```
**********************************************************
   67549 Thu Feb 28 11:26:00 2019
new/usr/src/cmd/auditconfig/auditconfig.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * 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) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright (c) 2019, Joyent, Inc.
 */

/*
 * auditconfig - set and display audit parameters
 */

#include <locale.h>
#include <sys/types.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdarg.h>
#include <unistd.h>
#include <errno.h>
#include <sys/param.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <nlist.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/mkdev.h>
#include <sys/param.h>
#include <pwd.h>
#include <libintl.h>
#include <zone.h>
#include <libscf_priv.h>
#include <tsol/label.h>
#include <bsm/libbsm.h>
#include <audit_policy.h>
#include <audit_scf.h>
```



```
enum    commands {
        AC_ARG_ACONF,
        AC_ARG_AUDIT,
        AC_ARG_CHKACONF,
        AC_ARG_CHKCONF,
        AC_ARG_CONF,
        AC_ARG_GETASID,
        AC_ARG_GETAUDIT,
        AC_ARG_GETAUID,
        AC_ARG_GETCAR,
        AC_ARG_GETCLASS,
        AC_ARG_GETCOND,
        AC_ARG_GETCWD,
        AC_ARG_GETESTATE,
        AC_ARG_GETFLAGS,
        AC_ARG_GETKAUDIT,
        AC_ARG_GETKMASK,
        AC_ARG_GETNAFLAGS,
        AC_ARG_GETPINFO,
        AC_ARG_GETPLUGIN,
        AC_ARG_GETPOLICY,
        AC_ARG_GETQBUFSZ,
        AC_ARG_GETQCTRL,
        AC_ARG_GETQDELAY,
        AC_ARG_GETQHIWATER,
        AC_ARG_GETQLOWATER,
        AC_ARG_GETSTAT,
        AC_ARG_GETTERMID,
        AC_ARG_LSEVENT,
        AC_ARG_LSPOLICY,
        AC_ARG_SETASID,
        AC_ARG_SETAUDIT,
        AC_ARG_SETAUID,
        AC_ARG_SETCLASS,
        AC_ARG_SETFLAGS,
        AC_ARG_SETKAUDIT,
        AC_ARG_SETKMASK,
        AC_ARG_SETNAFLAGS,
        AC_ARG_SETPLUGIN,
        AC_ARG_SETPMASK,
        AC_ARG_SETPOLICY,
        AC_ARG_SETQBUFSZ,
        AC_ARG_SETQCTRL,
        AC_ARG_SETQDELAY,
        AC_ARG_SETQHIWATER,
        AC_ARG_SETQLOWATER,
        AC_ARG_SETSMASK,
        AC_ARG_SETSTAT,
        AC_ARG_SETUMASK,
        AC_ARG_SET_TEMPORARY
};
```

_____*unchanged_portion_omitted_*

```
static int
strisnum(char *s)
{
        if (s == NULL || !*s)
                return (0);

        for (; *s == '-' || *s == '+'; s++) {
        for (; *s == '-' || *s == '+'; s++)

                if (!*s)
                        return (0);
```

```
2582         }

2584         for (; *s; s++) {
2580         for (; *s; s++)
2585                 if (!isdigit(*s))
2586                         return (0);
2587         }

2589         return (1);
2590 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   35175 Thu Feb 28 11:26:01 2019
new/usr/src/cmd/busstat/busstat.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg   Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
/*
 * 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 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <time.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/modctl.h>
#include <sys/systeminfo.h>
#include <limits.h>
#include <signal.h>
#include <fcntl.h>
#include <unistd.h>
#include <stropts.h>
#include <locale.h>
#include <libintl.h>
#include <libgen.h>
#include <nl_types.h>
#include <kstat.h>
#include <ctype.h>
#include <signal.h>
#include <errno.h>
#include <time.h>

#include "busstat.h"
```



```
/* Global defines */
static int              delta = TRUE;
static int              banner = TRUE;
static int              max_pic_num = 1;
static int              initial_read = TRUE;
static char             *pgmname;
static kstat_ctl_t      *kc;                    /* libkstat cookie */
static dev_node_t       *dev_list_head  = NULL;
static dev_node_t       *dev_list_tail  = NULL;

/*
 * Global flags.
 */
static char     curr_dev_name[KSTAT_STRLEN];
static int      curr_inst_num;

static void print_evt(void);
static void print_dev(int, char *);
static void parse_cmd(int);
static void parse_dev_inst(char *);
static void parse_pic_evt(char *);
static void add_dev_node(char *, int);
static void add_all_dev_node(char *);
static void add_evt_node(dev_node_t *);
static void modify_evt_node(dev_node_t *, char *);
static void prune_evt_nodes(dev_node_t *);
static void setup_evts(void);
static void set_evt(dev_node_t *);
static void read_evts(void);
static void read_r_evt_node(dev_node_t *, int, kstat_named_t *);
static void read_w_evt_node(dev_node_t *, int, kstat_named_t *);
static void check_dr_ops(void);
static void remove_dev_node(dev_node_t *);
static dev_node_t *find_dev_node(char *, int, int);
static kstat_t *find_pic_kstat(char *, int, char *);
static int64_t is_num(char *);
static void print_banner(void);
static void print_timestamp(void);
static void usage(void);
static void *safe_malloc(size_t);
static void set_timer(int);
static void handle_sig(int);
static int strisnum(const char *);

int
main(int argc, char **argv)
{
        int             c, i;
        int             interval = 1;   /* Interval between displays */
        int             count = 0;      /* Number of times to sample */
        int             write_evts = FALSE;
        int             pos = 0;

#if !defined(TEXT_DOMAIN)
#define TEXT_DOMAIN     "SYS_TEST"
#endif

        /* For I18N */
        (void) setlocale(LC_ALL, "");
        (void) textdomain(TEXT_DOMAIN);

        pgmname = basename(argv[0]);

        if ((kc = kstat_open()) == NULL) {
                (void) fprintf(stderr, gettext("%s: could not "
                        "open /dev/kstat\n"), pgmname);
```

```
125                 exit(1);
126         }

128         while ((c = getopt(argc, argv, "e:w:r:ahln")) != EOF) {
129                 switch (c) {
130                 case 'a':
131                         delta = FALSE;
132                         break;
133                 case 'e':
134                         (void) print_evt();
135                         break;
136                 case 'h':
137                         usage();
138                         break;
139                 case 'l':
140                         (void) print_dev(argc, argv[argc-1]);
141                         break;
142                 case 'n':
143                         banner = FALSE;
144                         break;
145                 case 'r':
146                         (void) parse_cmd(READ_EVT);
147                         break;
148                 case 'w':
149                         (void) parse_cmd(WRITE_EVT);
150                         write_evts = TRUE;
151                         break;
152                 default:
153                         (void) fprintf(stderr, gettext("%s: invalid "
154                                 "option\n"), pgmname);
155                         usage();
156                         break;
157                 }
158         }

160         if ((argc == 1) || (dev_list_head == NULL))
161                 usage();

163         /*
164          * validate remaining operands are numeric.
165          */
166         pos = optind;
167         while (pos < argc) {
168                 if (strisnum(argv[pos]) == 0) {
169                         (void) fprintf(stderr,
170                                 gettext("%s: syntax error\n"),
171                                 pgmname);
172                         usage();
173                 }
174                 pos++;
175         }

177         if (optind < argc) {
178                 if ((interval = atoi(argv[optind])) == 0) {
179                         (void) fprintf(stderr, gettext("%s: invalid "
180                                 "interval value\n"), pgmname);
181                         exit(1);
182                 }

184                 optind++;
185                 if (optind < argc)
186                         if ((count = atoi(argv[optind])) <= 0) {
187                                 (void) fprintf(stderr, gettext("%s: "
188                                         "invalid iteration value.\n"),
189                                             pgmname);
190                                 exit(1);
```



```
191                         }
192         }

194         set_timer(interval);

196         /*
197          * Set events for the first time.
198          */
199         if (write_evts == TRUE)
200                 setup_evts();


203         if (count > 0) {
204                 for (i = 0; i < count; i++) {
205                         if (banner)
206                                 print_banner();

208                         check_dr_ops();
209                         read_evts();
210                         (void) fflush(stdout);
211                         (void) pause();
212                 }
213         } else {
214                 for (;;) {
215                         if (banner)
216                                 print_banner();

218                         check_dr_ops();
219                         read_evts();
220                         (void) fflush(stdout);
221                         (void) pause();
222                 }
223         }

225         read_evts();
226         return (0);
227 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   13318 Thu Feb 28 11:26:01 2019
new/usr/src/cmd/cat/cat.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */


/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 *      Concatenate files.
