```
**********************************************************
   72740 Mon Jun  5 10:46:46 2017
new/usr/src/cmd/ls/ls.c
8175/8183: memory leak fixes + incorrect test of dereferenced pointer
**********************************************************
______unchanged_portion_omitted_

 219 /*
 220  * A numbuf_t is used when converting a number to a string representation
 221  */
 222 typedef char numbuf_t[NUMBER_WIDTH];

 224 static struct dchain *dfirst;   /* start of the dir chain */
 225 static struct dchain *cdfirst;  /* start of the current dir chain */
 226 static struct dchain *dtemp;    /* temporary - used for linking */
 227 static char *curdir;            /* the current directory */

 229 static int      first = 1;      /* true if first line is not yet printed */
 230 static int      nfiles = 0;     /* number of flist entries in current use */
 231 static int      nargs = 0;      /* number of flist entries used for arguments */
 232 static int      maxfils = 0;    /* number of flist/lbuf entries allocated */
 233 static int      maxn = 0;       /* number of flist entries with lbufs asigned */
 234 static int      quantn = 64;    /* allocation growth quantum */
 235 static size_t   hlbfsz = 1;

 237 static struct lbuf      *nxtlbf;        /* ptr to next lbuf to be assigned */
 238 static struct lbuf      **hlbf;         /* lbuf bookkeeping */
 239 static struct lbuf      **flist;        /* ptr to list of lbuf pointers */
 240 static struct lbuf      *gstat(char *, int, struct ditem *);
 241 static char             *getname(uid_t);
 242 static char             *getgroup(gid_t);
 243 static char             *makename(char *, char *);
 244 static void             pentry(struct lbuf *);
 245 static void             column(void);
 246 static void             pmode(mode_t aflag);
 247 static void             selection(int *);
 248 static void             new_line(void);
 249 static void             rddir(char *, struct ditem *);
 250 static int              strcol(unsigned char *);
 251 static void             pem(struct lbuf **, struct lbuf **, int);
 252 static void             pdirectory(char *, int, int, int, struct ditem *);
 253 static struct cachenode *findincache(struct cachenode **, long);
 254 static void             freecachenodes(void);
 255 static void             csi_pprintf(unsigned char *);
 256 static void             pprintf(char *, char *);
 257 static int              compar(struct lbuf **pp1, struct lbuf **pp2);
 258 static char             *number_to_scaled_string(numbuf_t buf,
 259                             unsigned long long number,
 260                             long scale);
 261 static void             record_ancestry(char *, struct stat *, struct lbuf *,
 262                             int, struct ditem *);
 263 static void             ls_color_init(void);
 264 static ls_color_t       *ls_color_find(const char *, mode_t);
 265 static void             ls_start_color(ls_color_t *);
 266 static void             ls_end_color(void);

 268 static int              aflg;
 269 static int              atflg;
 270 static int              bflg;
 271 static int              cflg;
 272 static int              dflg;
 273 static int              eflg;
 274 static int              fflg;
 275 static int              gflg;
 276 static int              hflg;
 277 static int              iflg;
```



```
 278 static int              lflg;
 279 static int              mflg;
 280 static int              nflg;
 281 static int              oflg;
 282 static int              pflg;
 283 static int              qflg;
 284 static int              rflg = 1; /* init to 1 for special use in compar */
 285 static int              sflg;
 286 static int              tflg;
 287 static int              uflg;
 288 static int              Uflg;
 289 static int              wflg;
 290 static int              xflg;
 291 static int              Aflg;
 292 static int              Bflg;
 293 static int              Cflg;
 294 static int              Eflg;
 295 static int              Fflg;
 296 static int              Hflg;
 297 static int              Lflg;
 298 static int              Rflg;
 299 static int              Sflg;
 300 static int              vflg;
 301 static int              Vflg;
 302 static int              saflg;          /* boolean extended system attr. */
 303 static int              sacnt;          /* number of extended system attr. */
 304 static int              copt;
 305 static int              vopt;
 306 static int              tmflg;          /* create time ext. system attr. */
 307 static int              ctm;
 308 static int              atm;
 309 static int              mtm;
 310 static int              crtm;
 311 static int              alltm;
 312 static long             hscale;
 313 static mode_t           flags;
 314 static int              err = 0;        /* Contains return code */
 315 static int              colorflg;
 316 static int              file_typeflg;
 317 static int              noflist = 0;

 319 static uid_t            lastuid = (uid_t)-1;
 320 static gid_t            lastgid = (gid_t)-1;
 321 static char             *lastuname = NULL;
 322 static char             *lastgname = NULL;

 324 /* statreq > 0 if any of sflg, (n)lflg, tflg, Sflg, colorflg are on */
 325 static int              statreq;

 327 static uint64_t         block_size = 1;
 328 static char             *dotp = ".";

 330 static u_longlong_t     tblocks; /* number of blocks of files in a directory */
 331 static time_t           year, now;

 333 static int              num_cols = 80;
 334 static int              colwidth;
 335 static int              filewidth;
 336 static int              fixedwidth;
 337 static int              nomocore;
 338 static int              curcol;

 340 static struct   winsize win;

 342 /* if time_fmt_new is left NULL, time_fmt_old is used for all times */
 343 static const char       *time_fmt_old = FORMAT_OLD;     /* non-recent files */
```

```
344 static const char       *time_fmt_new = FORMAT_NEW;     /* recent files */
345 static int              time_custom;    /* != 0 if a custom format */
346 static char     time_buf[FMTSIZE];      /* array to hold day and time */

348 static int              lsc_debug;
349 static ls_color_t       *lsc_match;
350 static ls_color_t       *lsc_colors;
351 static size_t           lsc_ncolors;
352 static char             *lsc_bold;
353 static char             *lsc_underline;
354 static char             *lsc_blink;
355 static char             *lsc_reverse;
356 static char             *lsc_concealed;
357 static char             *lsc_none;
358 static char             *lsc_setfg;
359 static char             *lsc_setbg;
360 static ls_color_t       *lsc_orphan;

362 #define NOTWORKINGDIR(d, l)     (((l) < 2) || \
363                                     (strcmp((d) + (l) - 2, "/.") != 0))

365 #define NOTPARENTDIR(d, l)      (((l) < 3) || \
366                                     (strcmp((d) + (l) - 3, "/..") != 0))
367 /* Extended system attributes support */
