```
**********************************************************
  117214 Tue May 26 15:56:27 2015
new/usr/src/cmd/fs.d/ufs/fsdb/fsdb.c
5396 gcc 4.8.2 longjmp errors for cscope-fast
**********************************************************
```

```
   1 /*
   2  * Copyright 2015 Gary Mills
   3  * Copyright (c) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
   4  */

   6 /*
   7  * Copyright (c) 1988 Regents of the University of California.
   8  * All rights reserved.
   9  *
  10  * This code is derived from software contributed to Berkeley by
  11  * Computer Consoles Inc.
  12  *
  13  * Redistribution and use in source and binary forms are permitted
  14  * provided that: (1) source distributions retain this entire copyright
  15  * notice and comment, and (2) distributions including binaries display
  16  * the following acknowledgement:  ''This product includes software
  17  * developed by the University of California, Berkeley and its contributors''
  18  * in the documentation or other materials provided with the distribution
  19  * and in all advertising materials mentioning features or use of this
  20  * software. Neither the name of the University nor the names of its
  21  * contributors may be used to endorse or promote products derived
  22  * from this software without specific prior written permission.
  23  * THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
  24  * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
  25  * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
  26  */

  28 #ifndef lint
  29 char copyright[] =
  30 "@(#) Copyright(c) 1988 Regents of the University of California.\n\
  31 All rights reserved.\n";
  32 #endif /* not lint */

  34 #ifndef lint
  35 static char sccsid[] = "@(#)fsdb.c      5.8 (Berkeley) 6/1/90";
  36 #endif /* not lint */

  38 /*
  39  *  fsdb - file system debugger
  40  *
  41  *  usage: fsdb [-o suboptions] special
  42  *  options/suboptions:
  43  *      -o
  44  *              ?               display usage
  45  *              o               override some error conditions
  46  *              p="string"      set prompt to string
  47  *              w               open for write
  48  */

  50 #include <sys/param.h>
  51 #include <sys/signal.h>
  52 #include <sys/file.h>
  53 #include <inttypes.h>
  54 #include <sys/sysmacros.h>

  56 #ifdef sun
  57 #include <unistd.h>
  58 #include <stdlib.h>
  59 #include <string.h>
  60 #include <fcntl.h>
  61 #include <signal.h>
```



```
  62 #include <sys/types.h>
  63 #include <sys/vnode.h>
  64 #include <sys/mntent.h>
  65 #include <sys/wait.h>
  66 #include <sys/fs/ufs_fsdir.h>
  67 #include <sys/fs/ufs_fs.h>
  68 #include <sys/fs/ufs_inode.h>
  69 #include <sys/fs/ufs_acl.h>
  70 #include <sys/fs/ufs_log.h>
  71 #else
  72 #include <sys/dir.h>
  73 #include <ufs/fs.h>
  74 #include <ufs/dinode.h>
  75 #include <paths.h>
  76 #endif /* sun */

  78 #include <stdio.h>
  79 #include <setjmp.h>

  81 #define OLD_FSDB_COMPATIBILITY  /* To support the obsoleted "-z" option */

  83 #ifndef _PATH_BSHELL
  84 #define _PATH_BSHELL    "/bin/sh"
  85 #endif /* _PATH_BSHELL */
  86 /*
  87  * Defines from the 4.3-tahoe file system, for systems with the 4.2 or 4.3
  88  * file system.
  89  */
  90 #ifndef FS_42POSTBLFMT
  91 #define cg_blktot(cgp) (((cgp))->cg_btot)
  92 #define cg_blks(fs, cgp, cylno) (((cgp))->cg_b[cylno])
  93 #define cg_inosused(cgp) (((cgp))->cg_iused)
  94 #define cg_blksfree(cgp) (((cgp))->cg_free)
  95 #define cg_chkmagic(cgp) ((cgp)->cg_magic == CG_MAGIC)
  96 #endif

  98 /*
  99  * Never changing defines.
 100  */
 101 #define OCTAL           8               /* octal base */
 102 #define DECIMAL         10              /* decimal base */
 103 #define HEX             16              /* hexadecimal base */

 105 /*
 106  * Adjustable defines.
 107  */
 108 #define NBUF            10              /* number of cache buffers */
 109 #define PROMPTSIZE      80              /* size of user definable prompt */
 110 #define MAXFILES        40000           /* max number of files ls can handle */
 111 #define FIRST_DEPTH     10              /* default depth for find and ls */
 112 #define SECOND_DEPTH    100             /* second try at depth (maximum) */
 113 #define INPUTBUFFER     1040            /* size of input buffer */
 114 #define BYTESPERLINE    16              /* bytes per line of /dxo output */
 115 #define NREG            36              /* number of save registers */

 117 #define DEVPREFIX       "/dev/"         /* Uninteresting part of "special" */

 119 #if defined(OLD_FSDB_COMPATIBILITY)
 120 #define FSDB_OPTIONS    "o:wp:z:"
 121 #else
 122 #define FSDB_OPTIONS    "o:wp:"
 123 #endif /* OLD_FSDB_COMPATIBILITY */


 126 /*
 127  * Values dependent on sizes of structs and such.
```

```
 128  */
 129 #define NUMB            3                       /* these three are arbitrary, */
 130 #define BLOCK           5                       /* but must be different from */
 131 #define FRAGMENT        7                       /* the rest (hence odd). */
 132 #define BITSPERCHAR     8                       /* couldn't find it anywhere  */
 133 #define CHAR            (sizeof (char))
 134 #define SHORT           (sizeof (short))
 135 #define LONG            (sizeof (long))
 136 #define U_OFFSET_T      (sizeof (u_offset_t))   /* essentially "long long" */
 137 #define INODE           (sizeof (struct dinode))
 138 #define DIRECTORY       (sizeof (struct direct))
 139 #define CGRP            (sizeof (struct cg))
 140 #define SB              (sizeof (struct fs))
 141 #define BLKSIZE         (fs->fs_bsize)          /* for clarity */
 142 #define FRGSIZE         (fs->fs_fsize)
 143 #define BLKSHIFT        (fs->fs_bshift)
 144 #define FRGSHIFT        (fs->fs_fshift)
 145 #define SHADOW_DATA     (sizeof (struct ufs_fsd))

 147 /*
 148  * Messy macros that would otherwise clutter up such glamorous code.
 149  */
 150 #define itob(i)         (((u_offset_t)itod(fs, (i)) << \
 151         (u_offset_t)FRGSHIFT) + (u_offset_t)itoo(fs, (i)) * (u_offset_t)INODE)
 152 #define min(x, y)       ((x) < (y) ? (x) : (y))
 153 #define STRINGSIZE(d)   ((long)d->d_reclen - \
 154                                 ((long)&d->d_name[0] - (long)&d->d_ino))
 155 #define letter(c)       ((((c) >= 'a')&&((c) <= 'z')) ||\
 156                                 (((c) >= 'A')&&((c) <= 'Z')))
 157 #define digit(c)        (((c) >= '0') && ((c) <= '9'))
 158 #define HEXLETTER(c)    (((c) >= 'A') && ((c) <= 'F'))
 159 #define hexletter(c)    (((c) >= 'a') && ((c) <= 'f'))
 160 #define octaldigit(c)   (((c) >= '0') && ((c) <= '7'))
 161 #define uppertolower(c) ((c) - 'A' + 'a')
 162 #define hextodigit(c)   ((c) - 'a' + 10)
 163 #define numtodigit(c)   ((c) - '0')

 165 #if !defined(loword)
 166 #define loword(X)       (((ushort_t *)&X)[1])
 167 #endif /* loword */

 169 #if !defined(lobyte)
 170 #define lobyte(X)       (((unsigned char *)&X)[1])
 171 #endif /* lobyte */

 173 /*
 174  * buffer cache structure.
 175  */
 176 static struct lbuf {
 177         struct  lbuf  *fwd;
 178         struct  lbuf  *back;
 179         char    *blkaddr;
 180         short   valid;
 181         u_offset_t      blkno;
 182 } lbuf[NBUF], bhdr;
_____unchanged_portion_omitted_

 369 /*
 370  * main - lines are read up to the unprotected ('\') newline and
 371  *      held in an input buffer.  Characters may be read from the
 372  *      input buffer using getachar() and unread using ungetachar().
 373  *      Reading the whole line ahead allows the use of debuggers
 374  *      which would otherwise be impossible since the debugger
 375  *      and fsdb could not share stdin.
 376  */
```



```
 378 int
 379 main(int argc, char *argv[])
 380 {

 382         char            c, *cptr;
 383         short           i;
 384         struct direct   *dirp;
 385         struct lbuf     *bp;
 386         char            *progname;
 387         volatile short  colon;
 388         short           mode;
 386         short           colon, mode;
 389         long            temp;

 391         /* Options/Suboptions processing */
 392         int     opt;
 393         char    *subopts;
 394         char    *optval;

 396         /*
 397          * The following are used to support the old fsdb functionality
 398          * of clearing an inode. It's better to use 'clri'.
 399          */
 400         int                     inum;   /* Inode number to clear */
 401         char                    *special;

 403         setbuf(stdin, NULL);
 404         progname = argv[0];
 405         prompt = &PROMPT[0];
 406         /*
 407          * Parse options.
 408          */
 409         while ((opt = getopt(argc, argv, FSDB_OPTIONS)) != EOF) {
 410                 switch (opt) {
 411 #if defined(OLD_FSDB_COMPATIBILITY)
 412                 case 'z':       /* Hack - Better to use clri */
 413                         (void) fprintf(stderr, "%s\n%s\n%s\n%s\n",
 414 "Warning: The '-z' option of 'fsdb_ufs' has been declared obsolete",
 415 "and may not be supported in a future version of Solaris.",
 416 "While this functionality is currently still supported, the",
 417 "recommended procedure to clear an inode is to use clri(1M).");
 418                         if (isnumber(optarg)) {
 419                                 inum = atoi(optarg);
 420                                 special = argv[optind];
 421                                 /* Doesn't return */
 422                                 old_fsdb(inum, special);
 423                         } else {
 424                                 usage(progname);
 425                                 exit(31+1);
 426                         }
 427                         /* Should exit() before here */
 428                         /*NOTREACHED*/
 429 #endif /* OLD_FSDB_COMPATIBILITY */
 430                 case 'o':
 431                         /* UFS Specific Options */
 432                         subopts = optarg;
 433                         while (*subopts != '\0') {
 434                                 switch (getsubopt(&subopts, subopt_v,
 435                                                                 &optval)) {
 436                                 case OVERRIDE:
 437                                         printf("error checking off\n");
 438                                         override = 1;
 439                                         break;

 441                                 /*
 442                                  * Change the "-o prompt=foo" option to
```

```
443                                 * "-o p=foo" to match documentation.
444                                 * ALT_PROMPT continues support for the
445                                 * undocumented "-o prompt=foo" option so
446                                 * that we don't break anyone.
447                                 */
448                                case NEW_PROMPT:
449                                case ALT_PROMPT:
450                                        if (optval == NULL) {
451                                                (void) fprintf(stderr,
452                                                        "No prompt string\n");
453                                                usage(progname);
454                                        }
455                                        (void) strncpy(PROMPT, optval,
456                                                                PROMPTSIZE);
457                                        break;

459                                case WRITE_ENABLED:
460                                        /* suitable for open */
461                                        wrtflag = O_RDWR;
462                                        break;

464                                default:
465                                        usage(progname);
466                                        /* Should exit here */
467                                }
468                        }
469                        break;

471                default:
472                        usage(progname);
473                }
474        }

476        if ((argc - optind) != 1) {     /* Should just have "special" left */
477                usage(progname);
478        }
479        special = argv[optind];

481        /*
482         * Unless it's already been set, the default prompt includes the
483         * name of the special device.
484         */
485        if (*prompt == NULL)
486                (void) sprintf(prompt, "%s > ", special);

488        /*
489         * Attempt to open the special file.
490         */
491        if ((fd = open(special, wrtflag)) < 0) {
492                perror(special);
493                exit(1);
494        }
495        /*
496         * Read in the super block and validate (not too picky).
