-gid ");
1488     else
1489         gp = "";
1490
1491     if (strncmp(xp, "/bin/sh", tp - xp) == 0)
1492         xp = gettext("shell");
1493     else if (strncmp(xp, "/bin/csh", tp - xp) == 0)
1494         xp = gettext("c-shell");
1495     else if (strncmp(xp, "/usr/sbin/dtrace", tp - xp) == 0)
1496         xp = gettext("DTrace");
1497     else
1498         *tp = '\0';
1499
1500     /* TRANSLATION NOTE
1501     * This message is printed by file command for shell scripts.
1502     * The first %s is for the translation for "set-uid " (if the script
1503     * has the set-uid bit set), or is for an empty string (if the
1504     * script does not have the set-uid bit set).
1505     * Similarly, the second %s is for the translation for "set-gid ",
1506     * or is for an empty string.
1507     * The third %s is for the translation for either: "shell", "c-shell",
1508     * or "DTrace", or is for the pathname of the program the script
1509     * executes.
1510     */
1511     (void) printf(gettext("%s%sexecutable %s script\n"), up, gp, xp);

```

```

1512         return (1);
1513     }

1515 static int
1516 get_door_target(char *file, char *buf, size_t bufsize)
1517 {
1518     int fd;
1519     door_info_t di;
1520     psinfo_t psinfo;

1522     if ((fd = open64(file, O_RDONLY)) < 0 ||
1523         door_info(fd, &di) != 0) {
1524         if (fd >= 0)
1525             (void) close(fd);
1526         return (-1);
1527     }
1528     (void) close(fd);

1530     (void) sprintf(buf, "/proc/%ld/psinfo", di.di_target);
1531     if ((fd = open64(buf, O_RDONLY)) < 0 ||
1532         read(fd, &psinfo, sizeof (psinfo)) != sizeof (psinfo)) {
1533         if (fd >= 0)
1534             (void) close(fd);
1535         return (-1);
1536     }
1537     (void) close(fd);

1539     (void) snprintf(buf, bufsize, "%s[%ld]", psinfo.pr_fname, di.di_target);
1540     return (0);
1541 }

1543 /*
1544  * ZIP file header information
1545  */
1546 #define SIGSZ          4
1547 #define LOCSIG         "PK\003\004"
1548 #define LOCHDRSIZ     30

1550 #define CH(b, n)       (((unsigned char *) (b))[n])
1551 #define SH(b, n)       (CH(b, n) | (CH(b, n+1) << 8))
1552 #define LG(b, n)       (SH(b, n) | (SH(b, n+2) << 16))

1554 #define LOCNAM(b)      (SH(b, 26)) /* filename size */
1555 #define LOCEXT(b)      (SH(b, 28)) /* extra field size */

1557 #define XFHSIZ         4 /* header id, data size */
1558 #define XFHDID(b)      (SH(b, 0)) /* extract field header id */
1559 #define XFDATASIZ(b)   (SH(b, 2)) /* extract field data size */
1560 #define XFJAVASIG     0xcafe /* java executables */

1562 static int
1563 zipfile(char *fbuf, int fd)
1564 {
1565     off_t xoff, xoff_end;

1567     if (strncmp(fbuf, LOCSIG, SIGSZ) != 0)
1568         return (0);

1570     xoff = LOCHDRSIZ + LOCNAM(fbuf);
1571     xoff_end = xoff + LOCEXT(fbuf);

1573     while (xoff < xoff_end) {
1574         char xfhdr[XFHSIZ];

1576         if (pread(fd, xfhdr, XFHSIZ, xoff) != XFHSIZ)
1577             break;

```