368 static int get_sysxattr(char *, struct lbuf *);
369 static void set_sysattrb_display(char *, boolean_t, struct lbuf *);
370 static void set_sysattrtm_display(char *, struct lbuf *);
371 static void format_time(time_t, time_t);
372 static void print_time(struct lbuf *);
373 static void format_attrtime(struct lbuf *);
374 static void *xmalloc(size_t, struct lbuf *);
375 static void free_sysattr(struct lbuf *);
376 static nvpair_t *pair;
377 static nvlist_t *response;
378 static int acl_err;

380 const struct option long_options[] = {
381         { "all", no_argument, NULL, 'a' },
382         { "almost-all", no_argument, NULL, 'A' },
383         { "escape", no_argument, NULL, 'b' },
384         { "classify", no_argument, NULL, 'F' },
385         { "human-readable", no_argument, NULL, 'h' },
386         { "dereference", no_argument, NULL, 'L' },
387         { "dereference-command-line", no_argument, NULL, 'H' },
388         { "ignore-backups", no_argument, NULL, 'B' },
389         { "inode", no_argument, NULL, 'i' },
390         { "numeric-uid-gid", no_argument, NULL, 'n' },
391         { "no-group", no_argument, NULL, 'o' },
392         { "hide-control-chars", no_argument, NULL, 'q' },
393         { "reverse", no_argument, NULL, 'r' },
394         { "recursive", no_argument, NULL, 'R' },
395         { "size", no_argument, NULL, 's' },
396         { "width", required_argument, NULL, 'w' },

398         /* no short options for these */
399         { "block-size", required_argument, NULL, 0 },
400         { "full-time", no_argument, NULL, 0 },
401         { "si", no_argument, NULL, 0 },
402         { "color", optional_argument, NULL, 0 },
403         { "colour", optional_argument, NULL, 0},
404         { "file-type", no_argument, NULL, 0 },
405         { "time-style", required_argument, NULL, 0 },

407         {0, 0, 0, 0}
408 };
```



```
410 int
411 main(int argc, char *argv[])
412 {
413         int             c;
414         int             i;
415         int             width;
416         int             amino = 0;
417         int             opterr = 0;
418         int             option_index = 0;
419         char            *told = NULL;
420         struct lbuf     *ep;
421         struct lbuf     lb;
422         struct ditem    *myinfo = NULL;

424         (void) setlocale(LC_ALL, "");
425 #if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
426 #define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
427 #endif
428         (void) textdomain(TEXT_DOMAIN);
429 #ifdef STANDALONE
430         if (argv[0][0] == '\0')
431                 argc = getargv("ls", &argv, 0);
432 #endif

434         lb.lmtime.tv_sec = time(NULL);
435         lb.lmtime.tv_nsec = 0;
436         year = lb.lmtime.tv_sec - 6L*30L*24L*60L*60L; /* 6 months ago */
437         now = lb.lmtime.tv_sec + 60;
438         if (isatty(1)) {
439                 Cflg = 1;
440                 mflg = 0;
441         }

443         while ((c = getopt_long(argc, argv,
444             "+aAbBcCdeEfFghHiklLmnopqrRsStuUw:x1@vV/:%:", long_options,
445             &option_index)) != -1)
446                 switch (c) {
447                 case 0:
448                         /* non-short options */
449                         if (strcmp(long_options[option_index].name,
450                             "color") == 0 ||
451                             strcmp(long_options[option_index].name,
452                             "colour") == 0) {
453                                 if (optarg == NULL ||
454                                     strcmp(optarg, "always") == 0 ||
455                                     strcmp(optarg, "yes") == 0 ||
456                                     strcmp(optarg, "force") == 0) {
457                                         colorflg++;
458                                         statreq++;
459                                         continue;
460                                 }

462                                 if (strcmp(optarg, "auto") == 0 ||
463                                     strcmp(optarg, "tty") == 0 ||
464                                     strcmp(optarg, "if-tty") == 0) {
465                                         if (isatty(1) == 1) {
466                                                 colorflg++;
467                                                 statreq++;
468                                         }
469                                         continue;
470                                 }

472                                 if (strcmp(optarg, "never") == 0 ||
473                                     strcmp(optarg, "no") == 0 ||
474                                     strcmp(optarg, "none") == 0) {
475                                         colorflg = 0;
```

```
476                                     continue;
477                             }
478                             (void) fprintf(stderr,
479                                 gettext("Invalid argument '%s' for "
480                                 "--color\n"), optarg);
481                             ++opterr;
482                             continue;
483                     }

485                     if (strcmp(long_options[option_index].name,
486                         "si") == 0) {
487                             hflg++;
488                             hscale = 1000;
489                             continue;
490                     }

492                     if (strcmp(long_options[option_index].name,
493                         "block-size") == 0) {
494                             size_t scale_len = strlen(optarg);
495                             uint64_t scale = 1;
496                             uint64_t kilo = 1024;
497                             char scale_c;

499                             if (scale_len == 0) {
500                                     (void) fprintf(stderr, gettext(
501                                         "invalid block size \'%s\'\n"),
502                                         optarg);
503                                     exit(1);
504                             }

506                             scale_c = optarg[scale_len - 1];
507                             if (scale_c == 'B') {
508                                     /* need at least digit, scale, B */
509                                     if (scale_len < 3) {
510                                             (void) fprintf(stderr, gettext(
511                                                 "invalid block size "
512                                                 "\'%s\'\n"), optarg);
513                                             exit(1);
514                                     }
515                                     kilo = 1000;
516                                     scale_c = optarg[scale_len - 2];
517                                     if (isdigit(scale_c)) {
518                                             (void) fprintf(stderr,
519                                                 gettext("invalid block size"
520                                                 " \'%s\'\n"), optarg);