497         */
498        if (llseek(fd, (offset_t)(SBLOCK * DEV_BSIZE), 0) == -1) {
499                perror(special);
500                exit(1);
501        }

503 #ifdef sun
504        if (read(fd, &filesystem, SBSIZE) != SBSIZE) {
505                printf("%s: cannot read superblock\n", special);
506                exit(1);
507        }
508 #else
```



```
509        if (read(fd, &filesystem, sizeof (filesystem)) != sizeof (filesystem)) {
510                printf("%s: cannot read superblock\n", special);
511                exit(1);
512        }
513 #endif /* sun */

515        fs = &filesystem;
516        if ((fs->fs_magic != FS_MAGIC) && (fs->fs_magic != MTB_UFS_MAGIC)) {
517                if (!override) {
518                        printf("%s: Bad magic number in file system\n",
519                                                                special);
520                        exit(1);
521                }

523                printf("WARNING: Bad magic number in file system. ");
524                printf("Continue? (y/n): ");
525                (void) fflush(stdout);
526                if (gets(input_buffer) == NULL) {
527                        exit(1);
528                }

530                if (*input_buffer != 'y' && *input_buffer != 'Y') {
531                        exit(1);
532                }
533        }

535        if ((fs->fs_magic == FS_MAGIC &&
536            (fs->fs_version != UFS_EFISTYLE4NONEFI_VERSION_2 &&
537            fs->fs_version != UFS_VERSION_MIN)) ||
538            (fs->fs_magic == MTB_UFS_MAGIC &&
539            (fs->fs_version > MTB_UFS_VERSION_1 ||
540            fs->fs_version < MTB_UFS_VERSION_MIN))) {
541                if (!override) {
542                        printf("%s: Unrecognized UFS version number: %d\n",
543                            special, fs->fs_version);
544                        exit(1);
545                }

547                printf("WARNING: Unrecognized UFS version number. ");
548                printf("Continue? (y/n): ");
549                (void) fflush(stdout);
550                if (gets(input_buffer) == NULL) {
551                        exit(1);
552                }

554                if (*input_buffer != 'y' && *input_buffer != 'Y') {
555                        exit(1);
556                }
557        }
558 #ifdef FS_42POSTBLFMT
559        if (fs->fs_postblformat == FS_42POSTBLFMT)
560                fs->fs_nrpos = 8;
561 #endif
562        printf("fsdb of %s %s -- last mounted on %s\n",
563                special,
564                (wrtflag == O_RDWR) ? "(Opened for write)" : "(Read only)",
565                &fs->fs_fsmnt[0]);
566 #ifdef sun
567        printf("fs_clean is currently set to ");
568        switch (fs->fs_clean) {

570        case FSACTIVE:
571                printf("FSACTIVE\n");
572                break;
573        case FSCLEAN:
574                printf("FSCLEAN\n");
```

```
575                 break;
576         case FSSTABLE:
577                 printf("FSSTABLE\n");
578                 break;
579         case FSBAD:
580                 printf("FSBAD\n");
581                 break;
582         case FSSUSPEND:
583                 printf("FSSUSPEND\n");
584                 break;
585         case FSLOG:
586                 printf("FSLOG\n");
587                 break;
588         case FSFIX:
589                 printf("FSFIX\n");
590                 if (!override) {
591                         printf("%s: fsck may be running on this file system\n",
592                                                                 special);
593                         exit(1);
594                 }
595
596                 printf("WARNING: fsck may be running on this file system. ");
597                 printf("Continue? (y/n): ");
598                 (void) fflush(stdout);
599                 if (gets(input_buffer) == NULL) {
600                         exit(1);
601                 }
602
603                 if (*input_buffer != 'y' && *input_buffer != 'Y') {
604                         exit(1);
605                 }
606                 break;
607         default:
608                 printf("an unknown value (0x%x)\n", fs->fs_clean);
609                 break;
610         }
611
612         if (fs->fs_state == (FSOKAY - fs->fs_time)) {
613                 printf("fs_state consistent (fs_clean CAN be trusted)\n");
614         } else {
615                 printf("fs_state inconsistent (fs_clean CAN'T trusted)\n");
616         }
617 #endif /* sun */
618         /*
619          * Malloc buffers and set up cache.
620          */
621         buffers = malloc(NBUF * BLKSIZE);
622         bhdr.fwd = bhdr.back = &bhdr;
623         for (i = 0; i < NBUF; i++) {
624                 bp = &lbuf[i];
625                 bp->blkaddr = buffers + (i * BLKSIZE);
626                 bp->valid = 0;
627                 insert(bp);
628         }
629         /*
630          * Malloc filenames structure.  The space for the actual filenames
631          * is allocated as it needs it. We estimate the size based on the
632          * number of inodes(objects) in the filesystem and the number of
633          * directories.  The number of directories are padded by 3 because
634          * each directory traversed during a "find" or "ls -R" needs 3
635          * entries.
636          */
637         maxfiles = (long)((((u_offset_t)fs->fs_ncg * (u_offset_t)fs->fs_ipg) -
638             (u_offset_t)fs->fs_cstotal.cs_nifree) +
639             ((u_offset_t)fs->fs_cstotal.cs_ndir * (u_offset_t)3));
```



```
641         filenames = (struct filenames *)calloc(maxfiles,
642             sizeof (struct filenames));
643         if (filenames == NULL) {
644                 /*
645                  * If we could not allocate memory for all of files
646                  * in the filesystem then, back off to the old fixed
647                  * value.
648                  */
649                 maxfiles = MAXFILES;
650                 filenames = (struct filenames *)calloc(maxfiles,
651                     sizeof (struct filenames));
652                 if (filenames == NULL) {
653                         printf("out of memory\n");
654                         exit(1);
655                 }
656         }
657
658         restore_inode(2);
659         /*
660          * Malloc a few filenames (needed by pwd for example).
661          */
662         for (i = 0; i < MAXPATHLEN; i++) {
663                 input_path[i] = calloc(1, MAXNAMLEN);
664                 stack_path[i] = calloc(1, MAXNAMLEN);
665                 current_path[i] = calloc(1, MAXNAMLEN);
666                 if (current_path[i] == NULL) {
667                         printf("out of memory\n");
668                         exit(1);
669                 }
670         }
671         current_pathp = -1;
672
673         (void) signal(2, err);
674         (void) setjmp(env);
675
676         getnextinput();
677         /*
678          * Main loop and case statement.  If an error condition occurs
679          * initialization and recovery is attempted.
680          */
681         for (;;) {
682                 if (error) {
683                         freemem(filenames, nfiles);
684                         nfiles = 0;
685                         c_count = 0;
686                         count = 1;
687                         star = 0;
688                         error = 0;
689                         paren = 0;
690                         acting_on_inode = 0;
691                         acting_on_directory = 0;
692                         should_print = 1;
693                         addr = erraddr;
694                         cur_ino = errino;
695                         cur_inum = errinum;
696                         cur_bytes = errcur_bytes;
697                         printf("?\n");
698                         getnextinput();
699                         if (error)
700                                 continue;
701                 }
702                 c_count++;
703
704                 switch (c = getachar()) {
705
706                 case '\n': /* command end */
```

```
707                     freemem(filenames, nfiles);
708                     nfiles = 0;
709                     if (should_print && laststyle == '=') {
710                             ungetachar(c);
711                             goto calc;
712                     }
713                     if (c_count == 1) {
714                             clear = 0;
715                             should_print = 1;
716                             erraddr = addr;
717                             errino = cur_ino;
718                             errinum = cur_inum;
719                             errcur_bytes = cur_bytes;
720                             switch (objsz) {
721                             case DIRECTORY:
722                                     if ((addr = getdirslot(
723                                                     (long)dirslot+1)) == 0)
724                                             should_print = 0;
725                                     if (error) {
726                                             ungetachar(c);
727                                             continue;
728                                     }
729                                     break;
730                             case INODE:
731                                     cur_inum++;
732                                     addr = itob(cur_inum);
733                                     if (!icheck(addr)) {
734                                             cur_inum--;
735                                             should_print = 0;
736                                     }
737                                     break;
738                             case CGRP:
739                             case SB:
740                                     cur_cgrp++;
741                                     addr = cgrp_check(cur_cgrp);
742                                     if (addr == 0) {
743                                             cur_cgrp--;
744                                             continue;
745                                     }
746                                     break;
747                             case SHADOW_DATA:
748                                     if ((addr = getshadowslot(
749                                         (long)cur_shad + 1)) == 0)
750                                             should_print = 0;
751                                     if (error) {
752                                             ungetachar(c);
753                                             continue;
754                                     }
755                                     break;
756                             default:
757                                     addr += objsz;
758                                     cur_bytes += objsz;
759                                     if (valid_addr() == 0)
760                                             continue;
761                             }
762                     }
763                     if (type == NUMB)
764                             trapped = 0;
765                     if (should_print)
766                             switch (objsz) {
767                             case DIRECTORY:
768                                     fprnt('?', 'd');
769                                     break;
770                             case INODE:
771                                     fprnt('?', 'i');
772                                     if (!error)
```



```
773                                             cur_ino = addr;
774                                     break;
775                             case CGRP:
776                                     fprnt('?', 'c');
777                                     break;
778                             case SB:
779                                     fprnt('?', 's');
780                                     break;
781                             case SHADOW_DATA:
782                                     fprnt('?', 'S');
783                                     break;
784                             case CHAR:
785                             case SHORT:
786                             case LONG:
787                                     fprnt(laststyle, lastpo);
788                             }
789                     if (error) {
790                             ungetachar(c);
791                             continue;
792                     }
793                     c_count = colon = acting_on_inode = 0;
794                     acting_on_directory = 0;
795                     should_print = 1;
796                     getnextinput();
797                     if (error)
798                             continue;
799                     erraddr = addr;
800                     errino = cur_ino;
801                     errinum = cur_inum;
802                     errcur_bytes = cur_bytes;
803                     continue;

805             case '(': /* numeric expression or unknown command */
806             default:
807                     colon = 0;
808                     if (digit(c) || c == '(') {
809                             ungetachar(c);
810                             addr = expr();
811                             type = NUMB;
812                             value = addr;
813                             continue;
814                     }
815                     printf("unknown command or bad syntax\n");
816                     error++;
817                     continue;

819             case '?': /* general print facilities */
820             case '/':
821                     fprnt(c, getachar());
822                     continue;

824             case ';': /* command separator and . */
825             case '\t':
826             case ' ':
827             case '.':
828                     continue;

830             case ':': /* command indicator */
831                     colon++;
832                     commands++;
833                     should_print = 0;
834                     stringsize = 0;
835                     trapped = 0;
836                     continue;

838             case ',': /* count indicator */
```

```
839                         colon = star = 0;
840                         if ((c = getachar()) == '*') {
841                                 star = 1;
842                                 count = BLKSIZE;
843                         } else {
844                                 ungetachar(c);
845                                 count = expr();
846                                 if (error)
847                                         continue;
848                                 if (!count)
849                                         count = 1;
850                         }
851                         clear = 0;
852                         continue;

854                 case '+': /* address addition */
855                         colon = 0;
856                         c = getachar();
857                         ungetachar(c);
858                         if (c == '\n')
859                                 temp = 1;
860                         else {
861                                 temp = expr();
862                                 if (error)
863                                         continue;
864                         }
865                         erraddr = addr;
866                         errcur_bytes = cur_bytes;
867                         switch (objsz) {
868                         case DIRECTORY:
869                                 addr = getdirslot((long)(dirslot + temp));
870                                 if (error)
871                                         continue;
872                                 break;
873                         case INODE:
874                                 cur_inum += temp;
875                                 addr = itob(cur_inum);
876                                 if (!icheck(addr)) {
877                                         cur_inum -= temp;
878                                         continue;
879                                 }
880                                 break;
881                         case CGRP:
882                         case SB:
883                                 cur_cgrp += temp;
884                                 if ((addr = cgrp_check(cur_cgrp)) == 0) {
885                                         cur_cgrp -= temp;
886                                         continue;
887                                 }
888                                 break;
889                         case SHADOW_DATA:
890                                 addr = getshadowslot((long)(cur_shad + temp));
891                                 if (error)
892                                     continue;
893                                 break;

895                         default:
896                                 laststyle = '/';
897                                 addr += temp * objsz;
898                                 cur_bytes += temp * objsz;
899                                 if (valid_addr() == 0)
900                                         continue;
901                         }
902                         value = get(objsz);
903                         continue;
```



```
905                 case '-': /* address subtraction */
906                         colon = 0;
907                         c = getachar();
908                         ungetachar(c);
909                         if (c == '\n')
910                                 temp = 1;
911                         else {
912                                 temp = expr();
913                                 if (error)
914                                         continue;
915                         }
916                         erraddr = addr;
917                         errcur_bytes = cur_bytes;
918                         switch (objsz) {
919                         case DIRECTORY:
920                                 addr = getdirslot((long)(dirslot - temp));
921                                 if (error)
922                                         continue;
923                                 break;
924                         case INODE:
925                                 cur_inum -= temp;
926                                 addr = itob(cur_inum);
927                                 if (!icheck(addr)) {
928                                         cur_inum += temp;
929                                         continue;
930                                 }
931                                 break;
932                         case CGRP:
933                         case SB:
934                                 cur_cgrp -= temp;
935                                 if ((addr = cgrp_check(cur_cgrp)) == 0) {
936                                         cur_cgrp += temp;
937                                         continue;
938                                 }
939                                 break;