 */

#include        <stdio.h>
#include        <stdlib.h>
#include        <ctype.h>
#include        <sys/types.h>
#include        <sys/stat.h>
#include        <locale.h>
#include        <unistd.h>
#include        <sys/mman.h>
#include        <errno.h>
#include        <string.h>

#include        <widec.h>
#include        <wctype.h>
#include        <limits.h>
#include        <libintl.h>
#define IDENTICAL(A, B) (A.st_dev == B.st_dev && A.st_ino == B.st_ino)

#define MAXMAPSIZE      (8*1024*1024)   /* map at most 8MB */
#define SMALLFILESIZE   (32*1024)       /* don't use mmap on little files */

static int vncat(FILE *);
static int cat(FILE *, struct stat *, struct stat *, char *);
```



```
static int      silent = 0;             /* s flag */
static int      visi_mode = 0;          /* v flag */
static int      visi_tab = 0;           /* t flag */
static int      visi_newline = 0;       /* e flag */
static int      bflg = 0;               /* b flag */
static int      nflg = 0;               /* n flag */
static long     ibsize;
static long     obsize;
static unsigned char    buf[SMALLFILESIZE];


int
main(int argc, char **argv)
{
        FILE *fi;
        int c;
        extern  int optind;
        int     errflg = 0;
        int     stdinflg = 0;
        int     status = 0;
        int     estatus = 0;
        struct stat source, target;

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

#ifdef STANDALONE
        /*
         * If the first argument is NULL,
         * discard arguments until we find cat.
         */
        if (argv[0][0] == '\0')
                argc = getargv("cat", &argv, 0);
#endif

        /*
         * Process the options for cat.
         */

        while ((c = getopt(argc, argv, "usvtebn")) != EOF) {
                switch (c) {

                case 'u':

                        /*
                         * If not standalone, set stdout to
                         * completely unbuffered I/O when
                         * the 'u' option is used.
                         */

#ifndef STANDALONE
                        setbuf(stdout, (char *)NULL);
#endif
                        continue;

                case 's':

                        /*
                         * The 's' option requests silent mode
                         * where no messages are written.
                         */
```

```
126                     silent++;
127                     continue;

129             case 'v':

131                     /*
132                      * The 'v' option requests that non-printing
133                      * characters (with the exception of newlines,
134                      * form-feeds, and tabs) be displayed visibly.
135                      *
136                      * Control characters are printed as "^x".
137                      * DEL characters are printed as "^?".
138                      * Non-printable  and non-contrlol characters with the
139                      * 8th bit set are printed as "M-x".
140                      */

142                     visi_mode++;
143                     continue;

145             case 't':

147                     /*
148                      * When in visi_mode, this option causes tabs
149                      * to be displayed as "^I".
150                      */

152                     visi_tab++;
153                     continue;

155             case 'e':

157                     /*
158                      * When in visi_mode, this option causes newlines
159                      * and form-feeds to be displayed as "$" at the end
160                      * of the line prior to the newline.
161                      */

163                     visi_newline++;
164                     continue;

166             case 'b':

168                     /*
169                      * Precede each line output with its line number,
170                      * but omit the line numbers from blank lines.
171                      */

173                     bflg++;
174                     nflg++;
175                     continue;

177             case 'n':

179                     /*
180                      * Precede each line output with its line number.
181                      */

183                     nflg++;
184                     continue;

186             case '?':
187                     errflg++;
188                     break;
189             }
190             break;
191     }
```



```
193     if (errflg) {
194             if (!silent)
195                     (void) fprintf(stderr,
196                         gettext("usage: cat [ -usvtebn ] [-|file] ...\n"));
197             exit(2);
198     }

200     /*
201      * Stat stdout to be sure it is defined.
202      */

204     if (fstat(fileno(stdout), &target) < 0) {
205             if (!silent)
206                     (void) fprintf(stderr,
207                         gettext("cat: Cannot stat stdout\n"));
208             exit(2);
209     }
210     obsize = target.st_blksize;

212     /*
213      * If no arguments given, then use stdin for input.
214      */

216     if (optind == argc) {
217             argc++;
218             stdinflg++;
219     }

221     /*
222      * Process each remaining argument,
223      * unless there is an error with stdout.
224      */


227     for (argv = &argv[optind];
228         optind < argc && !ferror(stdout); optind++, argv++) {

230             /*
231              * If the argument was '-' or there were no files
232              * specified, take the input from stdin.
233              */

235             if (stdinflg ||
236                 ((*argv)[0] == '-' && (*argv)[1] == '\0'))
237                     fi = stdin;
238             else {
239                     /*
240                      * Attempt to open each specified file.
241                      */

243                     if ((fi = fopen(*argv, "r")) == NULL) {
244                             if (!silent)
245                                     (void) fprintf(stderr, gettext(
246                                         "cat: cannot open %s: %s\n"),
247                                         *argv, strerror(errno));
248                             status = 2;
249                             continue;
250                     }
251             }

253             /*
254              * Stat source to make sure it is defined.
255              */

257             if (fstat(fileno(fi), &source) < 0) {
```

```
258                         if (!silent)
259                                 (void) fprintf(stderr,
260                                     gettext("cat: cannot stat %s: %s\n"),
261                                     (stdinflg) ? "-" : *argv, strerror(errno));
262                         status = 2;
263                         continue;
264                 }


267                 /*
268                  * If the source is not a character special file, socket or a
269                  * block special file, make sure it is not identical
270                  * to the target.
271                  */

273                 if (!S_ISCHR(target.st_mode) &&
274                     !S_ISBLK(target.st_mode) &&
275                     !S_ISSOCK(target.st_mode) &&
276                     IDENTICAL(target, source)) {
277                         if (!silent) {
278                                 (void) fprintf(stderr, gettext("cat: "
279                                     "input/output files '%s' identical\n"),
274                         if (!silent)
275                         (void) fprintf(stderr,
276                             gettext("cat: input/output files '%s' identical\n"),
280                                     stdinflg?"-": *argv);
281                         }

283                         if (fclose(fi) != 0)
284                                 (void) fprintf(stderr,
285                                     gettext("cat: close error: %s\n"),
286                                     strerror(errno));
287                         status = 2;
288                         continue;
289                 }
290                 ibsize = source.st_blksize;

292                 /*
293                  * If in visible mode and/or nflg, use vncat;
294                  * otherwise, use cat.
295                  */

297                 if (visi_mode || nflg)
298                         estatus = vncat(fi);
299                 else
300                         estatus = cat(fi, &source, &target,
301                             fi != stdin ? *argv : "standard input");

303                 if (estatus)
304                         status = estatus;

306                 /*
307                  * If the input is not stdin, close the source file.
308                  */

310                 if (fi != stdin) {
311                         if (fclose(fi) != 0)
312                                 if (!silent)
313                                         (void) fprintf(stderr,
314                                             gettext("cat: close error: %s\n"),
315                                             strerror(errno));
316                 }
317         }

319         /*
320          * Display any error with stdout operations.
```



```
321          */

323         if (fclose(stdout) != 0) {
324                 if (!silent)
325                         perror(gettext("cat: close error"));
326                 status = 2;
327         }
328         return (status);
329 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   13089 Thu Feb 28 11:26:01 2019
new/usr/src/cmd/chown/chown.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T     */
/*        All Rights Reserved   */

/*
 * Portions of this source code were derived from Berkeley 4.3 BSD
 * under license from the Regents of the University of California.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * chown [-fhR] uid[:gid] file ...
 * chown -R [-f] [-H|-L|-P] uid[:gid] file ...
 * chown -s [-fhR] ownersid[:groupsid] file ...
 * chown -s -R [-f] [-H|-L|-P] ownersid[:groupsid] file ...