```

1579         if (XFHID(xfhdr) == XFJAVASIG) {
1580             (void) printf("%s\n", gettext("java archive file"));
1581             return (1);
1582         }
1583         xoff += sizeof (xfhdr) + XFDATASIZ(xfhdr);
1584     }
1585
1586     /*
1587     * We could just print "ZIP archive" here.
1588     *
1589     * However, customers may be using their own entries in
1590     * /etc/magic to distinguish one kind of ZIP file from another, so
1591     * let's defer the printing of "ZIP archive" to there.
1592     */
1593     return (0);
1594 }
1595
1596 static int
1597 is_crash_dump(const char *buf, int fd)
1598 {
1599     /* LINTED: pointer cast may result in improper alignment */
1600     const dumphdr_t *dhp = (const dumphdr_t *)buf;
1601
1602     /*
1603     * The current DUMP_MAGIC string covers Solaris 7 and later releases.
1604     * The utsname struct is only present in dumphdr_t's with dump_version
1605     * greater than or equal to 9.
1606     */
1607     if (dhp->dump_magic == DUMP_MAGIC) {
1608         print_dumphdr(fd, dhp, return_uint32, NATIVE_ISA);
1609     } else if (dhp->dump_magic == swap_uint32(DUMP_MAGIC)) {
1610         print_dumphdr(fd, dhp, swap_uint32, OTHER_ISA);
1611     } else if (dhp->dump_magic == OLD_DUMP_MAGIC ||
1612         dhp->dump_magic == swap_uint32(OLD_DUMP_MAGIC)) {
1613         char *isa = (dhp->dump_magic == OLD_DUMP_MAGIC ?
1614             NATIVE_ISA : OTHER_ISA);
1615         (void) printf(gettext("SunOS 32-bit %s crash dump\n"), isa);
1616     } else {
1617         return (0);
1618     }
1619
1620     return (1);
1621 }
1622
1623
1624 }
1625
1626 static void
1627 print_dumphdr(const int fd, const dumphdr_t *dhp, uint32_t (*swap)(uint32_t),
1628     const char *isa)
1629 {
1630     dumphdr_t dh;
1631
1632     /*
1633     * A dumphdr_t is bigger than FBSZ, so we have to manually read the
1634     * rest of it.
1635     */
1636     if (swap(dhp->dump_version) > 8 && pread(fd, &dh, sizeof (dumphdr_t),
1637         (off_t)0) == sizeof (dumphdr_t)) {
1638         const char *c = swap(dh.dump_flags) & DF_COMPRESSED ?
1639             "compressed " : "";
1640         const char *l = swap(dh.dump_flags) & DF_LIVE ?
1641             "live" : "crash";
1642
1643         (void) printf(gettext(

```

```

1644         "%s %s %s %u-bit %s %s%s dump from '%s'\n"),
1645         dh.dump_utsname.sysname, dh.dump_utsname.release,
1646         dh.dump_utsname.version, swap(dh.dump_wordsize), isa,
1647         c, l, dh.dump_utsname.nodename);
1648     } else {
1649         (void) printf(gettext("SunOS %u-bit %s crash dump\n"),
1650             swap(dhp->dump_wordsize), isa);
1651     }
1652 }
1653
1654 static void
1655 usage(void)
1656 {
1657     (void) fprintf(stderr, gettext(
1658         "usage: file [-bdh] [-M mfile] [-m mfile] [-f ffile] file ...\n"
1659         "          file [-bdh] [-M mfile] [-m mfile] -f ffile\n"
1660         "          file -i [-bh] [-f ffile] file ...\n"
1661         "          file -i [-bh] -f ffile\n"
1662         "usage: file [-dh] [-M mfile] [-m mfile] [-f ffile] file ...\n"
1663         "          file [-dh] [-M mfile] [-m mfile] -f ffile\n"
1664         "          file -i [-h] [-f ffile] file ...\n"
1665         "          file -i [-h] -f ffile\n"
1666         "          file -c [-d] [-M mfile] [-m mfile]\n"));
1667     exit(2);
1668 }

```

unchanged_portion_omitted

8986 Sat Apr 12 09:17:29 2014

new/usr/src/man/man1/file.1

4732 /usr/bin/file should provide -b option for compatibility with GNU/BSD file