940                         case SHADOW_DATA:
941                                 addr = getshadowslot((long)(cur_shad - temp));
942                                 if (error)
943                                         continue;
944                                 break;
945                         default:
946                                 laststyle = '/';
947                                 addr -= temp * objsz;
948                                 cur_bytes -= temp * objsz;
949                                 if (valid_addr() == 0)
950                                         continue;
951                         }
952                         value = get(objsz);
953                         continue;

955                 case '*': /* address multiplication */
956                         colon = 0;
957                         temp = expr();
958                         if (error)
959                                 continue;
960                         if (objsz != INODE && objsz != DIRECTORY)
961                                 laststyle = '/';
962                         addr *= temp;
963                         value = get(objsz);
964                         continue;

966                 case '%': /* address division */
967                         colon = 0;
968                         temp = expr();
969                         if (error)
970                                 continue;
```

```
                        if (!temp) {
                                printf("divide by zero\n");
                                error++;
                                continue;
                        }
                        if (objsz != INODE && objsz != DIRECTORY)
                                laststyle = '/';
                        addr /= temp;
                        value = get(objsz);
                        continue;

                case '=': { /* assignment operation */
                        short tbase;
calc:
                        tbase = base;

                        c = getachar();
                        if (c == '\n') {
                                ungetachar(c);
                                c = lastpo;
                                if (acting_on_inode == 1) {
                                        if (c != 'o' && c != 'd' && c != 'x' &&
                                            c != 'O' && c != 'D' && c != 'X') {
                                                switch (objsz) {
                                                case LONG:
                                                        c = lastpo = 'X';
                                                        break;
                                                case SHORT:
                                                        c = lastpo = 'x';
                                                        break;
                                                case CHAR:
                                                        c = lastpo = 'c';
                                                }
                                        }
                                } else {
                                        if (acting_on_inode == 2)
                                                c = lastpo = 't';
                                }
                        } else if (acting_on_inode)
                                lastpo = c;
                        should_print = star = 0;
                        count = 1;
                        erraddr = addr;
                        errcur_bytes = cur_bytes;
                        switch (c) {
                        case '"': /* character string */
                                if (type == NUMB) {
                                        blocksize = BLKSIZE;
                                        filesize = BLKSIZE * 2;
                                        cur_bytes = blkoff(fs, addr);
                                        if (objsz == DIRECTORY ||
                                                                objsz == INODE)
                                                lastpo = 'X';
                                }
                                puta();
                                continue;
                        case '+': /* =+ operator */
                                temp = expr();
                                value = get(objsz);
                                if (!error)
                                        put(value+temp, objsz);
                                continue;
                        case '-': /* =- operator */
                                temp = expr();
                                value = get(objsz);
                                if (!error)
```



```
                                        put(value-temp, objsz);
                                continue;
                        case 'b':
                        case 'c':
                                if (objsz == CGRP)
                                        fprnt('?', c);
                                else
                                        fprnt('/', c);
                                continue;
                        case 'i':
                                addr = cur_ino;
                                fprnt('?', 'i');
                                continue;
                        case 's':
                                fprnt('?', 's');
                                continue;
                        case 't':
                        case 'T':
                                laststyle = '=';
                                printf("\t\t");
                                {
                                        /*
                                         * Truncation is intentional so
                                         * ctime is happy.
                                         */
                                        time_t tvalue = (time_t)value;
                                        printf("%s", ctime(&tvalue));
                                }
                                continue;
                        case 'o':
                                base = OCTAL;
                                goto otx;
                        case 'd':
                                if (objsz == DIRECTORY) {
                                        addr = cur_dir;
                                        fprnt('?', 'd');
                                        continue;
                                }
                                base = DECIMAL;
                                goto otx;
                        case 'x':
                                base = HEX;
otx:
                                laststyle = '=';
                                printf("\t\t");
                                if (acting_on_inode)
                                        print(value & 0177777L, 12, -8, 0);
                                else
                                        print(addr & 0177777L, 12, -8, 0);
                                printf("\n");
                                base = tbase;
                                continue;
                        case 'O':
                                base = OCTAL;
                                goto OTX;
                        case 'D':
                                base = DECIMAL;
                                goto OTX;
                        case 'X':
                                base = HEX;
OTX:
                                laststyle = '=';
                                printf("\t\t");
                                if (acting_on_inode)
                                        print(value, 12, -8, 0);
                                else
```

```
1103                                    print(addr, 12, -8, 0);
1104                            printf("\n");
1105                            base = tbase;
1106                            continue;
1107                    default: /* regular assignment */
1108                            ungetachar(c);
1109                            value = expr();
1110                            if (error)
1111                                    printf("syntax error\n");
1112                            else
1113                                    put(value, objsz);
1114                            continue;
1115                    }
1116            }

1117            case '>': /* save current address */
1118                    colon = 0;
1119                    should_print = 0;
1120                    c = getachar();
1121                    if (!letter(c) && !digit(c)) {
1122                            printf("invalid register specification, ");
1123                            printf("must be letter or digit\n");
1124                            error++;
1125                            continue;
1126                    }
1127                    if (letter(c)) {
1128                            if (c < 'a')
1129                                    c = uppertolower(c);
1130                            c = hextodigit(c);
1131                    } else
1132                            c = numtodigit(c);
1133                    regs[c].sv_addr = addr;
1134                    regs[c].sv_value = value;
1135                    regs[c].sv_objsz = objsz;
1136                    continue;

1137            case '<': /* restore saved address */
1138                    colon = 0;
1139                    should_print = 0;
1140                    c = getachar();
1141                    if (!letter(c) && !digit(c)) {
1142                            printf("invalid register specification, ");
1143                            printf("must be letter or digit\n");
1144                            error++;
1145                            continue;
1146                    }
1147                    if (letter(c)) {
1148                            if (c < 'a')
1149                                    c = uppertolower(c);
1150                            c = hextodigit(c);
1151                    } else
1152                            c = numtodigit(c);
1153                    addr = regs[c].sv_addr;
1154                    value = regs[c].sv_value;
1155                    objsz = regs[c].sv_objsz;
1156                    continue;

1157            case 'a':
1158                    if (colon)
1159                            colon = 0;
1160                    else
1161                            goto no_colon;
1162                    if (match("at", 2)) {          /* access time */
1163                            acting_on_inode = 2;
1164                            should_print = 1;
1165                            addr = (long)&((struct dinode *)
1166                                    (uintptr_t)cur_ino)->di_atime;
1167                            value = get(LONG);
1168                            type = NULL;
1169                            continue;
1170                    }
1171                    goto bad_syntax;

1172            case 'b':
1173                    if (colon)
1174                            colon = 0;
1175                    else
1176                            goto no_colon;
1177                    if (match("block", 2)) {        /* block conversion */
1178                            if (type == NUMB) {
1179                                    value = addr;
1180                                    cur_bytes = 0;
1181                                    blocksize = BLKSIZE;
1182                                    filesize = BLKSIZE * 2;
1183                            }
1184                            addr = value << FRGSHIFT;
1185                            bod_addr = addr;
1186                            value = get(LONG);
1187                            type = BLOCK;
1188                            dirslot = 0;
1189                            trapped++;
1190                            continue;
1191                    }
1192                    if (match("bs", 2)) {           /* block size */
1193                            acting_on_inode = 1;
1194                            should_print = 1;
1195                            if (icheck(cur_ino) == 0)
1196                                    continue;
1197                            addr = (long)&((struct dinode *)
1198                                    (uintptr_t)cur_ino)->di_blocks;
1199                            value = get(LONG);
1200                            type = NULL;
1201                            continue;
1202                    }
1203                    if (match("base", 2)) {         /* change/show base */
1204            showbase:
1205                            if ((c = getachar()) == '\n') {
1206                                    ungetachar(c);
1207                                    printf("base =\t\t");
1208                                    switch (base) {
1209                                    case OCTAL:
1210                                            printf("OCTAL\n");
1211                                            continue;
1212                                    case DECIMAL:
1213                                            printf("DECIMAL\n");
1214                                            continue;
1215                                    case HEX:
1216                                            printf("HEX\n");
1217                                            continue;
1218                                    }
1219                            }
1220                            if (c != '=') {
1221                                    printf("missing '='\n");
1222                                    error++;
1223                                    continue;
1224                            }
1225                            value = expr();
1226                            switch (value) {
1227                            default:
1228                                    printf("invalid base\n");
1229                                    error++;
1230                                    break;
1231
1232
1233
1234
```

```
1235                                    case OCTAL:
1236                                    case DECIMAL:
1237                                    case HEX:
1238                                            base = (short)value;
1239                                    }
1240                                    goto showbase;
1241                            }
1242                            goto bad_syntax;

1244                    case 'c':
1245                            if (colon)
1246                                    colon = 0;
1247                            else
1248                                    goto no_colon;
1249                            if (match("cd", 2)) {           /* change directory */
1250                                    top = filenames - 1;
1251                                    eat_spaces();
1252                                    if ((c = getachar()) == '\n') {
1253                                            ungetachar(c);
1254                                            current_pathp = -1;
1255                                            restore_inode(2);
1256                                            continue;
1257                                    }
1258                                    ungetachar(c);
1259                                    temp = cur_inum;
1260                                    doing_cd = 1;
1261                                    parse();
1262                                    doing_cd = 0;
1263                                    if (nfiles != 1) {
1264                                            restore_inode((ino_t)temp);
1265                                            if (!error) {
1266                                                    print_path(input_path,
1267                                                            (int)input_pathp);
1268                                                    if (nfiles == 0)
1269                                                            printf(" not found\n");
1270                                                    else
1271                                                            printf(" ambiguous\n");
1272                                                    error++;
1273                                            }
1274                                            continue;
1275                                    }
1276                                    restore_inode(filenames->ino);
1277                                    if ((mode = icheck(addr)) == 0)
1278                                            continue;
1279                                    if ((mode & IFMT) != IFDIR) {
1280                                            restore_inode((ino_t)temp);
1281                                            print_path(input_path,
1282                                                    (int)input_pathp);
1283                                            printf(" not a directory\n");
1284                                            error++;
1285                                            continue;
1286                                    }
1287                                    for (i = 0; i <= top->len; i++)
1288                                            (void) strcpy(current_path[i],
1289                                                    top->fname[i]);
1290                                    current_pathp = top->len;
1291                                    continue;
1292                            }
1293                            if (match("cg", 2)) {           /* cylinder group */
1294                                    if (type == NUMB)
1295                                            value = addr;
1296                                    if (value > fs->fs_ncg - 1) {
1297                                            printf("maximum cylinder group is ");
1298                                            print(fs->fs_ncg - 1, 8, -8, 0);
1299                                            printf("\n");
1300                                            error++;
```



```
1301                                            continue;
1302                                    }
1303                                    type = objsz = CGRP;
1304                                    cur_cgrp = (long)value;
1305                                    addr = cgtod(fs, cur_cgrp) << FRGSHIFT;
1306                                    continue;
1307                            }
1308                            if (match("ct", 2)) {           /* creation time */
1309                                    acting_on_inode = 2;
1310                                    should_print = 1;
1311                                    addr = (long)&((struct dinode *)
1312                                                    (uintptr_t)cur_ino)->di_ctime;
1313                                    value = get(LONG);
1314                                    type = NULL;
1315                                    continue;
1316                            }