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/avl.h>
#include <pwd.h>
#include <grp.h>
#include <unistd.h>
#include <locale.h>
#include <errno.h>
#include <libcmdutils.h>
```



```
#include <aclutils.h>

static struct           passwd  *pwd;
static struct           group   *grp;
static struct           stat    stbuf;
static uid_t            uid = (uid_t)-1;
static gid_t            gid = (gid_t)-1;
static int              status = 0;     /* total number of errors received */
static int              hflag = 0,
                        rflag = 0,
                        fflag = 0,
                        Hflag = 0,
                        Lflag = 0,
                        Pflag = 0,
                        sflag = 0;
static avl_tree_t       *tree;

static int              Perror(char *);
static int              isnumber(char *);
static void             chownr(char *, uid_t, gid_t);
static void             usage();

#ifdef XPG4
/*
 * Check to see if we are to follow symlinks specified on the command line.
 * This assumes we've already checked to make sure neither -h or -P was
 * specified, so we are just looking to see if -R -H, or -R -L was specified,
 * or, since -R has the same behavior as -R -L, if -R was specified by itself.
 * Therefore, all we really need to check for is if -R was specified.
 */
#define FOLLOW_CL_LINKS (rflag)
#else
/*
 * Check to see if we are to follow symlinks specified on the command line.
 * This assumes we've already checked to make sure neither -h or -P was
 * specified, so we are just looking to see if -R -H, or -R -L was specified.
 * Note: -R by itself will change the ownership of a directory referenced by a
 * symlink however it will now follow the symlink to any other part of the
 * file hierarchy.
 */
#define FOLLOW_CL_LINKS (rflag && (Hflag || Lflag))
#endif

#ifdef XPG4
/*
 * Follow symlinks when traversing directories.  Since -R behaves the
 * same as -R -L, we always want to follow symlinks to other parts
 * of the file hierarchy unless -H was specified.
 */
#define FOLLOW_D_LINKS  (!Hflag)
#else
/*
 * Follow symlinks when traversing directories.  Only follow symlinks
 * to other parts of the file hierarchy if -L was specified.
 */
#define FOLLOW_D_LINKS  (Lflag)
#endif

#define CHOWN(f, u, g)  if (chown(f, u, g) < 0) { \
                                status += Perror(f); \
                        }
#define LCHOWN(f, u, g) if (lchown(f, u, g) < 0) { \
                                status += Perror(f); \
                        }
```

```
125 int
126 main(int argc, char *argv[])
127 {
128         int c;
129         int ch;
130         char *grpp;                     /* pointer to group name arg */
131         extern int optind;
132         int errflg = 0;

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

140         while ((ch = getopt(argc, argv, "hRfHLPs")) != EOF) {
141                 switch (ch) {
142                 case 'h':
143                         hflag++;
144                         break;

146                 case 'R':
147                         rflag++;
148                         break;

150                 case 'f':
151                         fflag++;
152                         break;

154                 case 'H':
155                         /*
156                          * If more than one of -H, -L, and -P
157                          * are specified, only the last option
158                          * specified determines the behavior of
159                          * chown.
160                          */
161                         Lflag = Pflag = 0;
162                         Hflag++;
163                         break;

165                 case 'L':
166                         Hflag = Pflag = 0;
167                         Lflag++;
168                         break;

170                 case 'P':
171                         Hflag = Lflag = 0;
172                         Pflag++;
173                         break;

175                 case 's':
176                         sflag++;
177                         break;

179                 default:
180                         errflg++;
181                         break;
182                 }
183         }
184         /*
185          * Check for sufficient arguments
186          * or a usage error.
187          */

189         argc -= optind;
190         argv = &argv[optind];
```



```
192         if (errflg || (argc < 2) ||
193             ((Hflag || Lflag || Pflag) && !rflag) ||
194             ((Hflag || Lflag || Pflag) && hflag)) {
195                 usage();
196         }

198         /*
199          * POSIX.2
200          * Check for owner[:group]
201          */
202         if ((grpp = strchr(argv[0], ':')) != NULL) {
203                 *grpp++ = 0;

205                 if (sflag) {
206                         if (sid_to_id(grpp, B_FALSE, &gid)) {
207                                 (void) fprintf(stderr, gettext(
208                                     "chown: invalid owning group sid %s\n"),
209                                     grpp);
210                                 exit(2);
211                         }
212                 } else if ((grp = getgrnam(grpp)) != NULL) {
213                         gid = grp->gr_gid;
214                 } else {
215                         if (isnumber(grpp)) {
216                                 errno = 0;
217                                 gid = (gid_t)strtoul(grpp, NULL, 10);
218                                 if (errno != 0) {
219                                         if (errno == ERANGE) {
220                                                 (void) fprintf(stderr, gettext(
221                                                 "chown: group id too large\n"));
222                                                 exit(2);
223                                         } else {
224                                                 (void) fprintf(stderr, gettext(
225                                                 "chown: invalid group id\n"));
226                                                 exit(2);
227                                         }
228                                 }
229                         } else {
230                                 (void) fprintf(stderr, gettext(
231                                     "chown: unknown group id %s\n"), grpp);
232                                 exit(2);
233                         }
234                 }
235         }

237         if (sflag) {
238                 if (sid_to_id(argv[0], B_TRUE, &uid)) {
239                         (void) fprintf(stderr, gettext(
240                             "chown: invalid owner sid %s\n"), argv[0]);
241                         exit(2);
242                 }
243         } else if ((pwd = getpwnam(argv[0])) != NULL) {
244                 uid = pwd->pw_uid;
245         } else {
246                 if (isnumber(argv[0])) {
247                         errno = 0;
248                         uid = (uid_t)strtoul(argv[0], NULL, 10);
249                         if (errno != 0) {
250                                 if (errno == ERANGE) {
251                                         (void) fprintf(stderr, gettext(
252                                         "chown: user id too large\n"));
253                                         exit(2);
254                                 } else {
255                                         (void) fprintf(stderr, gettext(
256                                         "chown: invalid user id\n"));
```

```
257                                     exit(2);
258                             }
259                     }
260             } else {
261                     (void) fprintf(stderr, gettext(
262                     "chown: unknown user id %s\n"), argv[0]);
263                     exit(2);
264             }
265     }

267     for (c = 1; c < argc; c++) {
268             tree = NULL;
269             if (lstat(argv[c], &stbuf) < 0) {
270                     status += Perror(argv[c]);
271                     continue;
272             }
273             if (rflag && ((stbuf.st_mode & S_IFMT) == S_IFLNK)) {
274                     if (hflag || Pflag) {
275                             /*
276                              * Change the ownership of the symlink
277                              * specified on the command line.
278                              * Don't follow the symbolic link to
279                              * any other part of the file hierarchy.
280                              */
281                             LCHOWN(argv[c], uid, gid);
282                     } else {
283                             struct stat stbuf2;
284                             if (stat(argv[c], &stbuf2) < 0) {
285                                     status += Perror(argv[c]);
286                                     continue;
287                             }
288                             /*
289                              * We know that we are to change the
290                              * ownership of the file referenced by the
291                              * symlink specified on the command line.
292                              * Now check to see if we are to follow
293                              * the symlink to any other part of the
294                              * file hierarchy.
295                              */
296                             if (FOLLOW_CL_LINKS) {
297                                     if ((stbuf2.st_mode & S_IFMT)
298                                         == S_IFDIR) {
299                                             /*
300                                              * We are following symlinks so
301                                              * traverse into the directory.
302                                              * Add this node to the search
303                                              * tree so we don't get into an
304                                              * endless loop.
305                                              */
306                                             if (add_tnode(&tree,
307                                                 stbuf2.st_dev,
308                                                 stbuf2.st_ino) == 1) {
309                                                     chownr(argv[c],
310                                                         uid, gid);
311                                             } else {
312                                                     /*
313                                                      * Error occurred.
314                                                      * rc can't be 0
315                                                      * as this is the first
316                                                      * node to be added to
317                                                      * the search tree.
318                                                      */
319                                                     status += Perror(
320                                                         argv[c]);
321                                             }
322                                     } else {
```



```
323                                             /*
324                                              * Change the user ID of the
325                                              * file referenced by the
326                                              * symlink.
327                                              */
328                                             CHOWN(argv[c], uid, gid);
329                                     }
330                             } else {
331                                     /*
332                                      * Change the user ID of the file
333                                      * referenced by the symbolic link.
334                                      */
335                                     CHOWN(argv[c], uid, gid);
336                             }
337                     }
338             } else if (rflag && ((stbuf.st_mode & S_IFMT) == S_IFDIR)) {
339                     /*
340                      * Add this node to the search tree so we don't
341                      * get into a endless loop.
342                      */
343                     if (add_tnode(&tree, stbuf.st_dev,
344                         stbuf.st_ino) == 1) {
345                             chownr(argv[c], uid, gid);
346                     } else {
347                             /*
348                              * An error occurred while trying
349                              * to add the node to the tree.