521                                             exit(1);
522                                     }
523                                     /*
524                                      * make optarg[scale_len - 1] point to
525                                      * the scale factor
526                                      */
527                                     --scale_len;
528                             }

530                             switch (scale_c) {
531                             case 'y':
532                             case 'Y':
533                                     scale *= kilo;
534                                     /*FALLTHROUGH*/
535                             case 'Z':
536                             case 'z':
537                                     scale *= kilo;
538                                     /*FALLTHROUGH*/
539                             case 'E':
540                             case 'e':
541                                     scale *= kilo;
```



```
542                                     /*FALLTHROUGH*/
543                             case 'P':
544                             case 'p':
545                                     scale *= kilo;
546                                     /*FALLTHROUGH*/
547                             case 'T':
548                             case 't':
549                                     scale *= kilo;
550                                     /*FALLTHROUGH*/
551                             case 'G':
552                             case 'g':
553                                     scale *= kilo;
554                                     /*FALLTHROUGH*/
555                             case 'M':
556                             case 'm':
557                                     scale *= kilo;
558                                     /*FALLTHROUGH*/
559                             case 'K':
560                             case 'k':
561                                     scale *= kilo;
562                                     break;
563                             default:
564                                     if (!isdigit(scale_c)) {
565                                             (void) fprintf(stderr,
566                                                 gettext("invalid character "
567                                                 "following block size in "
568                                                 "\'%s\'\n"), optarg);
569                                             exit(1);
570                                     }
571                             }

573                             /* NULL out scale constant if present */
574                             if (scale > 1 && !isdigit(scale_c))
575                                     optarg[scale_len - 1] = '\0';

577                             /* Based on testing, this is what GNU ls does */
578                             block_size = strtoll(optarg, NULL, 0) * scale;
579                             if (block_size < 1) {
580                                     (void) fprintf(stderr,
581                                         gettext("invalid block size "
582                                         "\'%s\'\n"), optarg);
583                                     exit(1);
584                             }
585                             continue;
586                     }

588                     if (strcmp(long_options[option_index].name,
589                         "file-type") == 0) {
590                             file_typeflg++;
591                             Fflg++;
592                             statreq++;
593                             continue;
594                     }


597                     if (strcmp(long_options[option_index].name,
598                         "full-time") == 0) {
599                             Eflg++;
600                             statreq++;
601                             eflg = 0;
602                             time_fmt_old = FORMAT_ISO_FULL;
603                             time_fmt_new = FORMAT_ISO_FULL;
604                             continue;
605                     }

607                     if (strcmp(long_options[option_index].name,
```

```
608                             "time-style") == 0) {
609                                 /* like -E, but doesn't imply -l */
610                                 if (strcmp(optarg, "full-iso") == 0) {
611                                         Eflg++;
612                                         statreq++;
613                                         eflg = 0;
614                                         time_fmt_old = FORMAT_ISO_FULL;
615                                         time_fmt_new = FORMAT_ISO_FULL;
616                                         continue;
617                                 }
618                                 if (strcmp(optarg, "long-iso") == 0) {
619                                         statreq++;
620                                         Eflg = 0;
621                                         eflg = 0;
622                                         time_fmt_old = FORMAT_ISO_LONG;
623                                         time_fmt_new = FORMAT_ISO_LONG;
624                                         continue;
625                                 }
626                                 if (strcmp(optarg, "iso") == 0) {
627                                         statreq++;
628                                         Eflg = 0;
629                                         eflg = 0;
630                                         time_fmt_old = FORMAT_ISO_OLD;
631                                         time_fmt_new = FORMAT_ISO_NEW;
632                                         continue;
633                                 }
634                                 /* should be the default */
635                                 if (strcmp(optarg, "locale") == 0) {
636                                         time_fmt_old = FORMAT_OLD;
637                                         time_fmt_new = FORMAT_NEW;
638                                         continue;
639                                 }
640                                 if (optarg[0] == '+') {
641                                         char    *tnew;
637                                         char    *told, *tnew;
642                                         char    *p;
643                                         size_t  timelen = strlen(optarg);

645                                         p = strchr(optarg, '\n');
646                                         if (p != NULL)
647                                                 *p++ = '\0';

649                                         /*
650                                          * Time format requires a leading and
651                                          * trailing space
652                                          * Add room for 3 spaces + 2 nulls
653                                          * The + in optarg is replaced with
654                                          * a space.