1317                            goto bad_syntax;

1319                    case 'd':
1320                            if (colon)
1321                                    colon = 0;
1322                            else
1323                                    goto no_colon;
1324                            if (match("directory", 2)) {    /* directory offsets */
1325                                    if (type == NUMB)
1326                                            value = addr;
1327                                    objsz = DIRECTORY;
1328                                    type = DIRECTORY;
1329                                    addr = (u_offset_t)getdirslot((long)value);
1330                                    continue;
1331                            }
1332                            if (match("db", 2)) {           /* direct block */
1333                                    acting_on_inode = 1;
1334                                    should_print = 1;
1335                                    if (type == NUMB)
1336                                            value = addr;
1337                                    if (value >= NDADDR) {
1338                                            printf("direct blocks are 0 to ");
1339                                            print(NDADDR - 1, 0, 0, 0);
1340                                            printf("\n");
1341                                            error++;
1342                                            continue;
1343                                    }
1344                                    addr = cur_ino;
1345                                    if (!icheck(addr))
1346                                            continue;
1347                                    addr = (long)
1348                                            &((struct dinode *)(uintptr_t)cur_ino)->
1349                                                            di_db[value];
1350                                    bod_addr = addr;
1351                                    cur_bytes = (value) * BLKSIZE;
1352                                    cur_block = (long)value;
1353                                    type = BLOCK;
1354                                    dirslot = 0;
1355                                    value = get(LONG);
1356                                    if (!value && !override) {
1357                                            printf("non existent block\n");
1358                                            error++;
1359                                    }
1360                                    continue;
1361                            }
1362                            goto bad_syntax;

1364                    case 'f':
1365                            if (colon)
1366                                    colon = 0;
```

```
1367                        else
1368                                goto no_colon;
1369                        if (match("find", 3)) {         /* find command */
1370                                find();
1371                                continue;
1372                        }
1373                        if (match("fragment", 2)) {     /* fragment conv. */
1374                                if (type == NUMB) {
1375                                        value = addr;
1376                                        cur_bytes = 0;
1377                                        blocksize = FRGSIZE;
1378                                        filesize = FRGSIZE * 2;
1379                                }
1380                                if (min(blocksize, filesize) - cur_bytes >
1381                                                        FRGSIZE) {
1382                                        blocksize = cur_bytes + FRGSIZE;
1383                                        filesize = blocksize * 2;
1384                                }
1385                                addr = value << FRGSHIFT;
1386                                bod_addr = addr;
1387                                value = get(LONG);
1388                                type = FRAGMENT;
1389                                dirslot = 0;
1390                                trapped++;
1391                                continue;
1392                        }
1393                        if (match("file", 4)) {         /* access as file */
1394                                acting_on_inode = 1;
1395                                should_print = 1;
1396                                if (type == NUMB)
1397                                        value = addr;
1398                                addr = cur_ino;
1399                                if ((mode = icheck(addr)) == 0)
1400                                        continue;
1401                                if (!override) {
1402                                        switch (mode & IFMT) {
1403                                        case IFCHR:
1404                                        case IFBLK:
1405                                            printf("special device\n");
1406                                            error++;
1407                                            continue;
1408                                        }
1409                                }
1410                                if ((addr = (u_offset_t)
1411                                    (bmap((long)value) << FRGSHIFT)) == 0)
1412                                        continue;
1413                                cur_block = (long)value;
1414                                bod_addr = addr;
1415                                type = BLOCK;
1416                                dirslot = 0;
1417                                continue;
1418                        }
1419                        if (match("fill", 4)) {         /* fill */
1420                                if (getachar() != '=') {
1421                                        printf("missing '=\n");
1422                                        error++;
1423                                        continue;
1424                                }
1425                                if (objsz == INODE || objsz == DIRECTORY ||
1426                                    objsz == SHADOW_DATA) {
1427                                        printf(
1428                                            "can't fill inode or directory\n");
1429                                        error++;
1430                                        continue;
1431                                }
1432                                fill();
```



```
1433                                        continue;
1434                                }
1435                                goto bad_syntax;

1437                        case 'g':
1438                                if (colon)
1439                                        colon = 0;
1440                                else
1441                                        goto no_colon;
1442                                if (match("gid", 1)) {          /* group id */
1443                                        acting_on_inode = 1;
1444                                        should_print = 1;
1445                                        addr = (long)&((struct dinode *)
1446                                                        (uintptr_t)cur_ino)->di_gid;
1447                                        value = get(SHORT);
1448                                        type = NULL;
1449                                        continue;
1450                                }
1451                                goto bad_syntax;

1453                        case 'i':
1454                                if (colon)
1455                                        colon = 0;
1456                                else
1457                                        goto no_colon;
1458                                if (match("inode", 2)) { /* i# to inode conversion */
1459                                        if (c_count == 2) {
1460                                                addr = cur_ino;
1461                                                value = get(INODE);
1462                                                type = NULL;
1463                                                laststyle = '=';
1464                                                lastpo = 'i';
1465                                                should_print = 1;
1466                                                continue;
1467                                        }
1468                                        if (type == NUMB)
1469                                                value = addr;
1470                                        addr = itob(value);
1471                                        if (!icheck(addr))
1472                                                continue;
1473                                        cur_ino = addr;
1474                                        cur_inum = (long)value;
1475                                        value = get(INODE);
1476                                        type = NULL;
1477                                        continue;
1478                                }
1479                                if (match("ib", 2)) {   /* indirect block */
1480                                        acting_on_inode = 1;
1481                                        should_print = 1;
1482                                        if (type == NUMB)
1483                                                value = addr;
1484                                        if (value >= NIADDR) {
1485                                                printf("indirect blocks are 0 to ");
1486                                                print(NIADDR - 1, 0, 0, 0);
1487                                                printf("\n");
1488                                                error++;
1489                                                continue;
1490                                        }
1491                                        addr = (long)&((struct dinode *)(uintptr_t)
1492                                                        cur_ino)->di_ib[value];
1493                                        cur_bytes = (NDADDR - 1) * BLKSIZE;
1494                                        temp = 1;
1495                                        for (i = 0; i < value; i++) {
1496                                                temp *= NINDIR(fs) * BLKSIZE;
1497                                                cur_bytes += temp;
1498                                        }
```

```
1499                                    type = BLOCK;
1500                                    dirslot = 0;
1501                                    value = get(LONG);
1502                                    if (!value && !override) {
1503                                            printf("non existent block\n");
1504                                            error++;
1505                                    }
1506                                    continue;
1507                            }
1508                            goto bad_syntax;

1510                    case 'l':
1511                            if (colon)
1512                                    colon = 0;
1513                            else
1514                                    goto no_colon;
1515                            if (match("log_head", 8)) {
1516                                    log_display_header();
1517                                    should_print = 0;
1518                                    continue;
1519                            }
1520                            if (match("log_delta", 9)) {
1521                                    log_show(LOG_NDELTAS);
1522                                    should_print = 0;
1523                                    continue;
1524                            }
1525                            if (match("log_show", 8)) {
1526                                    log_show(LOG_ALLDELTAS);
1527                                    should_print = 0;
1528                                    continue;
1529                            }
1530                            if (match("log_chk", 7)) {
1531                                    log_show(LOG_CHECKSCAN);
1532                                    should_print = 0;
1533                                    continue;
1534                            }
1535                            if (match("log_otodb", 9)) {
1536                                    if (log_lodb((u_offset_t)addr, &temp)) {
1537                                            addr = temp;
1538                                            should_print = 1;
1539                                            laststyle = '=';
1540                                    } else
1541                                            error++;
1542                                    continue;
1543                            }
1544                            if (match("ls", 2)) {           /* ls command */
1545                                    temp = cur_inum;
1546                                    recursive = long_list = 0;
1547                                    top = filenames - 1;
1548                                    for (;;) {
1549                                            eat_spaces();
1550                                            if ((c = getachar()) == '-') {
1551                                                    if ((c = getachar()) == 'R') {
1552                                                            recursive = 1;
1553                                                            continue;
1554                                                    } else if (c == 'l') {
1555                                                            long_list = 1;
1556                                                    } else {
1557                                                            printf(
1558                                                                "unknown option ");
1559                                                            printf("'%c'\n", c);
1560                                                            error++;
1561                                                            break;
1562                                                    }
1563                                            } else
1564                                                    ungetachar(c);
```



```
1565                                            if ((c = getachar()) == '\n') {
1566                                                    if (c_count != 2) {
1567                                                            ungetachar(c);
1568                                                            break;
1569                                                    }
1570                                            }
1571                                            c_count++;
1572                                            ungetachar(c);
1573                                            parse();
1574                                            restore_inode((ino_t)temp);
1575                                            if (error)
1576                                                    break;
1577                                    }
1578                                    recursive = 0;
1579                                    if (error || nfiles == 0) {
1580                                            if (!error) {
1581                                                    print_path(input_path,
1582                                                            (int)input_pathp);
1583                                                    printf(" not found\n");
1584                                            }
1585                                            continue;
1586                                    }
1587                                    if (nfiles) {
1588                                        cmp_level = 0;
1589                                        qsort((char *)filenames, nfiles,
1590                                            sizeof (struct filenames), ffcmp);
1591                                        ls(filenames, filenames + (nfiles - 1), 0);
1592                                    } else {
1593                                        printf("no match\n");
1594                                        error++;
1595                                    }
1596                                    restore_inode((ino_t)temp);
1597                                    continue;
1598                            }
1599                            if (match("ln", 2)) {           /* link count */
1600                                    acting_on_inode = 1;
1601                                    should_print = 1;
1602                                    addr = (long)&((struct dinode *)
1603                                            (uintptr_t)cur_ino)->di_nlink;
1604                                    value = get(SHORT);
1605                                    type = NULL;
1606                                    continue;
1607                            }
1608                            goto bad_syntax;

1610                    case 'm':
1611                            if (colon)
1612                                    colon = 0;
1613                            else
1614                                    goto no_colon;
1615                            addr = cur_ino;
1616                            if ((mode = icheck(addr)) == 0)
1617                                    continue;
1618                            if (match("mt", 2)) {           /* modification time */
1619                                    acting_on_inode = 2;
1620                                    should_print = 1;
1621                                    addr = (long)&((struct dinode *)
1622                                            (uintptr_t)cur_ino)->di_mtime;
1623                                    value = get(LONG);
1624                                    type = NULL;
1625                                    continue;
1626                            }
1627                            if (match("md", 2)) {           /* mode */
1628                                    acting_on_inode = 1;
1629                                    should_print = 1;
1630                                    addr = (long)&((struct dinode *)
```

```
1631                                            (uintptr_t)cur_ino)->di_mode;
1632                            value = get(SHORT);
1633                            type = NULL;
1634                            continue;
1635                    }
1636                    if (match("maj", 2)) {  /* major device number */
1637                            acting_on_inode = 1;
1638                            should_print = 1;
1639                            if (devcheck(mode))
1640                                    continue;
1641                            addr = (uintptr_t)&((struct dinode *)(uintptr_t)
1642                                            cur_ino)->di_ordev;