350                              * Continue on with next file
351                              * specified.  Note: rc shouldn't
352                              * be 0 as this was the first node
353                              * being added to the search tree.
354                              */
355                             status += Perror(argv[c]);
356                     }
357             } else if (hflag || Pflag) {
358                     LCHOWN(argv[c], uid, gid);
359             } else {
360                     CHOWN(argv[c], uid, gid);
361             }
362     }
363     return (status);
364 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   19822 Thu Feb 28 11:26:01 2019
new/usr/src/cmd/cmd-inet/common/kcmd.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 1983 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by the University of California, Berkeley.  The name of the
 * University may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

/* derived from @(#)rcmd.c      5.17 (Berkeley) 6/27/88 */

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <pwd.h>
#include <sys/param.h>
#include <sys/types.h>
#include <fcntl.h>

#include <signal.h>
#include <sys/file.h>
#include <sys/socket.h>
#include <sys/stat.h>

#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <locale.h>
#include <syslog.h>

#include <errno.h>
#include <com_err.h>
#include <k5-int.h>
#include <kcmd.h>

static char *default_service = "host";

#define KCMD_BUFSIZ     102400
#define KCMD8_BUFSIZ    (4096 - 256)
/*
```



```
 * For compatibility with earlier versions of Solaris and other OS
 * (kerborized rsh uses 4KB of RSH_BUFSIZE)- 256 to make sure
 * there is room
 */
static int deswrite_compat(int, char *, int, int);

#define KCMD_KEYUSAGE   1026

static char storage[KCMD_BUFSIZ];
static int nstored = 0;
static int MAXSIZE = (KCMD_BUFSIZ + 8);
static char *store_ptr = storage;
static krb5_data desinbuf, desoutbuf;

static boolean_t encrypt_flag = B_FALSE;
static krb5_context kcmd_context;

/* XXX Overloaded: use_ivecs!=0 -> new protocol, inband signalling, etc.  */
static boolean_t use_ivecs = B_FALSE;
static krb5_data encivec_i[2], encivec_o[2];
static krb5_keyusage enc_keyusage_i[2], enc_keyusage_o[2];
static krb5_enctype final_enctype;
static krb5_keyblock *skey;

/* ARGSUSED */
int
kcmd(int *sock, char **ahost, ushort_t rport,
        char *locuser, char *remuser,
        char *cmd, int *fd2p, char *service, char *realm,
        krb5_context bsd_context, krb5_auth_context *authconp,
        krb5_creds **cred, krb5_int32 *seqno, krb5_int32 *server_seqno,
        krb5_flags authopts,
        int anyport, enum kcmd_proto *protonump)
{
        int s = -1;
        sigset_t oldmask, urgmask;
        struct sockaddr_in sin;
        struct sockaddr_storage from;
        krb5_creds *get_cred = NULL;
        krb5_creds *ret_cred = NULL;
        char c;
        struct hostent *hp;
        int rc;
        char *host_save = NULL;
        krb5_error_code status;
        krb5_ap_rep_enc_part *rep_ret;
        krb5_error      *error = 0;
        krb5_ccache cc;
        krb5_data outbuf;
        krb5_flags options = authopts;
        krb5_auth_context auth_context = NULL;
        char *cksumbuf;
        krb5_data cksumdat;
        int bsize = 0;
        char *kcmd_version;
        enum kcmd_proto protonum = *protonump;

        bsize = strlen(cmd) + strlen(remuser) + 64;
        if ((cksumbuf = malloc(bsize)) == 0) {
                (void) fprintf(stderr, gettext("Unable to allocate"
                                               " memory for checksum buffer.\n"));
                return (-1);
        }
        (void) snprintf(cksumbuf, bsize, "%u:", ntohs(rport));
        if (strlcat(cksumbuf, cmd, bsize) >= bsize) {
                (void) fprintf(stderr, gettext("cmd buffer too long.\n"));
```

```
126                 free(cksumbuf);
127                 return (-1);
128         }
129         if (strlcat(cksumbuf, remuser, bsize) >= bsize) {
130                 (void) fprintf(stderr, gettext("remuser too long.\n"));
131                 free(cksumbuf);
132                 return (-1);
133         }
134         cksumdat.data = cksumbuf;
135         cksumdat.length = strlen(cksumbuf);

137         hp = gethostbyname(*ahost);
138         if (hp == 0) {
139                 (void) fprintf(stderr,
140                         gettext("%s: unknown host\n"), *ahost);
141                 return (-1);
142         }

144         if ((host_save = (char *)strdup(hp->h_name)) == NULL) {
145                 (void) fprintf(stderr, gettext("kcmd: no memory\n"));
146                 return (-1);
147         }

149         /* If no service is given set to the default service */
150         if (!service) service = default_service;

152         if (!(get_cred = (krb5_creds *)calloc(1, sizeof (krb5_creds)))) {
153                 (void) fprintf(stderr, gettext("kcmd: no memory\n"));
154                 return (-1);
155         }
156         (void) sigemptyset(&urgmask);
157         (void) sigaddset(&urgmask, SIGURG);
158         (void) sigprocmask(SIG_BLOCK, &urgmask, &oldmask);

160         status = krb5_sname_to_principal(bsd_context, host_save, service,
161                                          KRB5_NT_SRV_HST, &get_cred->server);
162         if (status) {
163                 (void) fprintf(stderr,
164                         gettext("kcmd: "
165                                 "krb5_sname_to_principal failed: %s\n"),
166                         error_message(status));
167                 status = -1;
168                 goto bad;
169         }

171         if (realm && *realm) {
172                 (void) krb5_xfree(
173                         krb5_princ_realm(bsd_context, get_cred->server)->data);
174                 krb5_princ_set_realm_length(bsd_context, get_cred->server,
175                                             strlen(realm));
176                 krb5_princ_set_realm_data(bsd_context, get_cred->server,
177                                                 strdup(realm));
178         }

180         s = socket(AF_INET, SOCK_STREAM, 0);
181         if (s < 0) {
182                 perror(gettext("Error creating socket"));
183                 status = -1;
184                 goto bad;
185         }
186         /*
187          * Kerberos only supports IPv4 addresses for now.