655                                          */
656                                         timelen += 2 + 3;
657                                         told = realloc(told, timelen);
653                                         told = malloc(timelen);
658                                         if (told == NULL) {
659                                                 perror("ls");
660                                                 exit(2);
661                                         }

663                                         (void) memset(told, 0, timelen);
664                                         told[0] = ' ';
665                                         (void) strlcat(told, &optarg[1],
666                                             timelen);
667                                         (void) strlcat(told, " ", timelen);

669                                         if (p != NULL) {
670                                                 size_t tnew_len;
671                                                 size_t told_len =strlen(told);
```



```
673                                                 tnew = told + told_len + 1;
668                                                 tnew = told + strlen(told) + 1;
674                                                 tnew_len = timelen -
675                                                     told_len - 1;
670                                                     strlen(told) - 1;

677                                                 tnew[0] = ' ';
678                                                 (void) strlcat(tnew, p,
679                                                     tnew_len);
680                                                 (void) strlcat(tnew, " ",
681                                                     tnew_len);
682                                                 time_fmt_new =
683                                                     (const char *)tnew;
684                                         } else {
685                                                 time_fmt_new =
686                                                     (const char *)told;
687                                         }

689                                         time_fmt_old = (const char *)told;
690                                         time_custom = 1;
691                                         continue;
692                                 }
693                                 continue;
694                         }

696                         continue;

698                 case 'a':
699                         aflg++;
700                         continue;
701                 case 'A':
702                         Aflg++;
703                         continue;
704                 case 'b':
705                         bflg = 1;
706                         qflg = 0;
707                         continue;
708                 case 'B':
709                         Bflg = 1;
710                         continue;
711                 case 'c':
712                         uflg = 0;
713                         atm = 0;
714                         ctm = 0;
715                         mtm = 0;
716                         crtm = 0;
717                         cflg++;
718                         continue;
719                 case 'C':
720                         Cflg = 1;
721                         mflg = 0;
722 #ifdef XPG4
723                         lflg = 0;
724 #endif
725                         continue;
726                 case 'd':
727                         dflg++;
728                         continue;
729                 case 'e':
730                         eflg++;
731                         lflg++;
732                         statreq++;
733                         Eflg = 0;
734                         time_fmt_old = FORMAT_LONG;
735                         time_fmt_new = FORMAT_LONG;
```

```
736                     continue;
737             case 'E':
738                     Eflg++;
739                     lflg++;
740                     statreq++;
741                     eflg = 0;
742                     time_fmt_old = FORMAT_ISO_FULL;
743                     time_fmt_new = FORMAT_ISO_FULL;
744                     continue;
745             case 'f':
746                     fflg++;
747                     continue;
748             case 'F':
749                     Fflg++;
750                     statreq++;
751                     continue;
752             case 'g':
753                     gflg++;
754                     lflg++;
755                     statreq++;
756                     continue;
757             case 'h':
758                     hflg++;
759                     hscale = 1024;
760                     continue;
761             case 'H':
762                     Hflg++;
763                     /* -H and -L are mutually exclusive */
764                     Lflg = 0;
765                     continue;
766             case 'i':
767                     iflg++;
768                     continue;
769             case 'k':
770                     block_size = 1024;
771                     continue;
772             case 'l':
773                     lflg++;
774                     statreq++;
775                     Cflg = 0;
776                     xflg = 0;
777                     mflg = 0;
778                     atflg = 0;
779                     continue;
780             case 'L':
781                     Lflg++;
782                     /* -H and -L are mutually exclusive */
783                     Hflg = 0;
784                     continue;
785             case 'm':
786                     Cflg = 0;
787                     mflg = 1;
788 #ifdef XPG4
789                     lflg = 0;
790 #endif
791                     continue;
792             case 'n':
793                     nflg++;
794                     lflg++;
795                     statreq++;
796                     Cflg = 0;
797                     xflg = 0;
798                     mflg = 0;
799                     atflg = 0;
800                     continue;
801             case 'o':
```



```
802                     oflg++;
803                     lflg++;
804                     statreq++;
805                     continue;
806             case 'p':
807                     pflg++;
808                     statreq++;
809                     continue;
810             case 'q':
811                     qflg = 1;
812                     bflg = 0;
813                     continue;
814             case 'r':
815                     rflg = -1;
816                     continue;
817             case 'R':
818                     Rflg++;
819                     statreq++;
820                     continue;
821             case 's':
822                     sflg++;
823                     statreq++;
824                     continue;
825             case 'S':
826                     tflg = 0;
827                     Uflg = 0;
828                     Sflg++;
829                     statreq++;
830                     continue;
831             case 't':
832                     Sflg = 0;
833                     Uflg = 0;
834                     tflg++;
835                     statreq++;
836                     continue;
837             case 'U':
838                     Sflg = 0;
839                     tflg = 0;
840                     Uflg++;
841                     continue;
842             case 'u':
843                     cflg = 0;
844                     atm = 0;
845                     ctm = 0;
846                     mtm = 0;
847                     crtm = 0;
848                     uflg++;
849                     continue;
850             case 'V':
851                     Vflg++;
852                     /*FALLTHROUGH*/
853             case 'v':
854                     vflg++;
855 #if !defined(XPG4)
856                     if (lflg)
857                             continue;
858 #endif
859                     lflg++;
860                     statreq++;
861                     Cflg = 0;
862                     xflg = 0;
863                     mflg = 0;
864                     continue;
865             case 'w':
866                     wflg++;
867                     num_cols = atoi(optarg);
```

```
868                         continue;
869                 case 'x':
870                         xflg = 1;
871                         Cflg = 1;
872                         mflg = 0;
873 #ifdef XPG4
874                         lflg = 0;
875 #endif
876                         continue;
877                 case '1':