1643                            {
1644                                    long    dvalue;
1645                                    dvalue = get(LONG);
1646                                    value = major(dvalue);
1647                            }
1648                            type = NULL;
1649                            continue;
1650                    }
1651                    if (match("min", 2)) {  /* minor device number */
1652                            acting_on_inode = 1;
1653                            should_print = 1;
1654                            if (devcheck(mode))
1655                                    continue;
1656                            addr = (uintptr_t)&((struct dinode *)(uintptr_t)
1657                                            cur_ino)->di_ordev;
1658                            {
1659                                    long    dvalue;
1660                                    dvalue = (long)get(LONG);
1661                                    value = minor(dvalue);
1662                            }
1663                            type = NULL;
1664                            continue;
1665                    }
1666                    goto bad_syntax;

1668            case 'n':
1669                    if (colon)
1670                            colon = 0;
1671                    else
1672                            goto no_colon;
1673                    if (match("nm", 1)) {           /* directory name */
1674                            objsz = DIRECTORY;
1675                            acting_on_directory = 1;
1676                            cur_dir = addr;
1677                            if ((cptr = getblk(addr)) == 0)
1678                                    continue;
1679                            /*LINTED*/
1680                            dirp = (struct direct *)(cptr+blkoff(fs, addr));
1681                            stringsize = (long)dirp->d_reclen -
1682                                    ((long)&dirp->d_name[0] -
1683                                            (long)&dirp->d_ino);
1684                            addr = (long)&((struct direct *)
1685                                    (uintptr_t)addr)->d_name[0];
1686                            type = NULL;
1687                            continue;
1688                    }
1689                    goto bad_syntax;

1691            case 'o':
1692                    if (colon)
1693                            colon = 0;
1694                    else
1695                            goto no_colon;
1696                    if (match("override", 1)) {     /* override flip flop */
```



```
1697                            override = !override;
1698                            if (override)
1699                                    printf("error checking off\n");
1700                            else
1701                                    printf("error checking on\n");
1702                            continue;
1703                    }
1704                    goto bad_syntax;

1706            case 'p':
1707                    if (colon)
1708                            colon = 0;
1709                    else
1710                            goto no_colon;
1711                    if (match("pwd", 2)) {          /* print working dir */
1712                            print_path(current_path, (int)current_pathp);
1713                            printf("\n");
1714                            continue;
1715                    }
1716                    if (match("prompt", 2)) {       /* change prompt */
1717                            if ((c = getachar()) != '=') {
1718                                    printf("missing '='\n");
1719                                    error++;
1720                                    continue;
1721                            }
1722                            if ((c = getachar()) != '"') {
1723                                    printf("missing '\"'\n");
1724                                    error++;
1725                                    continue;
1726                            }
1727                            i = 0;
1728                            prompt = &prompt[0];
1729                            while ((c = getachar()) != '"' && c != '\n') {
1730                                    prompt[i++] = c;
1731                                    if (i >= PROMPTSIZE) {
1732                                            printf("string too long\n");
1733                                            error++;
1734                                            break;
1735                                    }
1736                            }
1737                            prompt[i] = '\0';
1738                            continue;
1739                    }
1740                    goto bad_syntax;

1742            case 'q':
1743                    if (!colon)
1744                            goto no_colon;
1745                    if (match("quit", 1)) {         /* quit */
1746                            if ((c = getachar()) != '\n') {
1747                                    error++;
1748                                    continue;
1749                            }
1750                            exit(0);
1751                    }
1752                    goto bad_syntax;

1754            case 's':
1755                    if (colon)
1756                            colon = 0;
1757                    else
1758                            goto no_colon;
1759                    if (match("sb", 2)) {           /* super block */
1760                            if (c_count == 2) {
1761                                    cur_cgrp = -1;
1762                                    type = objsz = SB;
```

```
1763                                    laststyle = '=';
1764                                    lastpo = 's';
1765                                    should_print = 1;
1766                                    continue;
1767                            }
1768                            if (type == NUMB)
1769                                    value = addr;
1770                            if (value > fs->fs_ncg - 1) {
1771                                    printf("maximum super block is ");
1772                                    print(fs->fs_ncg - 1, 8, -8, 0);
1773                                    printf("\n");
1774                                    error++;
1775                                    continue;
1776                            }
1777                            type = objsz = SB;
1778                            cur_cgrp = (long)value;
1779                            addr = cgsblock(fs, cur_cgrp) << FRGSHIFT;
1780                            continue;
1781                    }
1782                    if (match("shadow", 2)) {       /* shadow inode data */
1783                            if (type == NUMB)
1784                                    value = addr;
1785                            objsz = SHADOW_DATA;
1786                            type = SHADOW_DATA;
1787                            addr = getshadowslot(value);
1788                            continue;
1789                    }
1790                    if (match("si", 2)) {   /* shadow inode field */
1791                            acting_on_inode = 1;
1792                            should_print = 1;
1793                            addr = (long)&((struct dinode *)
1794                                    (uintptr_t)cur_ino)->di_shadow;
1795                            value = get(LONG);
1796                            type = NULL;
1797                            continue;
1798                    }

1800                    if (match("sz", 2)) {           /* file size */
1801                            acting_on_inode = 1;
1802                            should_print = 1;
1803                            addr = (long)&((struct dinode *)
1804                                    (uintptr_t)cur_ino)->di_size;
1805                            value = get(U_OFFSET_T);
1806                            type = NULL;
1807                            objsz = U_OFFSET_T;
1808                            laststyle = '=';
1809                            lastpo = 'X';
1810                            continue;
1811                    }
1812                    goto bad_syntax;

1814            case 'u':
1815                    if (colon)
1816                            colon = 0;
1817                    else
1818                            goto no_colon;
1819                    if (match("uid", 1)) {          /* user id */
1820                            acting_on_inode = 1;
1821                            should_print = 1;
1822                            addr = (long)&((struct dinode *)
1823                                    (uintptr_t)cur_ino)->di_uid;
1824                            value = get(SHORT);
1825                            type = NULL;
1826                            continue;
1827                    }
1828                    goto bad_syntax;
```



```
1830                    case 'F': /* buffer status (internal use only) */
1831                            if (colon)
1832                                    colon = 0;
1833                            else
1834                                    goto no_colon;
1835                            for (bp = bhdr.fwd; bp != &bhdr; bp = bp->fwd)
1836                                    printf("%8" PRIx64 " %d\n",
1837                                        bp->blkno, bp->valid);
1838                            printf("\n");
1839                            printf("# commands\t\t%ld\n", commands);
1840                            printf("# read requests\t\t%ld\n", read_requests);
1841                            printf("# actual disk reads\t%ld\n", actual_disk_reads);
1842                            continue;
1843 no_colon:
1844                    printf("a colon should precede a command\n");
1845                    error++;
1846                    continue;
1847 bad_syntax:
1848                    printf("more letters needed to distinguish command\n");
1849                    error++;
1850                    continue;
1851                    }
1852            }
1853 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   15824 Tue May 26 15:56:27 2015
new/usr/src/lib/libcurses/screen/tparm.c
5396 gcc 4.8.2 longjmp errors for cscope-fast
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (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 2015 Gary Mills
 * Copyright (c) 1996-1997 by Sun Microsystems, Inc.
 * All rights reserved.
 */

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


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

/* Copyright (c) 1979 Regents of the University of California */

/*LINTLIBRARY*/

#include        "curses_inc.h"
#include        "curshdr.h"
#include        "term.h"
#include        <string.h>
#include        <setjmp.h>
#include        <stdlib.h>
#include        <stdio.h>

#ifndef _CHCTRL
#define _CHCTRL(c)      ((c) & 037)
#endif  /* _CHCTRL */

char    *_branchto(char *, char);

/*
 * Routine to perform parameter substitution.
 * instring is a string containing printf type escapes.
 * The whole thing uses a stack, much like an HP 35.
 * The following escapes are defined for substituting row/column:
 *
 *      %[:[-+ #0]][0-9][.][0-9][dsoxX]
 *              print pop() as in printf(3), as defined in the local
 *              sprintf(3), except that a leading + or - must be preceded
 *              with a colon (:) to distinguish from the plus/minus operators.
```



```
 *
 *      %c      print pop() like %c in printf(3)
 *      %l      pop() a string address and push its length.
 *      %P[a-z] set dynamic variable a-z
 *      %g[a-z] get dynamic variable a-z
 *      %P[A-Z] set static variable A-Z
 *      %g[A-Z] get static variable A-Z
 *
 *      %p[1-0] push ith parm
 *      %'c'    char constant c
 *      %{nn}   integer constant nn
 *
 *      %+ %- %* %/ %m          arithmetic (%m is mod): push(pop() op pop())
 *      %& %| %^                bit operations:         push(pop() op pop())
 *      %= %> %<                logical operations:     push(pop() op pop())
 *      %A %O                   logical AND, OR         push(pop() op pop())
 *      %! %~                   unary operations        push(op pop())
 *      %%                      output %
 *      %? expr %t thenpart %e elsepart %;
 *                              if-then-else, %e elsepart is optional.
 *                              else-if's are possible ala Algol 68:
 *                              %? c1 %t %e c2 %t %e c3 %t %e c4 %t %e %;
 *      % followed by anything else
 *                              is not defined, it may output the character,
 *                              and it may not. This is done so that further
 *                              enhancements to the format capabilities may
 *                              be made without worrying about being upwardly
 *                              compatible from buggy code.
 *
 * all other characters are ''self-inserting''.  %% gets % output.
 *
 * The stack structure used here is based on an idea by Joseph Yao.
 */

#define MAX             10
#define MEM_ALLOC_FAIL  1
#define STACK_UNDERFLOW 2

typedef struct {
        long    top;
        int     stacksize;
        long    *stack;

}STACK;
______unchanged_portion_omitted_

/* VARARGS */
char    *
tparm(char *instring, long fp1, long fp2, long p3, long p4,
        long p5, long p6, long p7, long p8, long p9)
{
        static  char    result[512];
        static  char    added[100];
        long            vars[26];
        STACK           stk;
        char            *cp = instring;
        char            *outp = result;
        char            c;
        long            op;
        long            op2;
        int             sign;
        int             onrow = 0;
        volatile long   p1 = fp1, p2 = fp2; /* copy in case < 2 actual parms */
        long            p1 = fp1, p2 = fp2;  /* copy in case < 2 actual parms */
        char            *xp;
        char            formatbuffer[100];
```

```
264         char            *format;
265         int             looping;
266         short           *regs = cur_term->_regs;
267         int             val;


270         if ((val = setjmp(env)) != 0) {
271 #ifdef DEBUG
272                 switch (val) {
273                         case MEM_ALLOC_FAIL:
274                                 fprintf(outf, "TPARM: Memory allocation"
275                                     " failure.");
276                                 break;
277                         case STACK_UNDERFLOW:
278                                 fprintf(outf, "TPARM: Stack underflow.");
279                                 break;
280                 }
281 #endif  /* DEBUG */

283                 if (val == STACK_UNDERFLOW)
284                         free_stack(&stk);
285                 return (NULL);
286         }

288         init_stack(&stk);
289         push(&stk, 0);

291         if (instring == 0) {
292 #ifdef  DEBUG
293                 if (outf)
294                         fprintf(outf, "TPARM: null arg\n");
295 #endif  /* DEBUG */
296                 free_stack(&stk);
297                 return (NULL);
298         }

300         added[0] = 0;

302         while ((c = *cp++) != 0) {
303                 if (c != '%') {
304                         *outp++ = c;
305                         continue;
306                 }
307                 op = tops(&stk);
308                 switch (c = *cp++) {
309                         /* PRINTING CASES */
310                         case ':':
311                         case ' ':
312                         case '#':
313                         case '0':
314                         case '1':
315                         case '2':
316                         case '3':
317                         case '4':
318                         case '5':
319                         case '6':
320                         case '7':
321                         case '8':
322                         case '9':
323                         case '.':
324                         case 'd':
325                         case 's':
326                         case 'o':
327                         case 'x':
328                         case 'X':
329                                 format = formatbuffer;
```



```
330                                 *format++ = '%';

332                         /* leading ':' to allow +/- in format */
333                         if (c == ':')
334                                 c = *cp++;

336                         /* take care of flags, width and precision */
337                         looping = 1;
338                         while (c && looping)
339                                 switch (c) {
340                                         case '-':
341                                         case '+':
342                                         case ' ':
343                                         case '#':
344                                         case '0':
345                                         case '1':
346                                         case '2':
347                                         case '3':
348                                         case '4':
349                                         case '5':
350                                         case '6':
351                                         case '7':
352                                         case '8':
353                                         case '9':
354                                         case '.':
355                                                 *format++ = c;
356                                                 c = *cp++;
357                                                 break;
358                                         default:
359                                                 looping = 0;
360                                 }

362                         /* add in the conversion type */
363                         switch (c) {
364                                 case 'd':
365                                 case 's':
366                                 case 'o':
367                                 case 'x':
368                                 case 'X':
369                                         *format++ = c;
370                                         break;
371                                 default:
372 #ifdef  DEBUG
373                                         if (outf)
374                                                 fprintf(outf, "TPARM: invalid "
375                                                     "conversion type\n");
376 #endif  /* DEBUG */
377                                         free_stack(&stk);
378                                         return (NULL);
379                         }
380                         *format = '\0';

382                         /*
383                          * Pass off the dirty work to sprintf.