188          */
189         if (hp->h_addrtype == AF_INET) {
190                 sin.sin_family = hp->h_addrtype;
191                 (void) memcpy((void *)&sin.sin_addr,
```



```
192                             hp->h_addr, hp->h_length);
193                 sin.sin_port = rport;
194         } else {
195                 syslog(LOG_ERR, "Address type %d not supported for "
196                     "Kerberos", hp->h_addrtype);
197                 status = -1;
198                 goto bad;
199         }

201         if (connect(s, (struct sockaddr *)&sin, sizeof (sin)) < 0) {
202                 perror(host_save);
203                 status = -1;
204                 goto bad;
205         }

207         if (fd2p == 0) {
208                 (void) write(s, "", 1);
209         } else {
210                 char num[16];
211                 int s2;
212                 int s3;
213                 struct sockaddr_storage sname;
214                 struct sockaddr_in *sp;
215                 int len = sizeof (struct sockaddr_storage);

217                 s2 = socket(AF_INET, SOCK_STREAM, 0);
218                 if (s2 < 0) {
219                         status = -1;
220                         goto bad;
221                 }
222                 (void) memset((char *)&sin, 0, sizeof (sin));
223                 sin.sin_family = AF_INET;
224                 sin.sin_addr.s_addr = INADDR_ANY;
225                 sin.sin_port = 0;

227                 if (bind(s2, (struct sockaddr *)&sin, sizeof (sin)) < 0) {
228                         perror(gettext("error binding socket"));
229                         (void) close(s2);
230                         status = -1;
231                         goto bad;
232                 }
233                 if (getsockname(s2, (struct sockaddr *)&sname, &len) < 0) {
234                         perror(gettext("getsockname error"));
235                         (void) close(s2);
236                         status = -1;
237                         goto bad;
238                 }
239                 sp = (struct sockaddr_in *)&sname;
240                 (void) listen(s2, 1);
241                 (void) snprintf(num, sizeof (num), "%d",
242                                 htons((ushort_t)sp->sin_port));
243                 if (write(s, num, strlen(num)+1) != strlen(num)+1) {
244                         perror(gettext("write: error setting up stderr"));
245                         (void) close(s2);
246                         status = -1;
247                         goto bad;
248                 }

250                 s3 = accept(s2, (struct sockaddr *)&from, &len);
251                 (void) close(s2);
252                 if (s3 < 0) {
253                         perror(gettext("accept"));
254                         status = -1;
255                         goto bad;
256                 }
257                 *fd2p = s3;
```

```
258                 if (SOCK_FAMILY(from) == AF_INET) {
259                         if (!anyport && SOCK_PORT(from) >= IPPORT_RESERVED) {
260                                 (void) fprintf(stderr,
261                                     gettext("socket: protocol "
262                                             "failure in circuit setup.\n"));
263                                 status = -1;
264                                 goto bad2;
265                         }
266                 } else {
267                         (void) fprintf(stderr,
268                                 gettext("Kerberos does not support "
269                                         "address type %d\n"),
270                                 SOCK_FAMILY(from));
271                         status = -1;
272                         goto bad2;
273                 }
274         }

276         if (status = krb5_cc_default(bsd_context, &cc))
277                 goto bad2;

279         status = krb5_cc_get_principal(bsd_context, cc, &get_cred->client);
280         if (status) {
281                 (void) krb5_cc_close(bsd_context, cc);
282                 goto bad2;
283         }

285         /* Get ticket from credentials cache or kdc */
286         status = krb5_get_credentials(bsd_context, 0, cc, get_cred, &ret_cred);
287         (void) krb5_cc_close(bsd_context, cc);
288         if (status) goto bad2;

290         /* Reset internal flags; these should not be sent. */
291         authopts &= (~OPTS_FORWARD_CREDS);
292         authopts &= (~OPTS_FORWARDABLE_CREDS);

294         if ((status = krb5_auth_con_init(bsd_context, &auth_context)))
295                 goto bad2;

297         if ((status = krb5_auth_con_setflags(bsd_context, auth_context,
298                                              KRB5_AUTH_CONTEXT_RET_TIME)))
299                 goto bad2;

301         /* Only need local address for mk_cred() to send to krlogind */
302         if ((status = krb5_auth_con_genaddrs(bsd_context, auth_context, s,
303                         KRB5_AUTH_CONTEXT_GENERATE_LOCAL_FULL_ADDR)))
304                 goto bad2;

306         if (protonum == KCMD_PROTOCOL_COMPAT_HACK) {
307                 krb5_boolean is_des;
308                 status = krb5_c_enctype_compare(bsd_context,
309                                                 ENCTYPE_DES_CBC_CRC,
310                                                 ret_cred->keyblock.enctype,
311                                                 &is_des);
312                 if (status)
313                         goto bad2;
314                 protonum = is_des ? KCMD_OLD_PROTOCOL : KCMD_NEW_PROTOCOL;
315         }

317         switch (protonum) {
318         case KCMD_NEW_PROTOCOL:
319                 authopts |= AP_OPTS_USE_SUBKEY;
320                 kcmd_version = "KCMDV0.2";
321                 break;
322         case KCMD_OLD_PROTOCOL:
323                 kcmd_version = "KCMDV0.1";
```



```
324                 break;
325         default:
326                 status = -1;
327                 goto bad2;
328         }

330         /*
331          * Call the Kerberos library routine to obtain an authenticator,
332          * pass it over the socket to the server, and obtain mutual
333          * authentication.
334          */
335         status = krb5_sendauth(bsd_context, &auth_context, (krb5_pointer) &s,
336                         kcmd_version, ret_cred->client, ret_cred->server,
337                         authopts, &cksumdat, ret_cred, 0, &error,
338                         &rep_ret, NULL);
339         krb5_xfree(cksumdat.data);
340         if (status) {
341                 (void) fprintf(stderr, gettext("Couldn't authenticate"
342                                                " to server: %s\n"),
343                                error_message(status));
344                 if (error) {
345                         (void) fprintf(stderr, gettext("Server returned error"
346                                                        " code %d (%s)\n"),
347                                 error->error,
348                                 error_message(ERROR_TABLE_BASE_krb5 +
349                                               error->error));
350                         if (error->text.length)
351                                 (void) fprintf(stderr,
352                                                gettext("Error text"
353                                                        " sent from server: %s\n"),
354                                                error->text.data);
355                 }
356                 if (error) {
357                         krb5_free_error(bsd_context, error);
358                         error = 0;
359                 }
360                 goto bad2;
361         }
362         if (rep_ret && server_seqno) {
363                 *server_seqno = rep_ret->seq_number;
364                 krb5_free_ap_rep_enc_part(bsd_context, rep_ret);
365         }

367         (void) write(s, remuser, strlen(remuser)+1);
368         (void) write(s, cmd, strlen(cmd)+1);
369         if (locuser)
370                 (void) write(s, locuser, strlen(locuser)+1);
371         else
372                 (void) write(s, "", 1);

374         if (options & OPTS_FORWARD_CREDS) {   /* Forward credentials */
375                 if (status = krb5_fwd_tgt_creds(bsd_context, auth_context,
376                                                 host_save,
377                                                 ret_cred->client, ret_cred->server,
378                                                 0, options & OPTS_FORWARDABLE_CREDS,
379                                                 &outbuf)) {
380                         (void) fprintf(stderr,
381                                        gettext("kcmd: Error getting"
382                                                " forwarded creds\n"));
383                         goto bad2;
384                 }
385                 /* Send forwarded credentials */
386                 if (status = krb5_write_message(bsd_context, (krb5_pointer)&s,
387                                                 &outbuf))
388                         goto bad2;
389         } else { /* Dummy write to signal no forwarding */
```

```
390                 outbuf.length = 0;
391                 if (status = krb5_write_message(bsd_context,
392                                                 (krb5_pointer)&s, &outbuf))
393                         goto bad2;
394         }

396         if ((rc = read(s, &c, 1)) != 1) {
397                 if (rc == -1) {
398                         perror(*ahost);
399                 } else {
400                         (void) fprintf(stderr, gettext("kcmd: bad connection "
401                                        "with remote host\n"));
402                 }
403                 status = -1;
404                 goto bad2;
405         }
406         if (c != 0) {
407                 while (read(s, &c, 1) == 1) {
408                         (void) write(2, &c, 1);
409                         if (c == '\n')
410                                 break;
411                 }
412                 status = -1;
413                 goto bad2;
414         }
415         (void) sigprocmask(SIG_SETMASK, &oldmask, (sigset_t *)0);
416         *sock = s;

418         /* pass back credentials if wanted */
419         if (cred)
420                 (void) krb5_copy_creds(bsd_context, ret_cred, cred);

415         if (cred) (void) krb5_copy_creds(bsd_context, ret_cred, cred);
422         krb5_free_creds(bsd_context, ret_cred);

424         /*
425          * Initialize *authconp to auth_context, so
426          * that the clients can make use of it
427          */
428         *authconp = auth_context;

430         return (0);
431 bad2:
432         if (fd2p != NULL)
433                 (void) close(*fd2p);
434 bad:
435         if (s > 0)
436                 (void) close(s);
437         if (get_cred)
438                 krb5_free_creds(bsd_context, get_cred);
439         if (ret_cred)
440                 krb5_free_creds(bsd_context, ret_cred);
441         if (host_save)
442                 free(host_save);
443         (void) sigprocmask(SIG_SETMASK, &oldmask, (sigset_t *)0);
444         return (status);
445 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   39953 Thu Feb 28 11:26:01 2019
new/usr/src/cmd/cmd-inet/usr.bin/finger.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
/*
 * 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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T     */
/*        All Rights Reserved   */

/*
 * Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 * This is a finger program.  It prints out useful information about users
 * by digging it up from various system files.
 *
 * There are three output formats, all of which give login name, teletype
 * line number, and login time.  The short output format is reminiscent
 * of finger on ITS, and gives one line of information per user containing
 * in addition to the minimum basic requirements (MBR), the user's full name,
 * idle time and location.
 * The quick style output is UNIX who-like, giving only name, teletype and
 * login time.  Finally, the long style output give the same information
 * as the short (in more legible format), the home directory and shell
 * of the user, and, if it exits, a copy of the file .plan in the users
 * home directory.  Finger may be called with or without a list of people
 * to finger -- if no list is given, all the people currently logged in
 * are fingered.