```

1  \" te
2  .\" Copyright 1989 AT&T Copyright (c) 1992, X/Open Company Limited All Rights Re
3  .\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
4  .\" http://www.opengroup.org/bookstore/.
5  .\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
6  .\" This notice shall appear on any product containing this material.
7  .\" The contents of this file are subject to the terms of the Common Development
8  .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
9  .\" When distributing Covered Code, include this CDDL HEADER in each file and in
10 .TH FILE 1 "Apr 11, 2014"
10 .TH FILE 1 "Sep 10, 2013"
11 .SH NAME
12 file \- determine file type
13 .SH SYNOPSIS
14 .LP
15 .nf
16 \fB/usr/bin/file\fR [\fB-bdh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\fR] [
16 \fB/usr/bin/file\fR [\fB-dh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\fR] [
17 .fi

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

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

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

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

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

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

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

```

```

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

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

64 .SH DESCRIPTION
65 .sp
66 .LP
67 The \fBfile\fR utility performs a series of tests on each file supplied by
68 \fIfile\fR and, optionally, on each file listed in \fIffile\fR in an attempt to
69 classify it. If the file is not a regular file, its file type is identified.
70 The file types directory, \fBFBFIFO\fR, block special, and character special are
71 identified as such. If the file is a regular file and the file is zero-length,
72 it is identified as an empty file.
73 .sp
74 .LP
75 If \fIfile\fR appears to be a text file, \fBfile\fR examines the first 512
76 bytes and tries to determine its programming language. If \fIfile\fR is a
77 symbolic link, by default the link is followed and \fBfile\fR tests the file to
78 which the symbolic link refers.
79 .sp
80 .LP
81 If \fIfile\fR is a relocatable object, executable, or shared object, \fBfile\fR
82 prints out information about the file's execution requirements. This
83 information includes the machine class, byte-ordering, static/dynamic linkage,
84 and any software or hardware capability requirements. If \fIfile\fR is a
85 runtime linking configuration file, \fBfile\fR prints information about the
86 target platform, including the machine class and byte-ordering.
87 .sp
88 .LP
89 By default, \fBfile\fR will try to use the localized magic file
90 \fB/usr/lib/locale/\fIlocale\fR/LC_MESSAGES/magic\fR, if it exists, to identify
91 files that have a magic number. For example, in the Japanese locale, \fBfile\fR
92 will try to use \fB/usr/lib/locale/ja/LC_MESSAGES/magic\fR. If a localized
93 magic file does not exist, \fBfile\fR will utilize \fB/etc/magic\fR. A magic
94 number is a numeric or string constant that indicates the file type. See
95 \fBmagic\fR(4) for an explanation of the format of \fB/etc/magic\fR.
96 .sp
97 .LP
98 If \fIfile\fR does not exist, cannot be read, or its file status could not be
99 determined, it is not considered an error that affects the exit status. The
100 output will indicate that the file was processed, but that its type could not
101 be determined.
102 .SH OPTIONS
103 .sp
104 .LP
105 The following options are supported:
106 .sp
107 .ne 2
108 .na
109 \fB-fb-b\fR
110 .ad
111 .RS 12n
112 Be brief, do not print leading filename.
113 .RE

115 .sp
116 .ne 2
117 .na
118 #endif /* ! codereview */

```

```

119 \fB\fB-c\fR\fR
120 .ad
121 .RS 12n
122 Checks the magic file for format errors. For reasons of efficiency, this
123 validation is normally not carried out.
124 .RE

126 .sp
127 .ne 2
128 .na
129 \fB\fB-d\fR\fR
130 .ad
131 .RS 12n
132 Applies any position-sensitive and context-sensitive default system tests to
133 the file.
134 .RE

136 .sp
137 .ne 2
138 .na
139 \fB\fB-f\fR \fIfifile\fR\fR
140 .ad
141 .RS 12n
142 \fIfifile\fR contains a list of the files to be examined.
143 .RE