878                         Cflg = 0;
879                         continue;
880                 case '@':
881 #if !defined(XPG4)
882                         /*
883                          * -l has precedence over -@
884                          */
885                         if (lflg)
886                                 continue;
887 #endif
888                         atflg++;
889                         lflg++;
890                         statreq++;
891                         Cflg = 0;
892                         xflg = 0;
893                         mflg = 0;
894                         continue;
895                 case '/':
896                         saflg++;
897                         if (optarg != NULL) {
898                                 if (strcmp(optarg, "c") == 0) {
899                                         copt++;
900                                         vopt = 0;
901                                 } else if (strcmp(optarg, "v") == 0) {
902                                         vopt++;
903                                         copt = 0;
904                                 } else
905                                         opterr++;
906                         } else
907                                 opterr++;
908                         lflg++;
909                         statreq++;
910                         Cflg = 0;
911                         xflg = 0;
912                         mflg = 0;
913                         continue;
914                 case '%':
915                         tmflg++;
916                         if (optarg != NULL) {
917                                 if (strcmp(optarg, "ctime") == 0) {
918                                         ctm++;
919                                         atm = 0;
920                                         mtm = 0;
921                                         crtm = 0;
922                                 } else if (strcmp(optarg, "atime") == 0) {
923                                         atm++;
924                                         ctm = 0;
925                                         mtm = 0;
926                                         crtm = 0;
927                                         uflg = 0;
928                                         cflg = 0;
929                                 } else if (strcmp(optarg, "mtime") == 0) {
930                                         mtm++;
931                                         atm = 0;
932                                         ctm = 0;
933                                         crtm = 0;
```



```
934                                         uflg = 0;
935                                         cflg = 0;
936                                 } else if (strcmp(optarg, "crtime") == 0) {
937                                         crtm++;
938                                         atm = 0;
939                                         ctm = 0;
940                                         mtm = 0;
941                                         uflg = 0;
942                                         cflg = 0;
943                                 } else if (strcmp(optarg, "all") == 0) {
944                                         alltm++;
945                                         atm = 0;
946                                         ctm = 0;
947                                         mtm = 0;
948                                         crtm = 0;
949                                 } else
950                                         opterr++;
951                         } else
952                                 opterr++;

954                         Sflg = 0;
955                         statreq++;
956                         mflg = 0;
957                         continue;
958                 case '?':
959                         opterr++;
960                         continue;
961                 }

963         if (opterr) {
964                 (void) fprintf(stderr, gettext(
965                     "usage: ls -aAbBcCdeEfFghHiklLmnopqrRsStuUwxvV1@/%[c | v]"
966                     "%%[atime | crtime | ctime | mtime | all]"
967                     " [files]\n"));
968                 exit(2);
969         }

971         if (fflg) {
972                 aflg++;
973                 lflg = 0;
974                 sflg = 0;
975                 tflg = 0;
976                 Sflg = 0;
977                 statreq = 0;
978         }

980         fixedwidth = 2;
981         if (pflg || Fflg)
982                 fixedwidth++;
983         if (iflg)
984                 fixedwidth += 11;
985         if (sflg)
986                 fixedwidth += 5;

988         if (lflg) {
989                 if (!gflg && !oflg)
990                         gflg = oflg = 1;
991                 else
992                 if (gflg && oflg)
993                         gflg = oflg = 0;
994                 Cflg = mflg = 0;
995         }

997         if (!wflg && (Cflg || mflg)) {
998                 char *clptr;
999                 if ((clptr = getenv("COLUMNS")) != NULL)
```

```
1000                        num_cols = atoi(clptr);
1001 #ifdef TERMINFO
1002                else {
1003                        if (ioctl(1, TIOCGWINSZ, &win) != -1)
1004                                num_cols = (win.ws_col == 0 ? 80 : win.ws_col);
1005                }
1006 #endif
1007        }

1009        /*
1010         * When certain options (-f, or -U and -1, and not -l, etc.) are
1011         * specified, don't cache each dirent as it's read.  This 'noflist'
1012         * option is set when there's no need to cache those dirents; instead,
1013         * print them out as they're read.
1014         */
1015        if ((Uflg || fflg) && !Cflg && !lflg && !iflg && statreq == 0)
1016                noflist = 1;

1018        if (num_cols < 20 || num_cols > 1000)
1019                /* assume it is an error */
1020                num_cols = 80;

1022        /* allocate space for flist and the associated  */
1023        /* data structures (lbufs)                      */
1024        maxfils = quantn;
1025        if (((flist = malloc(maxfils * sizeof (struct lbuf *))) == NULL) ||
1026            ((nxtlbf = malloc(quantn * sizeof (struct lbuf))) == NULL)) {
1027                perror("ls");
1028                exit(2);
1029        }
1030        if ((hlbf = malloc(sizeof(*hlbf))) == NULL) {
1031                perror("ls");
1032                exit(2);
1033        }
1034        hlbf[0] = nxtlbf;
1035        if ((amino = (argc-optind)) == 0) {
1036                                        /*
1037                                         * case when no names are given
1038                                         * in ls-command and current
1039                                         * directory is to be used
1040                                         */
1041                argv[optind] = dotp;
1042        }

1044        if (colorflg)
1045                ls_color_init();

1047        for (i = 0; i < (amino ? amino : 1); i++) {

1049                /*
1050                 * If we are recursing, we need to make sure we don't
1051                 * get into an endless loop.  To keep track of the inodes
1052                 * (actually, just the directories) visited, we
1053                 * maintain a directory ancestry list for a file
1054                 * hierarchy.  As we go deeper into the hierarchy,
1055                 * a parent directory passes its directory list
1056                 * info (device id, inode number, and a pointer to
1057                 * its parent) to each of its children.  As we
1058                 * process a child that is a directory, we save
1059                 * its own personal directory list info.  We then
1060                 * check to see if the child has already been
1061                 * processed by comparing its device id and inode
1062                 * number from its own personal directory list info
1063                 * to that of each of its ancestors.  If there is a
1064                 * match, then we know we've detected a cycle.
1065                 */
```



```
1066                if (Rflg) {
1067                        /*
1068                         * This is the first parent in this lineage
1069                         * (first in a directory hierarchy), so
1070                         * this parent's parent doesn't exist.  We
1071                         * only initialize myinfo when we are
1072                         * recursing, otherwise it's not used.