```



```
 *
 * The program is validly called by one of the following:
 *
 *      finger                  {short form list of users}
 *      finger -l               {long form list of users}
 *      finger -b               {briefer long form list of users}
 *      finger -q               {quick list of users}
 *      finger -i               {quick list of users with idle times}
 *      finger -m               {matches arguments against only username}
 *      finger -f               {suppress header in non-long form}
 *      finger -p               {suppress printing of .plan file}
 *      finger -h               {suppress printing of .project file}
 *      finger -i               {forces "idle" output format}
 *      finger namelist         {long format list of specified users}
 *      finger -s namelist      {short format list of specified users}
 *      finger -w namelist      {narrow short format list of specified users}
 *
 * where 'namelist' is a list of users login names.
 * The other options can all be given after one '-', or each can have its
 * own '-'.  The -f option disables the printing of headers for short and
 * quick outputs.  The -b option briefens long format outputs.  The -p
 * option turns off plans for long format outputs.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <utmpx.h>
#include <sys/signal.h>
#include <pwd.h>
#include <stdio.h>
#include <lastlog.h>
#include <ctype.h>
#include <sys/time.h>
#include <time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <locale.h>
#include <sys/select.h>
#include <stdlib.h>
#include <strings.h>
#include <fcntl.h>
#include <curses.h>
#include <unctrl.h>
#include <maillock.h>
#include <deflt.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <macros.h>

static char gecos_ignore_c = '*';       /* ignore this in real name */
static char gecos_sep_c = ',';          /* separator in pw_gecos field */
static char gecos_samename = '&';       /* repeat login name in real name */

#define TALKABLE        0220            /* tty is writable if this mode */

#define NMAX    sizeof (((struct utmpx *)0)->ut_name)
#define LMAX    sizeof (((struct utmpx *)0)->ut_line)
#define HMAX    sizeof (((struct utmpx *)0)->ut_host)

struct person {                         /* one for each person fingered */
        char *name;                     /* name */
        char tty[LMAX+1];               /* null terminated tty line */
        char host[HMAX+1];              /* null terminated remote host name */
        char *ttyloc;                   /* location of tty line, if any */
        time_t loginat;                 /* time of (last) login */
```

```
 126         time_t idletime;               /* how long idle (if logged in) */
 127         char *realname;                /* pointer to full name */
 128         struct passwd *pwd;            /* structure of /etc/passwd stuff */
 129         char loggedin;                 /* person is logged in */
 130         char writable;                 /* tty is writable */
 131         char original;                 /* this is not a duplicate entry */
 132         struct person *link;           /* link to next person */
 133 };
______unchanged_portion_omitted_
```

```
**********************************************************
   42671 Thu Feb 28 11:26:02 2019
new/usr/src/cmd/compress/compress.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

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

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */


/*
 * Copyright (c) 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * Compress - data compression program
 */
#define min(a, b)       ((a > b) ? b : a)

/*
 * machine variants which require cc -Dmachine:  pdp11, z8000, pcxt
 */

/*
 * Set USERMEM to the maximum amount of physical user memory available
 * in bytes.  USERMEM is used to determine the maximum BITS that can be used
 * for compression.
 *
 * SACREDMEM is the amount of physical memory saved for others; compress
 * will hog the rest.
 */
#ifndef SACREDMEM
#define SACREDMEM       0
#endif

#ifndef USERMEM
#define USERMEM         450000  /* default user memory */
#endif

#ifdef USERMEM
#if USERMEM >= (433484+SACREDMEM)
#define PBITS   16
#else
#if USERMEM >= (229600+SACREDMEM)
#define PBITS   15
#else
#if USERMEM >= (127536+SACREDMEM)
#define PBITS   14
#else
#if USERMEM >= (73464+SACREDMEM)
#define PBITS   13
#else
#define PBITS   12
```



```
#endif
#endif
#endif
#endif
#undef USERMEM
#endif /* USERMEM */

#ifdef PBITS            /* Preferred BITS for this memory size */
#ifndef BITS
#define BITS PBITS
#endif /* BITS */
#endif /* PBITS */

#if BITS == 16
#define HSIZE   69001           /* 95% occupancy */
#endif
#if BITS == 15
#define HSIZE   35023           /* 94% occupancy */
#endif
#if BITS == 14
#define HSIZE   18013           /* 91% occupancy */
#endif
#if BITS == 13
#define HSIZE   9001            /* 91% occupancy */
#endif
#if BITS <= 12
#define HSIZE   5003            /* 80% occupancy */
#endif

#define OUTSTACKSIZE    (2<<BITS)

/*
 * a code_int must be able to hold 2**BITS values of type int, and also -1
 */
#if BITS > 15
typedef long int        code_int;
#else
typedef int             code_int;
#endif

typedef long int        count_int;
typedef long long       count_long;

typedef unsigned char   char_type;

static char_type magic_header[] = { "\037\235" }; /* 1F 9D */

/* Defines for third byte of header */
#define BIT_MASK        0x1f
#define BLOCK_MASK      0x80
/*
 * Masks 0x40 and 0x20 are free.  I think 0x20 should mean that there is
 * a fourth header byte(for expansion).
 */
#define INIT_BITS 9                     /* initial number of bits/code */

/*
 * compress.c - File compression ala IEEE Computer, June 1984.
 */
static char rcs_ident[] =
        "$Header: compress.c,v 4.0 85/07/30 12:50:00 joe Release $";

#include <ctype.h>
#include <signal.h>
#include <sys/param.h>
#include <locale.h>
```

```
125 #include <langinfo.h>
126 #include <sys/acl.h>
127 #include <utime.h>
128 #include <libgen.h>
129 #include <setjmp.h>
130 #include <aclutils.h>
131 #include <libcmdutils.h>
132 #include "getresponse.h"


135 static int n_bits;                      /* number of bits/code */
136 static int maxbits = BITS;      /* user settable max # bits/code */
137 static code_int maxcode;        /* maximum code, given n_bits */
138                         /* should NEVER generate this code */
139 static code_int maxmaxcode = 1 << BITS;
140 #define MAXCODE(n_bits) ((1 << (n_bits)) - 1)

142 static count_int htab [OUTSTACKSIZE];
143 static unsigned short codetab [OUTSTACKSIZE];

145 #define htabof(i)       htab[i]
146 #define codetabof(i)    codetab[i]
147 static code_int hsize = HSIZE; /* for dynamic table sizing */
148 static off_t    fsize;  /* file size of input file */

150 /*
151  * To save much memory, we overlay the table used by compress() with those
152  * used by decompress().  The tab_prefix table is the same size and type
153  * as the codetab.  The tab_suffix table needs 2**BITS characters.  We
154  * get this from the beginning of htab.  The output stack uses the rest
155  * of htab, and contains characters.  There is plenty of room for any
156  * possible stack (stack used to be 8000 characters).
157  */

159 #define tab_prefixof(i)         codetabof(i)
160 #define tab_suffixof(i)         ((char_type *)(htab))[i]
161 #define de_stack                ((char_type *)&tab_suffixof(1<<BITS))
162 #define stack_max               ((char_type *)&tab_suffixof(OUTSTACKSIZE))

164 static code_int free_ent = 0; /* first unused entry */
165 static int newline_needed = 0;
166 static int didnt_shrink = 0;
167 static int perm_stat = 0;       /* permanent status */

169 static code_int getcode();

171         /* Use a 3-byte magic number header, unless old file */
172 static int nomagic = 0;
173         /* Write output on stdout, suppress messages */
174 static int zcat_flg = 0;        /* use stdout on all files */
175 static int zcat_cmd = 0;        /* zcat cmd */
176 static int use_stdout = 0;      /* set for each file processed */
177         /* Don't unlink output file on interrupt */
178 static int precious = 1;
179 static int quiet = 1;   /* don't tell me about compression */

181 /*
182  * block compression parameters -- after all codes are used up,
183  * and compression rate changes, start over.
184  */
185 static int block_compress = BLOCK_MASK;
186 static int clear_flg = 0;
187 static long int ratio = 0;
188 #define CHECK_GAP 10000 /* ratio check interval */
189 static count_long checkpoint = CHECK_GAP;
190 /*
```



```
191  * the next two codes should not be changed lightly, as they must not
192  * lie within the contiguous general code space.