145 .sp
146 .ne 2
147 .na
148 \fB\fB-h\fR\fR
149 .ad
150 .RS 12n
151 When a symbolic link is encountered, this option identifies the file as a
152 symbolic link. If \fB-h\fR is not specified and \fIfifile\fR is a symbolic link
153 that refers to a non-existent file, the \fBfile\fR utility identifies the file
154 as a symbolic link, as if \fB-h\fR had been specified.
155 .RE

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

167 .sp
168 .ne 2
169 .na
170 \fB\fB-m\fR \fImfile\fR\fR
171 .ad
172 .RS 12n
173 .sp
174 .ne 2
175 .na
176 \fB\fB/usr/bin/file\fR\fR
177 .ad
178 .RS 22n
179 Uses \fImfile\fR as an alternate magic file, instead of \fB/etc/magic\fR.
180 .RE

182 .sp
183 .ne 2
184 .na

```

```

185 \fB\fB/usr/xpg4/bin/file\fR\fR
186 .ad
187 .RS 22n
188 Specifies the name of a file containing position-sensitive tests that are
189 applied to a file in order to classify it (see \fBmagic\fR(4)). If the \fB-m\fR
190 option is specified without specifying the \fB-d\fR option or the \fB-M\fR
191 option, position-sensitive default system tests are applied after the
192 position-sensitive tests specified by the \fB-m\fR option.
193 .RE

195 .RE

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

209 .sp
210 .LP
211 If the \fB-M\fR option is specified with the \fB-d\fR option, the \fB-m\fR
212 option, or both, or if the \fB-m\fR option is specified with the \fB-d\fR
213 option, the concatenation of the position-sensitive tests specified by these
214 options is applied in the order specified by the appearance of these options.
215 .SH OPERANDS
216 .sp
217 .LP
218 The following operands are supported:
219 .sp
220 .ne 2
221 .na
222 \fB\fIfifile\fR\fR
223 .ad
224 .RS 8n
225 A path name of a file to be tested.
226 .RE

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

240 .sp
241 .in +2
242 .nf
243 file "$1" | grep \(\miFq executable &&
244         printf "%s is executable.\n" "$1"
245 .fi
246 .in -2
247 .sp

249 .SH ENVIRONMENT VARIABLES
250 .sp

```

```

251 .LP
252 See \fBenviron\fR(5) for descriptions of the following environment variables
253 that affect the execution of \fBfile\fR: \fBBLANG\fR, \fBLC_ALL\fR,
254 \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
255 .SH EXIT STATUS
256 .sp
257 .LP
258 The following exit values are returned:
259 .sp
260 .ne 2
261 .na
262 \fB\FB0\fR
263 .ad
264 .RS 6n
265 Successful completion.
266 .RE

268 .sp
269 .ne 2
270 .na
271 \fB\FB>0\fR
272 .ad
273 .RS 6n
274 An error occurred.
275 .RE

277 .SH FILES
278 .sp
279 .ne 2
280 .na
281 \fB\FB/etc/magic\fR
282 .ad
283 .RS 14n
284 \fBfile\fR's magic number file
285 .RE

287 .SH ATTRIBUTES
288 .sp
289 .LP
290 See \fBattributes\fR(5) for descriptions of the following attributes:
291 .sp

293 .sp
294 .TS
295 box;
296 c | c
297 l | l .
298 ATTRIBUTE TYPE ATTRIBUTE VALUE
299 -
300 CSI Enabled
301 -
302 Interface Stability Standard
303 .TE

305 .SH SEE ALSO
306 .sp
307 .LP
308 \fBcrle\fR(1), \fBelfdump\fR(1), \fBls\fR(1), \fBmagic\fR(4),
309 \fBattributes\fR(5), \fBenviron\fR(5), \fBlargefile\fR(5), \fBstandards\fR(5)

```