1073                         */
1074                        if ((myinfo = (struct ditem *)malloc(
1075                            sizeof (struct ditem))) == NULL) {
1076                                perror("ls");
1077                                exit(2);
1078                        } else {
1079                                myinfo->dev = 0;
1080                                myinfo->ino = 0;
1081                                myinfo->parent = NULL;
1082                        }
1083                }

1085                if (Cflg || mflg) {
1086                        width = strcol((unsigned char *)argv[optind]);
1087                        if (width > filewidth)
1088                                filewidth = width;
1089                }
1090                if ((ep = gstat((*argv[optind] ? argv[optind] : dotp),
1091                    1, myinfo)) == NULL) {
1092                        if (nomocore)
1093                                exit(2);
1094                        err = 2;
1095                        optind++;
1096                        continue;
1097                }
1098                ep->ln.namep = (*argv[optind] ? argv[optind] : dotp);
1099                ep->lflags |= ISARG;
1100                optind++;
1101                nargs++;        /* count good arguments stored in flist */
1102                if (acl_err)
1103                        err = 2;
1104        }
1105        colwidth = fixedwidth + filewidth;
1106        if (!Uflg)
1107                qsort(flist, (unsigned)nargs, sizeof (struct lbuf *),
1108                    (int (*)(const void *, const void *))compar);
1109        for (i = 0; i < nargs; i++) {
1110                if ((flist[i]->ltype == 'd' && dflg == 0) || fflg)
1111                        break;
1112        }

1114        pem(&flist[0], &flist[i], 0);
1115        for (; i < nargs; i++) {
1116                pdirectory(flist[i]->ln.namep, Rflg ||
1117                    (amino > 1), nargs, 0, flist[i]->ancinfo);
1118                if (nomocore)
1119                        exit(2);
1120                /* -R: print subdirectories found */
1121                while (dfirst || cdfirst) {
1122                        /* Place direct subdirs on front in right order */
1123                        while (cdfirst) {
1124                                /* reverse cdfirst onto front of dfirst */
1125                                dtemp = cdfirst;
1126                                cdfirst = cdfirst -> dc_next;
1127                                dtemp -> dc_next = dfirst;
1128                                dfirst = dtemp;
1129                        }
1130                        /* take off first dir on dfirst & print it */
1131                        dtemp = dfirst;
```

```
1132                        dfirst = dfirst->dc_next;
1133                        pdirectory(dtemp->dc_name, 1, nargs,
1134                            dtemp->cycle_detected, dtemp->myancinfo);
1135                        if (nomocore)
1136                                exit(2);
1137                        free(dtemp->dc_name);
1138                        free(dtemp);
1139                }
1140        }

1142        for (i = 0; i < hlbfsz; i ++)
1143                free(hlbf[i]);

1145        free(told);
1146        free(hlbf);
1147        free(flist);
1148        freecachenodes();

1150        return (err);
1151 }
_____unchanged_portion_omitted_

1778 /*
1779  * get status of file and recomputes tblocks;
1780  * argfl = 1 if file is a name in ls-command and = 0
1781  * for filename in a directory whose name is an
1782  * argument in the command;
1783  * stores a pointer in flist[nfiles] and
1784  * returns that pointer;
1785  * returns NULL if failed;
1786  */
1787 static struct lbuf *
1788 gstat(char *file, int argfl, struct ditem *myparent)
1789 {
1790        struct stat statb, statb1;
1791        struct lbuf *rep;
1792        char buf[BUFSIZ];
1793        ssize_t cc;
1794        int (*statf)() = ((Lflg) || (Hflg && argfl)) ? stat : lstat;
1795        int aclcnt;
1796        int error;
1797        aclent_t *tp;
1798        o_mode_t groupperm, mask;
1799        int grouppermfound, maskfound;

1801        if (nomocore)
1802                return (NULL);

1804        if (nfiles >= maxfils) {
1805                /*
1806                 * all flist/lbuf pair assigned files, time to get some
1807                 * more space
1808                 */
1809                maxfils += quantn;
1810                if (((flist = realloc(flist,
1811                    maxfils * sizeof (struct lbuf *))) == NULL) ||
1812                    ((nxtlbf = malloc(quantn *
1813                    sizeof (struct lbuf))) == NULL)) {
1814                        perror("ls");
1815                        nomocore = 1;
1816                        return (NULL);
1817                }
1818                if ((hlbf = realloc(hlbf, sizeof(*hlbf) * (hlbfsz + 1))) == NULL
1819                        perror("ls");
1820                        nomocore = 1;
1821                        return (NULL);
```



```
1822                }
1823                hlbf[hlbfsz++] = nxtlbf;
1824        }

1826        /*
1827         * nfiles is reset to nargs for each directory
1828         * that is given as an argument maxn is checked
1829         * to prevent the assignment of an lbuf to a flist entry
1830         * that already has one assigned.
1831         */
1832        if (nfiles >= maxn) {
1833                rep = nxtlbf++;
1834                flist[nfiles++] = rep;
1835                maxn = nfiles;
1836        } else {
1837                rep = flist[nfiles++];
1838        }

1840        /* Clear the lbuf */
1841        (void) memset((void *) rep, 0, sizeof (struct lbuf));

1843        /*
1844         * When noflist is set, none of the extra information about the dirent
1845         * will be printed, so omit remaining initialization of this lbuf
1846         * as well as the  stat(2) call.
1847         */
1848        if (!argfl && noflist)
1849                return (rep);

1851        /* Initialize non-zero members */

1853        rep->lat.tv_sec = time(NULL);
1854        rep->lct.tv_sec = time(NULL);
1855        rep->lmt.tv_sec = time(NULL);

1857        if (argfl || statreq) {
1858                int doacl;

1860                if (lflg)
1861                        doacl = 1;
1862                else
1863                        doacl = 0;

1865                if ((*statf)(file, &statb) < 0) {
1866                        if (argfl || errno != ENOENT ||
1867                            (Lflg && lstat(file, &statb) == 0)) {
1868                                /*
1869                                 * Avoid race between readdir and lstat.
1870                                 * Print error message in case of dangling link.
1871                                 */
1872                                perror(file);
1873                                err = 2;
1874                        }
1875                        nfiles--;
1876                        return (NULL);
1877                }

1879                /*
1880                 * If -H was specified, and the file linked to was
1881                 * not a directory, then we need to get the info
1882                 * for the symlink itself.