193  */
194 #define FIRST   257     /* first free entry */
195 #define CLEAR   256     /* table clear output code */

197 static int force = 0;
198 static char ofname [MAXPATHLEN];

200 static int Vflg = 0;
201 static int vflg = 0;
202 static int qflg = 0;
203 static int bflg = 0;
204 static int Fflg = 0;
205 static int dflg = 0;
206 static int cflg = 0;
207 static int Cflg = 0;

209 #ifdef DEBUG
210 int verbose = 0;
211 int debug = 0;
212 #endif /* DEBUG */

214 static void (*oldint)();
215 static int bgnd_flag;

217 static int do_decomp = 0;

219 static char *progname;
220 static char *optstr;
221 /*
222  * Fix lint errors
223  */

225 static char *local_basename(char *);

227 static int  addDotZ(char *, size_t);

229 static void Usage(void);
230 static void cl_block(count_long);
231 static void cl_hash(count_int);
232 static void compress(void);
233 static void copystat(char *, struct stat *, char *);
234 static void decompress(void);
235 static void ioerror(void);
236 static void onintr();
237 static void oops();
238 static void output(code_int);
239 static void prratio(FILE *, count_long, count_long);
240 static void version(void);

242 #ifdef DEBUG
243 static int in_stack(int, int);
244 static void dump_tab(void);
245 static void printcodes(void);
246 #endif

248 /* For error-handling */

250 static jmp_buf env;

252 /* For input and ouput */

254 static FILE *inp;               /* the current input file */
255 static FILE *infile;            /* disk-based input stream */
256 static FILE *outp;              /* current output file */
```

```
257 static FILE *outfile;           /* disk-based output stream */

259 /* For output() */

261 static char buf[BITS];

263 static char_type lmask[9] =
264         {0xff, 0xfe, 0xfc, 0xf8, 0xf0, 0xe0, 0xc0, 0x80, 0x00};
265 static char_type rmask[9] =
266         {0x00, 0x01, 0x03, 0x07, 0x0f, 0x1f, 0x3f, 0x7f, 0xff};

268 /* For compress () */

270 static int offset;
271 static count_long bytes_out;    /* length of compressed output */
272         /* # of codes output (for debugging) */

274 /* For dump_tab() */

276 #define STACK_SIZE      15000
277 #ifdef DEBUG
278 code_int sorttab[1<<BITS];      /* sorted pointers into htab */
279 #endif

281 /* Extended system attribute support */

283 static int saflg = 0;

285 /*
286  * ************************************************************
287  * TAG( main )
288  *
289  * Algorithm from "A Technique for High Performance Data Compression",
290  * Terry A. Welch, IEEE Computer Vol 17, No 6 (June 1984), pp 8-19.
291  *
292  * Usage: compress [-dfvc/] [-b bits] [file ...]
293  * Inputs:
294  *      -d:         If given, decompression is done instead.
295  *
296  *      -c:         Write output on stdout, don't remove original.
297  *
298  *      -b:         Parameter limits the max number of bits/code.
299  *
300  *      -f:         Forces output file to be generated, even if one already
301  *                  exists, and even if no space is saved by compressing.
302  *                  If -f is not used, the user will be prompted if stdin is
303  *                  a tty, otherwise, the output file will not be overwritten.
304  *
305  *      -/          Copies extended attributes and extended system attributes.
306  *
307  *  -v:     Write compression statistics
308  *
309  *      file ...:   Files to be compressed.  If none specified, stdin
310  *                  is used.
311  * Outputs:
312  *      file.Z:     Compressed form of file with same mode, owner, and utimes
313  *      or stdout   (if stdin used as input)
314  *
315  * Assumptions:
316  * When filenames are given, replaces with the compressed version
317  * (.Z suffix) only if the file decreases in size.
318  * Algorithm:
319  * Modified Lempel-Ziv method (LZW).  Basically finds common
320  * substrings and replaces them with a variable size code.  This is
321  * deterministic, and can be done on the fly.  Thus, the decompression
322  * procedure needs no input table, but tracks the way the table was built.
```



```
323  */

325 int
326 main(int argc, char *argv[])
327 {
328         int overwrite = 0;      /* Do not overwrite unless given -f flag */
329         char tempname[MAXPATHLEN];
330         char line[LINE_MAX];
331         char **filelist, **fileptr;
332         char *cp;
333         struct stat statbuf;
334         struct stat ostatbuf;
335         int ch;                         /* XCU4 */
336         char    *p;
337         extern int optind, optopt;
338         extern char *optarg;
339         int dash_count = 0;             /* times "-" is on cmdline */

341         /* XCU4 changes */
342         (void) setlocale(LC_ALL, "");
343 #if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
344 #define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
345 #endif
346         (void) textdomain(TEXT_DOMAIN);

348         if (init_yes() < 0) {
349                 (void) fprintf(stderr, gettext(ERR_MSG_INIT_YES),
350                     strerror(errno));
351                 exit(1);
352         }

354         /* This bg check only works for sh. */
355         if ((oldint = signal(SIGINT, SIG_IGN)) != SIG_IGN) {
356                 (void) signal(SIGINT, onintr);
357                 (void) signal(SIGSEGV, oops);
358         }
359         bgnd_flag = oldint != SIG_DFL;

361         /* Allocate room for argv + "-" (if stdin needs to be added) */

363         filelist = fileptr = (char **)(malloc((argc + 1) * sizeof (*argv)));
364         *filelist = NULL;

366         if ((cp = rindex(argv[0], '/')) != 0) {
367                 cp++;
368         } else {
369                 cp = argv[0];
370         }

372         if (strcmp(cp, "uncompress") == 0) {
373                 do_decomp = 1;
374         } else if (strcmp(cp, "zcat") == 0) {
375                 do_decomp = 1;
376                 zcat_cmd = zcat_flg = 1;
377         }

379         progname = local_basename(argv[0]);

381         /*
382          * Argument Processing
383          * All flags are optional.
384          * -D = > debug
385          * -V = > print Version; debug verbose
386          * -d = > do_decomp
387          * -v = > unquiet
388          * -f = > force overwrite of output file
```

```
389          * -n = > no header: useful to uncompress old files
390          * -b     maxbits => maxbits.  If -b is specified,
391          *        then maxbits MUST be given also.
392          * -c = > cat all output to stdout
393          * -C = > generate output compatible with compress 2.0.
394          * if a string is left, must be an input filename.