384                          * It's debatable whether we should just pull in
385                          * the appropriate code here. I decided not to for
386                          * now.
387                          */
388                         if (c == 's')
389                                 (void) sprintf(outp, formatbuffer,
390                                     (char *) op);
391                         else
392                                 (void) sprintf(outp, formatbuffer, op);
393                         /*
394                          * Advance outp past what sprintf just did.
395                          * sprintf returns an indication of its length on some
```

```
396                          * systems, others the first char, and there's
397                          * no easy way to tell which. The Sys V on
398                          * BSD emulations are particularly confusing.
399                          */
400                                 while (*outp)
401                                     outp++;
402                                 (void) pop(&stk);

404                                 continue;

406                         case 'c':
407                         /*
408                          * This code is worth scratching your head at for a
409                          * while.  The idea is that various weird things can
410                          * happen to nulls, EOT's, tabs, and newlines by the
411                          * tty driver, arpanet, and so on, so we don't send
412                          * them if we can help it.  So we instead alter the
413                          * place being addessed and then move the cursor
414                          * locally using UP or RIGHT.
415                          *
416                          * This is a kludge, clearly.  It loses if the
417                          * parameterized string isn't addressing the cursor
418                          * (but hopefully that is all that %c terminals do
419                          * with parms).  Also, since tab and newline happen
420                          * to be next to each other in ASCII, if tab were
421                          * included a loop would be needed.  Finally, note
422                          * that lots of other processing is done here, so
423                          * this hack won't always work (e.g. the Ann Arbor
424                          * 4080, which uses %B and then %c.)
425                          */
426                                 switch (op) {
427                                 /*
428                                 * Null.  Problem is that our
429                                 * output is, by convention, null terminated.
430                                 */
431                                         case 0:
432                                                 op = 0200; /* Parity should */
433                                                         /* be ignored. */
434                                                 break;
435                                 /*
436                                 * Control D.  Problem is that certain very
437                                 * ancient hardware hangs up on this, so the
438                                 * current(!) UNIX tty driver doesn't xmit
439                                 * control D's.
440                                 */
441                                         case _CHCTRL('d'):
442                                 /*
443                                 * Newline.  Problem is that UNIX will expand
444                                 * this to CRLF.
445                                 */
446                                         case '\n':
447                                                 xp = (onrow ? cursor_down :
448                                                     cursor_right);
449                                         if (onrow && xp && op < lines-1 &&
450                                             cursor_up) {
451                                                 op += 2;
452                                                 xp = cursor_up;
453                                         }
454                                         if (xp && instring ==
455                                             cursor_address) {
456                                                 (void) strcat(added, xp);
457                                                 op--;
458                                         }
459                                         break;
460                                 /*
461                                  * Tab used to be in this group too,
```



```
462                                  * because UNIX might expand it to blanks.
463                                  * We now require that this tab mode be turned
464                                  * off by any program using this routine,
465                                  * or using termcap in general, since some
466                                  * terminals use tab for other stuff, like
467                                  * nondestructive space.  (Filters like ul
468                                  * or vcrt will lose, since they can't stty.)
469                                  * Tab was taken out to get the Ann Arbor
470                                  * 4080 to work.
471                                  */
472                                 }

474                                 /* LINTED */
475                                 *outp++ = (char)op;
476                                 (void) pop(&stk);
477                                 break;

479                         case 'l':
480                                 xp = pop_char_p(&stk);
481                                 push(&stk, strlen(xp));
482                                 break;

484                         case '%':
485                                 *outp++ = c;
486                                 break;

488                         /*
489                         * %i: shorthand for increment first two parms.
490                         * Useful for terminals that start numbering from
491                         * one instead of zero(like ANSI terminals).
492                         */
493                         case 'i':
494                                 p1++;
495                                 p2++;
496                                 break;

498                         /* %pi: push the ith parameter */
499                         case 'p':
500                                 switch (c = *cp++) {
501                                         case '1':
502                                                 push(&stk, p1);
503                                                 break;
504                                         case '2':
505                                                 push(&stk, p2);
506                                                 break;
507                                         case '3':
508                                                 push(&stk, p3);
509                                                 break;
510                                         case '4':
511                                                 push(&stk, p4);
512                                                 break;
513                                         case '5':
514                                                 push(&stk, p5);
515                                                 break;
516                                         case '6':
517                                                 push(&stk, p6);
518                                                 break;
519                                         case '7':
520                                                 push(&stk, p7);
521                                                 break;
522                                         case '8':
523                                                 push(&stk, p8);
524                                                 break;
525                                         case '9':
526                                                 push(&stk, p9);
527                                                 break;
```

```
528                                     default:
529 #ifdef  DEBUG
530                                             if (outf)
531                                                     fprintf(outf, "TPARM:"
532                                                         " bad parm"
533                                                         " number\n");
534 #endif  /* DEBUG */
535                                             free_stack(&stk);
536                                             return (NULL);
537                                     }
538                             onrow = (c == '1');
539                             break;

541                             /* %Pi: pop from stack into variable i (a-z) */
542                             case 'P':
543                                     if (*cp >= 'a' && *cp <= 'z') {
544                                             vars[*cp++ - 'a'] = pop(&stk);
545                                     } else {
546                                             if (*cp >= 'A' && *cp <= 'Z') {
547                                                     regs[*cp++ - 'A'] =
548                                                             /* LINTED */
549                                                             (short) pop(&stk);
550                                             }
551 #ifdef  DEBUG
552                                             else if (outf) {
553                                                     fprintf(outf, "TPARM: bad"
554                                                         " register name\n");
555                                             }
556 #endif  /* DEBUG */
557                                     }
558                                     break;

560                             /* %gi: push variable i (a-z) */
561                             case 'g':
562                                     if (*cp >= 'a' && *cp <= 'z') {
563                                             push(&stk, vars[*cp++ - 'a']);
564                                     } else {
565                                             if (*cp >= 'A' && *cp <= 'Z') {
566                                                     push(&stk, regs[*cp++ - 'A']);
567                                             }
568 #ifdef  DEBUG
569                                             else if (outf) {
570                                                     fprintf(outf, "TPARM: bad"
571                                                         " register name\n");

573                                             }
574 #endif  /* DEBUG */
575                                     }
576                                     break;

578                             /* %'c' : character constant */
579                             case '\'':
580                                     push(&stk, *cp++);
581                                     if (*cp++ != '\'') {
582 #ifdef  DEBUG
583                                             if (outf)
584                                                     fprintf(outf, "TPARM: missing"
585                                                         " closing quote\n");
586 #endif  /* DEBUG */
587                                             free_stack(&stk);
588                                             return (NULL);
589                                     }
590                                     break;

592                             /* %{nn} : integer constant.  */
593                             case '{':
```



```
594                                     op = 0;
595                                     sign = 1;
596                                     if (*cp == '-') {
597                                             sign = -1;
598                                             cp++;
599                                     } else
600                                             if (*cp == '+')
601                                                     cp++;
602                                     while ((c = *cp++) >= '0' && c <= '9') {
603                                             op = 10 * op + c - '0';
604                                     }
605                                     if (c != '}') {
606 #ifdef  DEBUG
607                                             if (outf)
608                                                     fprintf(outf, "TPARM: missing "
609                                                         "closing brace\n");
610 #endif  /* DEBUG */
611                                             free_stack(&stk);
612                                             return (NULL);
613                                     }
614                                     push(&stk, (sign * op));
615                                     break;

617                             /* binary operators */
618                             case '+':
619                                     op2 = pop(&stk);
620                                     op = pop(&stk);
621                                     push(&stk, (op + op2));
622                                     break;
623                             case '-':
624                                     op2 = pop(&stk);
625                                     op = pop(&stk);
626                                     push(&stk, (op - op2));
627                                     break;
628                             case '*':
629                                     op2 = pop(&stk);
630                                     op = pop(&stk);
631                                     push(&stk, (op * op2));
632                                     break;
633                             case '/':
634                                     op2 = pop(&stk);
635                                     op = pop(&stk);
636                                     push(&stk, (op / op2));
637                                     break;
638                             case 'm':
639                                     op2 = pop(&stk);
640                                     op = pop(&stk);
641                                     push(&stk, (op % op2));
642                                     break; /* %m: mod */
643                             case '&':
644                                     op2 = pop(&stk);
645                                     op = pop(&stk);
646                                     push(&stk, (op & op2));
647                                     break;
648                             case '|':
649                                     op2 = pop(&stk);
650                                     op = pop(&stk);
651                                     push(&stk, (op | op2));
652                                     break;
653                             case '^':
654                                     op2 = pop(&stk);
655                                     op = pop(&stk);
656                                     push(&stk, (op ^ op2));
657                                     break;
658                             case '=':
659                                     op2 = pop(&stk);
```

```
660                                 op = pop(&stk);
661                                 push(&stk, (op == op2));
662                                 break;
663                         case '>':
664                                 op2 = pop(&stk);
665                                 op = pop(&stk);
666                                 push(&stk, (op > op2));
667                                 break;
668                         case '<':
669                                 op2 = pop(&stk);
670                                 op = pop(&stk);
671                                 push(&stk, (op < op2));
672                                 break;
673                         case 'A':
674                                 op2 = pop(&stk);
675                                 op = pop(&stk);
676                                 push(&stk, (op && op2));
677                                 break; /* AND */
678                         case 'O':
679                                 op2 = pop(&stk);
680                                 op = pop(&stk);
681                                 push(&stk, (op || op2));
682                                 break; /* OR */

684                         /* Unary operators. */
685                         case '!':
686                                 push(&stk, !pop(&stk));
687                                 break;
688                         case '~':
689                                 push(&stk, ~pop(&stk));
690                                 break;

692                         /* Sorry, no unary minus, because minus is binary. */

694                         /*
695                         * If-then-else.  Implemented by a low level hack of
696                         * skipping forward until the match is found, counting
697                         * nested if-then-elses.
698                         */
699                         case '?':       /* IF - just a marker */
700                                 break;

702                         case 't':       /* THEN - branch if false */
703                                 if (!pop(&stk))
704                                         cp = _branchto(cp, 'e');
705                                         break;

707                         case 'e':       /* ELSE - branch to ENDIF */
708                                 cp = _branchto(cp, ';');
709                                 break;

711                         case ';':       /* ENDIF - just a marker */
712                                 break;

714                         default:
715 #ifdef  DEBUG
716                                 if (outf)
717                                         fprintf(outf, "TPARM: bad % "
718                                             "sequence\n");
719 #endif  /* DEBUG */
720                                 free_stack(&stk);
721                                 return (NULL);
722                 }
723         }
724         (void) strcpy(outp, added);
725         free_stack(&stk);
```



```
726         return (result);
727 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   70897 Tue May 26 15:56:27 2015
new/usr/src/lib/libdtrace/common/dt_cc.c
5396 gcc 4.8.2 longjmp errors for cscope-fast
**********************************************************
```

```
/*
 * 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) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013, Joyent Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright 2015 Gary Mills
 */

/*
 * DTrace D Language Compiler
 *
 * The code in this source file implements the main engine for the D language
 * compiler.  The driver routine for the compiler is dt_compile(), below.  The
 * compiler operates on either stdio FILEs or in-memory strings as its input
 * and can produce either dtrace_prog_t structures from a D program or a single
 * dtrace_difo_t structure from a D expression.  Multiple entry points are
 * provided as wrappers around dt_compile() for the various input/output pairs.