1883                 */
1884                if ((Hflg) && (argfl) &&
1885                    ((statb.st_mode & S_IFMT) != S_IFDIR)) {
1886                        if (lstat(file, &statb) < 0) {
1887                                perror(file);
```

```
1888                                    err = 2;
1889                            }
1890                    }

1892                    rep->lnum = statb.st_ino;
1893                    rep->lsize = statb.st_size;
1894                    rep->lblocks = statb.st_blocks;
1895                    if (colorflg)
1896                            rep->color = ls_color_find(file, statb.st_mode);

1898                    switch (statb.st_mode & S_IFMT) {
1899                    case S_IFDIR:
1900                            rep->ltype = 'd';
1901                            if (Rflg) {
1902                                    record_ancestry(file, &statb, rep,
1903                                        argfl, myparent);
1904                            }
1905                            break;
1906                    case S_IFBLK:
1907                            rep->ltype = 'b';
1908                            rep->lsize = (off_t)statb.st_rdev;
1909                            break;
1910                    case S_IFCHR:
1911                            rep->ltype = 'c';
1912                            rep->lsize = (off_t)statb.st_rdev;
1913                            break;
1914                    case S_IFIFO:
1915                            rep->ltype = 'p';
1916                            break;
1917                    case S_IFSOCK:
1918                            rep->ltype = 's';
1919                            rep->lsize = 0;
1920                            break;
1921                    case S_IFLNK:
1922                            /* symbolic links may not have ACLs, so elide acl() */
1923                            if ((Lflg == 0) || (Hflg == 0) ||
1924                                ((Hflg) && (!argfl))) {
1925                                    doacl = 0;
1926                            }
1927                            rep->ltype = 'l';
1928                            if (lflg || colorflg) {
1929                                    cc = readlink(file, buf, BUFSIZ);
1930                                    if (cc < 0)
1931                                            break;

1933                                    /*
1934                                     * follow the symbolic link
1935                                     * to generate the appropriate
1936                                     * Fflg marker for the object
1937                                     * eg, /bin -> /sym/bin/
1938                                     */
1939                                    error = 0;
1940                                    if (Fflg || pflg || colorflg)
1941                                            error = stat(file, &statb1);

1943                                    if (colorflg) {
1944                                            if (error >= 0)
1945                                                    rep->link_color =
1946                                                        ls_color_find(file,
1947                                                        statb1.st_mode);
1948                                            else
1949                                                    rep->link_color =
1950                                                        lsc_orphan;
1951                                    }

1953                                    if ((Fflg || pflg) && error >= 0) {
```



```
1954                                            switch (statb1.st_mode & S_IFMT) {
1955                                            case S_IFDIR:
1956                                                    buf[cc++] = '/';
1957                                                    break;
1958                                            case S_IFSOCK:
1959                                                    buf[cc++] = '=';
1960                                                    break;
1961                                            case S_IFDOOR:
1962                                                    buf[cc++] = '>';
1963                                                    break;
1964                                            case S_IFIFO:
1965                                                    buf[cc++] = '|';
1966                                                    break;
1967                                            default:
1968                                                    if ((statb1.st_mode & ~S_IFMT) &
1969                                                        (S_IXUSR|S_IXGRP| S_IXOTH))
1970                                                            buf[cc++] = '*';
1971                                                    break;
1972                                            }
1973                                    }
1974                                    buf[cc] = '\0';
1975                                    rep->flinkto = strdup(buf);
1976                                    if (rep->flinkto == NULL) {
1977                                            perror("ls");
1978                                            nomocore = 1;
1979                                            return (NULL);
1980                                    }
1981                                    break;
1982                            }

1984                            /*
1985                             * ls /sym behaves differently from ls /sym/
1986                             * when /sym is a symbolic link. This is fixed
1987                             * when explicit arguments are specified.
1988                             */

1990 #ifdef XPG6
1991                            /* Do not follow a symlink when -F is specified */
1992                            if ((!argfl) || (argfl && Fflg) ||
1993                                (stat(file, &statb1) < 0))
1994 #else
1995                            /* Follow a symlink when -F is specified */
1996                            if (!argfl || stat(file, &statb1) < 0)
1997 #endif /* XPG6 */
1998                                    break;
1999                            if ((statb1.st_mode & S_IFMT) == S_IFDIR) {
2000                                    statb = statb1;
2001                                    rep->ltype = 'd';
2002                                    rep->lsize = statb1.st_size;
2003                                    if (Rflg) {
2004                                            record_ancestry(file, &statb, rep,
2005                                                argfl, myparent);
2006                                    }
2007                            }
2008                            break;
2009                    case S_IFDOOR:
2010                            rep->ltype = 'D';
2011                            break;
2012                    case S_IFREG:
2013                            rep->ltype = '-';
2014                            break;
2015                    case S_IFPORT:
2016                            rep->ltype = 'P';
2017                            break;
2018                    default:
2019                            rep->ltype = '?';
```

```
2020                    break;
2021            }
2022            rep->lflags = statb.st_mode & ~S_IFMT;

2024            if (!S_ISREG(statb.st_mode))
2025                    rep->lflags |= LS_NOTREG;

2027            rep->luid = statb.st_uid;
2028            rep->lgid = statb.st_gid;
2029            rep->lnl = statb.st_nlink;
2030            if (uflg || (tmflg && atm))
2031                    rep->lmtime = statb.st_atim;
2032            else if (cflg || (tmflg && ctm))
2033                    rep->lmtime = statb.st_ctim;
2034            else
2035                    rep->lmtime = statb.st_mtim;
2036            rep->lat = statb.st_atim;
2037            rep->lct = statb.st_ctim;
2038            rep->lmt = statb.st_mtim;

2040            /* ACL: check acl entries count */
2041            if (doacl) {

2043                    error = acl_get(file, 0, &rep->aclp);
2044                    if (error) {
2045                            (void) fprintf(stderr,
2046                                gettext("ls: can't read ACL on %s: %s\n"),
2047                                file, acl_strerror(error));
2048                            rep->acl = ' ';
2049                            acl_err++;
2050                            return (rep);
2051                    }

2053                    rep->acl = ' ';

2055                    if (rep->aclp &&
2056                        ((acl_flags(rep->aclp) & ACL_IS_TRIVIAL) == 0)) {
2057                            rep->acl = '+';
2058                            /*
2059                             * Special handling for ufs aka aclent_t ACL's
2060                             */
2061                            if (acl_type(rep->aclp) == ACLENT_T) {
2062                                    /*
2063                                     * For files with non-trivial acls, the
2064                                     * effective group permissions are the
2065                                     * intersection of the GROUP_OBJ value
2066                                     * and the CLASS_OBJ (acl mask) value.
2067                                     * Determine both the GROUP_OBJ and
2068                                     * CLASS_OBJ for this file and insert
2069                                     * the logical AND of those two values
2070                                     * in the group permissions field
2071                                     * of the lflags value for this file.