 * The compiler itself is implemented across the following source files:
 *
 * dt_lex.l - lex scanner
 * dt_grammar.y - yacc grammar
 * dt_parser.c - parse tree creation and semantic checking
 * dt_decl.c - declaration stack processing
 * dt_xlator.c - D translator lookup and creation
 * dt_ident.c - identifier and symbol table routines
 * dt_pragma.c - #pragma processing and D pragmas
 * dt_printf.c - D printf() and printa() argument checking and processing
 * dt_cc.c - compiler driver and dtrace_prog_t construction
 * dt_cg.c - DIF code generator
 * dt_as.c - DIF assembler
 * dt_dof.c - dtrace_prog_t -> DOF conversion
 *
 * Several other source files provide collections of utility routines used by
 * these major files.  The compiler itself is implemented in multiple passes:
 *
 * (1) The input program is scanned and parsed by dt_lex.l and dt_grammar.y
 *     and parse tree nodes are constructed using the routines in dt_parser.c.
 *     This node construction pass is described further in dt_parser.c.
 *
 * (2) The parse tree is "cooked" by assigning each clause a context (see the
 *     routine dt_setcontext(), below) based on its probe description and then
```



```
 *     recursively descending the tree performing semantic checking.  The cook
 *     routines are also implemented in dt_parser.c and described there.
 *
 * (3) For actions that are DIF expression statements, the DIF code generator
 *     and assembler are invoked to create a finished DIFO for the statement.
 *
 * (4) The dtrace_prog_t data structures for the program clauses and actions
 *     are built, containing pointers to any DIFOs created in step (3).
 *
 * (5) The caller invokes a routine in dt_dof.c to convert the finished program
 *     into DOF format for use in anonymous tracing or enabling in the kernel.
 *
 * In the implementation, steps 2-4 are intertwined in that they are performed
 * in order for each clause as part of a loop that executes over the clauses.
 *
 * The D compiler currently implements nearly no optimization.  The compiler
 * implements integer constant folding as part of pass (1), and a set of very
 * simple peephole optimizations as part of pass (3).  As with any C compiler,
 * a large number of optimizations are possible on both the intermediate data
 * structures and the generated DIF code.  These possibilities should be
 * investigated in the context of whether they will have any substantive effect
 * on the overall DTrace probe effect before they are undertaken.
 */

#include <sys/types.h>
#include <sys/wait.h>
#include <sys/sysmacros.h>

#include <assert.h>
#include <strings.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <ucontext.h>
#include <limits.h>
#include <ctype.h>
#include <dirent.h>
#include <dt_module.h>
#include <dt_program.h>
#include <dt_provider.h>
#include <dt_printf.h>
#include <dt_pid.h>
#include <dt_grammar.h>
#include <dt_ident.h>
#include <dt_string.h>
#include <dt_impl.h>

static const dtrace_diftype_t dt_void_rtype = {
        DIF_TYPE_CTF, CTF_K_INTEGER, 0, 0, 0
};
______unchanged_portion_omitted_

static void *
dt_compile(dtrace_hdl_t *dtp, int context, dtrace_probespec_t pspec, void *arg,
    uint_t cflags, int argc, char *const argv[], FILE *fp, const char *s)
{
        dt_node_t *dnp;
        dt_decl_t *ddp;
        dt_pcb_t pcb;
        void *volatile rv;
        void *rv;
        int err;

        if ((fp == NULL && s == NULL) || (cflags & ~DTRACE_C_MASK) != 0) {
```

```
2343                (void) dt_set_errno(dtp, EINVAL);
2344                return (NULL);
2345        }

2347        if (dt_list_next(&dtp->dt_lib_path) != NULL && dt_load_libs(dtp) != 0)
2348                return (NULL); /* errno is set for us */

2350        if (dtp->dt_globals->dh_nelems != 0)
2351                (void) dt_idhash_iter(dtp->dt_globals, dt_idreset, NULL);

2353        if (dtp->dt_tls->dh_nelems != 0)
2354                (void) dt_idhash_iter(dtp->dt_tls, dt_idreset, NULL);

2356        if (fp && (cflags & DTRACE_C_CPP) && (fp = dt_preproc(dtp, fp)) == NULL)
2357                return (NULL); /* errno is set for us */

2359        dt_pcb_push(dtp, &pcb);

2361        pcb.pcb_fileptr = fp;
2362        pcb.pcb_string = s;
2363        pcb.pcb_strptr = s;
2364        pcb.pcb_strlen = s ? strlen(s) : 0;
2365        pcb.pcb_sargc = argc;
2366        pcb.pcb_sargv = argv;
2367        pcb.pcb_sflagv = argc ? calloc(argc, sizeof (ushort_t)) : NULL;
2368        pcb.pcb_pspec = pspec;
2369        pcb.pcb_cflags = dtp->dt_cflags | cflags;
2370        pcb.pcb_amin = dtp->dt_amin;
2371        pcb.pcb_yystate = -1;
2372        pcb.pcb_context = context;
2373        pcb.pcb_token = context;

2375        if (context != DT_CTX_DPROG)
2376                yybegin(YYS_EXPR);
2377        else if (cflags & DTRACE_C_CTL)
2378                yybegin(YYS_CONTROL);
2379        else
2380                yybegin(YYS_CLAUSE);

2382        if ((err = setjmp(yypcb->pcb_jmpbuf)) != 0)
2383                goto out;

2385        if (yypcb->pcb_sargc != 0 && yypcb->pcb_sflagv == NULL)
2386                longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2388        yypcb->pcb_idents = dt_idhash_create("ambiguous", NULL, 0, 0);
2389        yypcb->pcb_locals = dt_idhash_create("clause local", NULL,
2390            DIF_VAR_OTHER_UBASE, DIF_VAR_OTHER_MAX);

2392        if (yypcb->pcb_idents == NULL || yypcb->pcb_locals == NULL)
2393                longjmp(yypcb->pcb_jmpbuf, EDT_NOMEM);

2395        /*
2396         * Invoke the parser to evaluate the D source code.  If any errors
2397         * occur during parsing, an error function will be called and we
2398         * will longjmp back to pcb_jmpbuf to abort.  If parsing succeeds,
2399         * we optionally display the parse tree if debugging is enabled.
2400         */
2401        if (yyparse() != 0 || yypcb->pcb_root == NULL)
2402                xyerror(D_EMPTY, "empty D program translation unit\n");

2404        yybegin(YYS_DONE);

2406        if (cflags & DTRACE_C_CTL)
2407                goto out;
```



```
2409        if (context != DT_CTX_DTYPE && DT_TREEDUMP_PASS(dtp, 1))
2410                dt_node_printr(yypcb->pcb_root, stderr, 0);

2412        if (yypcb->pcb_pragmas != NULL)
2413                (void) dt_idhash_iter(yypcb->pcb_pragmas, dt_idpragma, NULL);

2415        if (argc > 1 && !(yypcb->pcb_cflags & DTRACE_C_ARGREF) &&
2416            !(yypcb->pcb_sflagv[argc - 1] & DT_IDFLG_REF)) {
2417                xyerror(D_MACRO_UNUSED, "extraneous argument '%s' ($%d is "
2418                    "not referenced)\n", yypcb->pcb_sargv[argc - 1], argc - 1);
2419        }

2421        /*
2422         * If we have successfully created a parse tree for a D program, loop
2423         * over the clauses and actions and instantiate the corresponding
2424         * libdtrace program.  If we are parsing a D expression, then we
2425         * simply run the code generator and assembler on the resulting tree.
2426         */
2427        switch (context) {
2428        case DT_CTX_DPROG:
2429                assert(yypcb->pcb_root->dn_kind == DT_NODE_PROG);

2431                if ((dnp = yypcb->pcb_root->dn_list) == NULL &&
2432                    !(yypcb->pcb_cflags & DTRACE_C_EMPTY))
2433                        xyerror(D_EMPTY, "empty D program translation unit\n");

2435                if ((yypcb->pcb_prog = dt_program_create(dtp)) == NULL)
2436                        longjmp(yypcb->pcb_jmpbuf, dtrace_errno(dtp));

2438                for (; dnp != NULL; dnp = dnp->dn_list) {
2439                        switch (dnp->dn_kind) {
2440                        case DT_NODE_CLAUSE:
2441                                dt_compile_clause(dtp, dnp);
2442                                break;
2443                        case DT_NODE_XLATOR:
2444                                if (dtp->dt_xlatemode == DT_XL_DYNAMIC)
2445                                        dt_compile_xlator(dnp);
2446                                break;
2447                        case DT_NODE_PROVIDER:
2448                                (void) dt_node_cook(dnp, DT_IDFLG_REF);
2449                                break;
2450                        }
2451                }

2453                yypcb->pcb_prog->dp_xrefs = yypcb->pcb_asxrefs;
2454                yypcb->pcb_prog->dp_xrefslen = yypcb->pcb_asxreflen;
2455                yypcb->pcb_asxrefs = NULL;
2456                yypcb->pcb_asxreflen = 0;

2458                rv = yypcb->pcb_prog;
2459                break;

2461        case DT_CTX_DEXPR:
2462                (void) dt_node_cook(yypcb->pcb_root, DT_IDFLG_REF);
2463                dt_cg(yypcb, yypcb->pcb_root);
2464                rv = dt_as(yypcb);
2465                break;

2467        case DT_CTX_DTYPE:
2468                ddp = (dt_decl_t *)yypcb->pcb_root; /* root is really a decl */
2469                err = dt_decl_type(ddp, arg);
2470                dt_decl_free(ddp);

2472                if (err != 0)
2473                        longjmp(yypcb->pcb_jmpbuf, EDT_COMPILER);
```

```
2475                 rv = NULL;
2476                 break;
2477         }

2479 out:
2480         if (context != DT_CTX_DTYPE && yypcb->pcb_root != NULL &&
2481             DT_TREEDUMP_PASS(dtp, 3))
2482                 dt_node_printr(yypcb->pcb_root, stderr, 0);

2484         if (dtp->dt_cdefs_fd != -1 && (ftruncate64(dtp->dt_cdefs_fd, 0) == -1 ||
2485             lseek64(dtp->dt_cdefs_fd, 0, SEEK_SET) == -1 ||
2486             ctf_write(dtp->dt_cdefs->dm_ctfp, dtp->dt_cdefs_fd) == CTF_ERR))
2487                 dt_dprintf("failed to update CTF cache: %s\n", strerror(errno));

2489         if (dtp->dt_ddefs_fd != -1 && (ftruncate64(dtp->dt_ddefs_fd, 0) == -1 ||
2490             lseek64(dtp->dt_ddefs_fd, 0, SEEK_SET) == -1 ||
2491             ctf_write(dtp->dt_ddefs->dm_ctfp, dtp->dt_ddefs_fd) == CTF_ERR))
2492                 dt_dprintf("failed to update CTF cache: %s\n", strerror(errno));

2494         if (yypcb->pcb_fileptr && (cflags & DTRACE_C_CPP))
2495                 (void) fclose(yypcb->pcb_fileptr); /* close dt_preproc() file */

2497         dt_pcb_pop(dtp, err);
2498         (void) dt_set_errno(dtp, err);
2499         return (err ? NULL : rv);
2500 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   25943 Tue May 26 15:56:28 2015
new/usr/src/lib/libxcurses/src/libc/xcurses/doupdate.c
5396 gcc 4.8.2 longjmp errors for cscope-fast
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (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 2015 Gary Mills
 * Copyright (c) 1995, by Sun Microsystems, Inc.
 * All rights reserved.
 */

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

/*
 * doupdate.c
 *
 * XCurses Library
 *
 * Copyright 1990, 1995 by Mortice Kern Systems Inc.  All rights reserved.
 *
 */

#ifdef M_RCSID
#ifndef lint
static char const rcsID[] = "$Header: /rd/src/libc/xcurses/rcs/doupdate.c 1.9 19
#endif
#endif

#include <private.h>
#include <string.h>
#include <setjmp.h>
#include <signal.h>

#undef SIGTSTP

/*
 * Disable typeahead trapping because it slow down updated dramatically
 * on MPE/iX.
 */
#ifdef MPE_STUB
#undef M_CURSES_TYPEAHEAD
#endif

/*
 * This value is the ideal length for the cursor addressing sequence
```



```
 * being four bytes long, ie. "<escape><cursor addressing code><row><col>".
 * eg. VT52 - "\EYrc" or ADM3A - "\E=rc"
 */
#define JUMP_SIZE       4

/*
 * This value is the ideal length for the clear-to-eol sequence
 * being two bytes long, ie "<escape><clear eol code>".
 */
#define CEOL_SIZE       2

#define GOTO(r,c)       (__m_mvcur(curscr->_cury, curscr->_curx,r,c,__m_outc),\
                        curscr->_cury = r, curscr->_curx = c)

typedef struct cost_op {
        short cost;
        short op;
} lcost;
```

*unchanged_portion_omitted_*

```
/*f
 * Send all changes made to _newscr to the physical terminal.
 *
 * If idlok() is set TRUE then doupdate will try and use hardware insert
 * and delete line sequences in an effort to optimize output.  idlok()
 * should really only be used in applications that want a proper scrolling
 * effect.
 *
 * Added scroll heuristic to handle special case where a full size window
 * with full size scroll region, will scroll the window and replace dirty
 * lines instead of performing usual cost/script operations.
 */
int
doupdate()
{
#ifdef SIGTSTP
        int (*oldsig)(int) = signal(SIGTSTP, SIG_IGN);
#endif

#ifdef M_CURSES_TYPEAHEAD
        unsigned char cc;
        volatile int min, time, icanon;
        int min, time, icanon;

        if (__m_screen->_flags & S_ISATTY) {
                /* Set up non-blocking input for typeahead trapping. */
                min = cur_term->_prog.c_cc[VMIN];
                time = cur_term->_prog.c_cc[VTIME];
                icanon = cur_term->_prog.c_lflag & ICANON;

                cur_term->_prog.c_cc[VMIN] = 0;
                cur_term->_prog.c_cc[VTIME] = 0;
                cur_term->_prog.c_lflag &= ~ICANON;

                (void) tcsetattr(__m_screen->_kfd, TCSANOW, &cur_term->_prog);
        }
#endif /* M_CURSES_TYPEAHEAD */

#ifdef M_CURSES_TRACE
        __m_trace(
                "doupdate(void) using %s algorithm.",
                (__m_screen->_flags & S_INS_DEL_LINE) ? "complex" : "simple"
        );
#endif

        newscr = __m_screen->_newscr;
```

```
940         if (__m_screen->_flags & S_ENDWIN) {
941                 /* Return from temporary escape done with endwin(). */
942                 __m_screen->_flags &= ~S_ENDWIN;

944                 (void) reset_prog_mode();
945                 if (enter_ca_mode != (char *) 0)
946                         (void) tputs(enter_ca_mode, 1, __m_outc);
947                 if (keypad_xmit != (char *) 0)
948                         (void) tputs(keypad_xmit, 1, __m_outc);
949                 if (ena_acs != (char *) 0)
950                         (void) tputs(ena_acs, 1, __m_outc);

952                 /* Force redraw of screen. */
953                 newscr->_flags |= W_CLEAR_WINDOW;
954         }

956 #ifdef M_CURSES_TYPEAHEAD
957         if (setjmp(breakout) == 0) {
958                 if ((__m_screen->_flags & S_ISATTY)
959                 && read(__m_screen->_kfd, &cc, sizeof cc) == sizeof cc) {
960                         (void) ungetch(cc);
961                         longjmp(breakout, 1);
962                 }
963 #endif /* M_CURSES_TYPEAHEAD */

965                 /* When redrawwing a window, we not only assume that line
966                  * noise may have lost characters, but line noise may have
967                  * generated bogus characters on the screen outside the
968                  * the window in question, in which case redraw the entire
969                  * screen to be sure.
970                  */
971                 if (newscr->_flags & (W_CLEAR_WINDOW | W_REDRAW_WINDOW)) {
972                         clear_bottom(0);
973                         newscr->_flags &= ~W_CLEAR_WINDOW;
974                         (void) wtouchln(newscr, 0, newscr->_maxy, 1);
975                 }

977                 if (newscr->_flags & W_REDRAW_WINDOW)
978                         simple();
979 #if 0           /* This first expression, of undefined section, is useless
980                  * since newscr->_scroll is unsigned and never LT zero.
981                  */
982                 else if (newscr->_scroll < 0 && scroll_dn(-newscr->_scroll))
983 #else
984                 else if (scroll_dn(-newscr->_scroll))
985 #endif
986                         ;
987                 else if (0 < newscr->_scroll && scroll_up(newscr->_scroll))
988                         ;
989                 else if (__m_screen->_flags & S_INS_DEL_LINE)
990                         complex();
991                 else
992                         simple();

994                 if (!(newscr->_flags & W_LEAVE_CURSOR))
995                         GOTO(newscr->_cury, newscr->_curx);

997                 if (!(curscr->_flags & W_FLUSH))
998                         (void) fflush(__m_screen->_of);
999 #ifdef M_CURSES_TYPEAHEAD
1000         }

1002        if (__m_screen->_flags & S_ISATTY) {
1003                /* Restore previous input mode. */
1004                cur_term->_prog.c_cc[VMIN] = min;
```



```
1005                cur_term->_prog.c_cc[VTIME] = time;
1006                cur_term->_prog.c_lflag |= icanon;

1008                (void) tcsetattr(__m_screen->_kfd,TCSANOW,&cur_term->_prog);
1009        }
1010 #endif /* M_CURSES_TYPEAHEAD */

1012        newscr->_scroll = curscr->_scroll = 0;
1013 #ifdef SIGTSTP
1014        signal(SIGTSTP, oldsig);
1015 #endif

1017        return __m_return_code("doupdate", OK);
1018 }
```

unchanged_portion_omitted_

```
**********************************************************
   16379 Tue May 26 15:56:28 2015
new/usr/src/tools/cscope-fast/display.c
5396 gcc 4.8.2 longjmp errors for cscope-fast
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*      Copyright (c) 1988 AT&T */
/*        All Rights Reserved   */


/*
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2015 Gary Mills
 */

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

/*
 *      cscope - interactive C symbol cross-reference
 *
 *      display functions
 */

#include "global.h"
#include "version.h"    /* FILEVERSION and FIXVERSION */
#include <curses.h>     /* COLS and LINES */
#include <setjmp.h>     /* jmp_buf */
#include <string.h>
#include <errno.h>

/* see if the function column should be displayed */
#define displayfcn()    (field <= ASSIGN)

#define MINCOLS 68      /* minimum columns for 3 digit Lines message numbers */

int     *displine;              /* screen line of displayed reference */
int     disprefs;               /* displayed references */
int     field;                  /* input field */
unsigned fldcolumn;             /* input field column */
int     mdisprefs;              /* maximum displayed references */
int     selectlen;              /* selection number field length */
int     nextline;               /* next line to be shown */
int     topline = 1;            /* top line of page */
int     bottomline;             /* bottom line of page */
int     totallines;             /* total reference lines */
```



```
FILE    *refsfound;             /* references found file */
FILE    *nonglobalrefs;         /* non-global references file */

static  int     fldline;                /* input field line */
static  int     subsystemlen;           /* OGS subsystem name display */
                                        /* field length */
static  int     booklen;                /* OGS book name display field length */
static  int     filelen;                /* file name display field length */
static  int     fcnlen;                 /* function name display field length */
static  jmp_buf env;                    /* setjmp/longjmp buffer */
static  int     lastdispline;           /* last displayed reference line */
static  char    lastmsg[MSGLEN + 1];    /* last message displayed */
static  int     numlen;                 /* line number display field length */
static  char    depthstring[] = "Depth: ";
static  char    helpstring[] = "Press the ? key for help";


typedef char *(*FP)();  /* pointer to function returning a character pointer */

static  struct  {
        char    *text1;
        char    *text2;
        FP      findfcn;
        enum {
                EGREP,
                REGCMP
        } patterntype;
} fields[FIELDS + 1] = {
_____unchanged_portion_omitted_

BOOL
search(void)
{
        char    *egreperror = NULL;     /* egrep error message */
        FINDINIT rc = NOERROR;          /* findinit return code */
        SIGTYPE (*volatile savesig)() = SIG_DFL; /* old value of signal */
        SIGTYPE (*savesig)();           /* old value of signal */
        FP      f;                      /* searching function */
        char    *s;
        int     c;

        /* note: the pattern may have been a cscope argument */
        if (caseless == YES) {
                for (s = pattern; *s != '\0'; ++s) {
                        *s = tolower(*s);
                }
        }
        /* open the references found file for writing */
        if (writerefsfound() == NO) {
                return (NO);
        }
        /* find the pattern - stop on an interrupt */
        if (linemode == NO) {
                putmsg("Searching");
        }
        initprogress();
        if (setjmp(env) == 0) {
                savesig = signal(SIGINT, jumpback);
                f = fields[field].findfcn;
                if (fields[field].patterntype == EGREP) {
                        egreperror = (*f)(pattern);
                } else {
                        if ((nonglobalrefs = fopen(temp2, "w")) == NULL) {
                                cannotopen(temp2);
                                return (NO);
                        }
```

```
                        if ((rc = findinit()) == NOERROR) {
                                (void) dbseek(0L); /* goto the first block */
                                (*f)();
                                findcleanup();

                                /* append the non-global references */
                                (void) freopen(temp2, "r", nonglobalrefs);
                                while ((c = getc(nonglobalrefs)) != EOF) {
                                        (void) putc(c, refsfound);
                                }
                        }
                        (void) fclose(nonglobalrefs);
                }
        }
        (void) signal(SIGINT, savesig);
        /* reopen the references found file for reading */
        (void) freopen(temp1, "r", refsfound);
        nextline = 1;
        totallines = 0;

        /* see if it is empty */
        if ((c = getc(refsfound)) == EOF) {
                if (egreperror != NULL) {
                        (void) sprintf(lastmsg, "Egrep %s in this pattern: %s",
                            egreperror, pattern);
                } else if (rc == NOTSYMBOL) {
                        (void) sprintf(lastmsg, "This is not a C symbol: %s",
                            pattern);
                } else if (rc == REGCMPERROR) {
                        (void) sprintf(lastmsg,
                            "Error in this regcmp(3X) regular expression: %s",
                            pattern);
                } else {
                        (void) sprintf(lastmsg, "Could not find the %s: %s",
                            fields[field].text2, pattern);
                }
                return (NO);
        }
        /* put back the character read */
        (void) ungetc(c, refsfound);

        countrefs();
        return (YES);
}
```

unchanged_portion_omitted_

```
**********************************************************
    5610 Tue May 26 15:56:28 2015
new/usr/src/tools/cscope-fast/input.c
5396 gcc 4.8.2 longjmp errors for cscope-fast
**********************************************************
```

```
/*
 * 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 2015 Gary Mills
 * Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
 */

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

/*
 *      cscope - interactive C symbol cross-reference
 *
 *      terminal input functions
 */

#include "global.h"
#include <curses.h>     /* KEY_BACKSPACE, KEY_BREAK, and KEY_ENTER */
#include <setjmp.h>     /* jmp_buf */

static  jmp_buf env;            /* setjmp/longjmp buffer */
static  int     prevchar;       /* previous, ungotten character */

/* catch the interrupt signal */

/*ARGSUSED*/
SIGTYPE
catchint(int sig)
{
        (void) signal(SIGINT, catchint);
        longjmp(env, 1);
}
```

_____***unchanged_portion_omitted_***

```
/* get a character from the terminal */

int
mygetch(void)
{
        SIGTYPE (*volatile savesig)() = SIG_DFL; /* old value of signal */
        SIGTYPE (*savesig)();           /* old value of signal */
        int     c;
```



```
        /* change an interrupt signal to a break key character */
        if (setjmp(env) == 0) {
                savesig = signal(SIGINT, catchint);
                (void) refresh();       /* update the display */
                reinitmouse();  /* curses can change the menu number */
                if (prevchar) {
                        c = prevchar;
                        prevchar = 0;
                } else {
                        c = getch();    /* get a character from the terminal */
                }
        } else {        /* longjmp to here from signal handler */
                c = KEY_BREAK;
        }
        (void) signal(SIGINT, savesig);
        return (c);
}
```

_____***unchanged_portion_omitted_***