2072                                     */

2074                                    /*
2075                                     * Until found in acl list, assume
2076                                     * maximum permissions for both group
2077                                     * a nd mask.  (Just in case the acl
2078                                     * lacks either value for some reason.)
2079                                     */
2080                                    groupperm = 07;
2081                                    mask = 07;
2082                                    grouppermfound = 0;
2083                                    maskfound = 0;
2084                                    aclcnt = acl_cnt(rep->aclp);
2085                                    for (tp =
```



```
2086                                        (aclent_t *)acl_data(rep->aclp);
2087                                        aclcnt--; tp++) {
2088                                            if (tp->a_type == GROUP_OBJ) {
2089                                                    groupperm = tp->a_perm;
2090                                                    grouppermfound = 1;
2091                                                    continue;
2092                                            }
2093                                            if (tp->a_type == CLASS_OBJ) {
2094                                                    mask = tp->a_perm;
2095                                                    maskfound = 1;
2096                                            }
2097                                            if (grouppermfound && maskfound)
2098                                                    break;
2099                                    }


2102                                    /* reset all the group bits */
2103                                    rep->lflags &= ~S_IRWXG;

2105                                    /*
2106                                     * Now set them to the logical AND of
2107                                     * the GROUP_OBJ permissions and the
2108                                     * acl mask.
2109                                     */

2111                                    rep->lflags |= (groupperm & mask) << 3;

2113                            } else if (acl_type(rep->aclp) == ACE_T) {
2114                                    int mode;
2115                                    mode = grp_mask_to_mode(rep);
2116                                    rep->lflags &= ~S_IRWXG;
2117                                    rep->lflags |= mode;
2118                            }
2119                    }

2121                    if (!vflg && !Vflg && rep->aclp) {
2122                            acl_free(rep->aclp);
2123                            rep->aclp = NULL;
2124                    }

2126                    if (atflg && pathconf(file, _PC_XATTR_EXISTS) == 1)
2127                            rep->acl = '@';

2129            } else
2130                    rep->acl = ' ';

2132            /* mask ISARG and other file-type bits */

2134            if (rep->ltype != 'b' && rep->ltype != 'c')
2135                    tblocks += rep->lblocks;

2137            /* Get extended system attributes */

2139            if ((saflg || (tmflg && crtm) || (tmflg && alltm)) &&
2140                (sysattr_support(file, _PC_SATTR_EXISTS) == 1)) {
2141                    int i;

2143                    sacnt = attr_count();
2144                    /*
2145                     * Allocate 'sacnt' size array to hold extended
2146                     * system attribute name (verbose) or respective
2147                     * symbol represenation (compact).
2148                     */
2149                    rep->exttr = xmalloc(sacnt * sizeof (struct attrb),
2150                        rep);
```

```
2152                        /* initialize boolean attribute list */
2153                        for (i = 0; i < sacnt; i++)
2154                                rep->exttr[i].name = NULL;
2155                        if (get_sysxattr(file, rep) != 0) {
2156                                (void) fprintf(stderr,
2157                                    gettext("ls:Failed to retrieve "
2158                                    "extended system attribute from "
2159                                    "%s\n"), file);
2160                                rep->exttr[0].name = xmalloc(2, rep);
2161                                (void) strlcpy(rep->exttr[0].name, "?", 2);
2162                        }
2163                }
2164        }
2165        return (rep);
2166 }
```

_____*unchanged_portion_omitted_*

```
2248 void
2249 freecachenode(struct cachenode *node)
2250 {
2251         struct cachenode *current = node;
2252         if (current != NULL) {
2253                 struct cachenode *grt = NULL;
2254                 struct cachenode *lss = NULL;

2256                 if (current->grtrchild != NULL) {
2257                         grt = current->grtrchild;
2258                         freecachenode(grt);
2259                 }
2260                 if (current->lesschild != NULL) {
2261                         lss = current->lesschild;
2262                         freecachenode(lss);
2263                 }

2265                 free(current);
2266                 current = NULL;
2267         }
2268 }

2270 void
2271 freecachenodes(void)
2272 {
2273         freecachenode(groups);
2274         freecachenode(names);
2275 }


2278 /*
2279  * get name from cache, or passwd file for a given uid;
2280  * lastuid is set to uid.
2281  */
2282 static char *
2283 getname(uid_t uid)
2284 {
2285         struct passwd *pwent;
2286         struct cachenode *c;

2288         if ((uid == lastuid) && lastuname)
2289                 return (lastuname);

2291         c = findincache(&names, uid);
2292         if (c->initted == 0) {
2293                 if ((pwent = getpwuid(uid)) != NULL) {
2294                         SCPYN(&c->name[0], pwent->pw_name);
2295                 } else {
2296                         (void) sprintf(&c->name[0], "%-8u", (int)uid);
```



```
2297                 }
2298                 c->initted = 1;
2299         }
2300         lastuid = uid;
2301         lastuname = &c->name[0];
2302         return (lastuname);
2303 }
```

_____*unchanged_portion_omitted_*

```
2669 /* Set extended system attribute timestamp display */

2671 void
2672 set_sysattrtm_display(char *name, struct lbuf *rep)
2673 {
2674         uint_t          nelem;
2675         uint64_t        *value;
2676         int             i;
2677         size_t          len;

2679         if (nvpair_value_uint64_array(pair, &value, &nelem) == 0) {
2680                 if (value != NULL) {
2626                 if (*value != NULL) {
2681                         len = strlen(name);
2682                         i = 0;
2683                         while (rep->extm[i].stm != 0 && i < sacnt)
2684                                 i++;
2685                         rep->extm[i].stm = value[0];
2686                         rep->extm[i].nstm = value[1];
2687                         rep->extm[i].name = xmalloc(len + 1, rep);
2688                         (void) strlcpy(rep->extm[i].name, name, len + 1);
2689                 }
2690         }
2691 }
```

_____*unchanged_portion_omitted_*