395          */
396 #ifdef DEBUG
397         optstr = "b:cCdDfFnqvV/";
398 #else
399         optstr = "b:cCdfFnqvV/";
400 #endif

402         while ((ch = getopt(argc, argv, optstr)) != EOF) {
403                 /* Process all flags in this arg */
404                 switch (ch) {
405 #ifdef DEBUG
406                         case 'D':
407                                 debug = 1;
408                                 break;
409                         case 'V':
410                                 verbose = 1;
411                                 version();
412                                 break;
413 #else
414                         case 'V':
415                                 version();
416                                 Vflg++;
417                                 break;
418 #endif /* DEBUG */
419                         case 'v':
420                                 quiet = 0;
421                                 vflg++;
422                                 break;
423                         case 'd':
424                                 do_decomp = 1;
425                                 dflg++;
426                                 break;
427                         case 'f':
428                         case 'F':
429                                 Fflg++;
430                                 overwrite = 1;
431                                 force = 1;
432                                 break;
433                         case 'n':
434                                 nomagic = 1;
435                                 break;
436                         case 'C':
437                                 Cflg++;
438                                 block_compress = 0;
439                                 break;
440                         case 'b':
441                                 bflg++;
442                                 p = optarg;
443                                 if (!p) {
444                                         (void) fprintf(stderr, gettext(
445                                             "Missing maxbits\n"));
446                                         Usage();
447                                         exit(1);
448                                 }
449                                 maxbits = strtoul(optarg, &p, 10);
450                                 if (*p) {
451                                         (void) fprintf(stderr, gettext(
452                                             "Missing maxbits\n"));
453                                         Usage();
454                                         exit(1);
```



```
455                                 }
456                                 break;

458                         case 'c':
459                                 cflg++;
460                                 zcat_flg = 1;
461                                 break;
462                         case 'q':
463                                 qflg++;
464                                 quiet = 1;
465                                 break;
466                         case '/':
467                                 saflg++;
468                                 break;
469                         default:
470                                 (void) fprintf(stderr, gettext(
471                                     "Unknown flag: '%c'\n"), optopt);
472                                 Usage();
473                                 exit(1);
474                 }
475         } /* while */

477         /*
478          * Validate zcat syntax
479          */

481         if (zcat_cmd && (Fflg | Cflg | cflg |
482             bflg | qflg | dflg | nomagic)) {
483                 (void) fprintf(stderr, gettext(
484                     "Invalid Option\n"));
485                 Usage();
486                 exit(1);
487         }

489         /*
490          * Process the file list
491          */

493         for (; optind < argc; optind++) {
494                 if (strcmp(argv[optind], "-") == 0) {
495                         dash_count++;
496                 }

498                 *fileptr++ = argv[optind];      /* Build input file list */
499                 *fileptr = NULL;
500         }

502         if (dash_count > 1) {
503                 (void) fprintf(stderr,
504                     gettext("%s may only appear once in the file"
505                     " list\n"), "\"-\"");
506                 exit(1);
507         }

509         if (fileptr - filelist == 0) {
510                 *fileptr++ = "-";
511                 *fileptr = NULL;
512         }

514         if (fileptr - filelist > 1 && cflg && !do_decomp) {
515                 (void) fprintf(stderr,
516                     gettext("compress: only one file may be compressed"
517                     " to stdout\n"));
518                 exit(1);
519         }
```

```
521         if (maxbits < INIT_BITS)
522                 maxbits = INIT_BITS;
523         if (maxbits > BITS)
524                 maxbits = BITS;
525         maxmaxcode = 1 << maxbits;

527         /* Need to open something to close with freopen later */

529         if ((infile = fopen("/dev/null", "r")) == NULL) {
530                 (void) fprintf(stderr, gettext("Error opening /dev/null for "
531                     "input\n"));
532                 exit(1);
533         }

535         if ((outfile = fopen("/dev/null", "w")) == NULL) {
536                 (void) fprintf(stderr, gettext("Error opening /dev/null for "
537                     "output\n"));
538                 exit(1);
539         }

541         for (fileptr = filelist; *fileptr; fileptr++) {
542                 int jmpval = 0;
543                 didnt_shrink = 0;
544                 newline_needed = 0;

546                 if (do_decomp) {
547                         /* DECOMPRESSION */

549                         if (strcmp(*fileptr, "-") == 0) {
550                                 /* process stdin */
551                                 inp = stdin;
552                                 outp = stdout;
553                                 use_stdout = 1;
554                                 *fileptr = "stdin"; /* for error messages */
555                         } else {
556                                 /* process the named file */

558                                 inp = infile;
559                                 outp = outfile;
560                                 use_stdout = 0;

562                                 if (zcat_flg) {
563                                         use_stdout = 1;
564                                         outp = stdout;
565                                 }

567                                 /* Check for .Z suffix */

569                                 if (strcmp(*fileptr +
570                                     strlen(*fileptr) - 2, ".Z") != 0) {
571                                         /* No .Z: tack one on */

573                                         if (strlcpy(tempname, *fileptr,
574                                             sizeof (tempname)) >=
575                                             sizeof (tempname)) {
576                                                 (void) fprintf(stderr,
577                                                     gettext("%s: filename "
578                                                     "too long\n"),
579                                                     *fileptr);
580                                                 perm_stat = 1;
581                                                 continue;
582                                         }

584                                         if (addDotZ(tempname,
585                                             sizeof (tempname)) < 0) {
586                                                 perm_stat = 1;
```



```
587                                                 continue;
588                                         }

590                                         *fileptr = tempname;
591                                 }

593                                 /* Open input file */

595                                 if (stat(*fileptr, &statbuf) < 0) {
596                                         perror(*fileptr);
597                                         perm_stat = 1;
598                                         continue;
599                                 }

601                                 if ((freopen(*fileptr, "r", inp)) == NULL) {
602                                         perror(*fileptr);
603                                         perm_stat = 1;
604                                         continue;
605                                 }
606                         }

608                         /* Check the magic number */

610                         if (nomagic == 0) {
611                                 if ((getc(inp) !=
612                                     (magic_header[0] & 0xFF)) ||
613                                     (getc(inp) !=
614                                     (magic_header[1] & 0xFF))) {
615                                         (void) fprintf(stderr, gettext(
616                                             "%s: not in compressed "
617                                             "format\n"),
618                                             *fileptr);
619                                         perm_stat = 1;
620                                         continue;
621                                 }

623                                 /* set -b from file */
624                                 if ((maxbits = getc(inp)) == EOF &&
625                                     ferror(inp)) {
626                                         perror(*fileptr);
627                                         perm_stat = 1;
628                                         continue;
629                                 }

631                                 block_compress = maxbits & BLOCK_MASK;
632                                 maxbits &= BIT_MASK;
633                                 maxmaxcode = 1 << maxbits;

635                                 if (maxbits > BITS) {
636                                         (void) fprintf(stderr,
637                                             gettext("%s: compressed "
638                                             "with %d bits, "
639                                             "can only handle"
640                                             " %d bits\n"),
641                                             *fileptr, maxbits, BITS);
642                                         perm_stat = 1;
643                                         continue;
644                                 }
645                         }

647                         if (!use_stdout) {
648                                 /* Generate output filename */

650                                 if (strlcpy(ofname, *fileptr,
651                                     sizeof (ofname)) >=
652                                     sizeof (ofname)) {
```

```
653                                     (void) fprintf(stderr,
654                                         gettext("%s: filename "
655                                         "too long\n"),
656                                         *fileptr);
657                                     perm_stat = 1;
658                                     continue;
659                             }

661                             /* Strip off .Z */

663                             ofname[strlen(*fileptr) - 2] = '\0';
664                     }
665             } else {
666                     /* COMPRESSION */

668                     if (strcmp(*fileptr, "-") == 0) {
669                             /* process stdin */
670                             inp = stdin;
671                             outp = stdout;
672                             use_stdout = 1;
673                             *fileptr = "stdin"; /* for error messages */

675                             /* Use the largest possible hash table */
676                             hsize =  HSIZE;
677                     } else {
678                             /* process the named file */

680                             inp = infile;
681                             outp = outfile;
682                             use_stdout = 0;

684                             if (zcat_flg) {
685                                     use_stdout = 1;
686                                     outp = stdout;
687                             }

689                             if (strcmp(*fileptr +
690                                 strlen(*fileptr) - 2, ".Z") == 0) {
691                                     (void) fprintf(stderr, gettext(
692                                         "%s: already has .Z "
693                                         "suffix -- no change\n"),
694                                         *fileptr);
695                                     perm_stat = 1;
696                                     continue;
697                             }
698                             /* Open input file */

700                             if (stat(*fileptr, &statbuf) < 0) {
701                                     perror(*fileptr);
702                                     perm_stat = 1;
703                                     continue;
704                             }

706                             if ((freopen(*fileptr, "r", inp)) == NULL) {
707                                     perror(*fileptr);
708                                     perm_stat = 1;
709                                     continue;
710                             }

712                             fsize = (off_t)statbuf.st_size;

714                             /*
715                              * tune hash table size for small
716                              * files -- ad hoc,
717                              * but the sizes match earlier #defines, which
718                              * serve as upper bounds on the number of
```



```
719                              * output codes.
720                              */
721                             hsize = HSIZE;
722                             if (fsize < (1 << 12))
723                                     hsize = min(5003, HSIZE);
724                             else if (fsize < (1 << 13))
725                                     hsize = min(9001, HSIZE);
726                             else if (fsize < (1 << 14))
727                                     hsize = min(18013, HSIZE);
728                             else if (fsize < (1 << 15))
729                                     hsize = min(35023, HSIZE);
730                             else if (fsize < 47000)
731                                     hsize = min(50021, HSIZE);

733                             if (!use_stdout) {
734                                     /* Generate output filename */

736                                     if (strlcpy(ofname, *fileptr,
737                                         sizeof (ofname)) >=
738                                         sizeof (ofname)) {
739                                             (void) fprintf(stderr,
740                                                 gettext("%s: filename "
741                                                 "too long\n"),
742                                                 *fileptr);
743                                             perm_stat = 1;
744                                             continue;
745                                     }

747                                     if (addDotZ(ofname,
748                                         sizeof (ofname)) < 0) {
749                                             perm_stat = 1;
750                                             continue;
751                                     }
752                             }
753                     }
754             }       /* if (do_decomp) */

756             /* Check for overwrite of existing file */

758             if (!overwrite && !use_stdout) {
759                     if (stat(ofname, &ostatbuf) == 0) {
760                             (void) fprintf(stderr, gettext(
761                                 "%s already exists;"), ofname);
762                             if (bgnd_flag == 0 && isatty(2)) {
763                                     int cin;

765                                     (void) fprintf(stderr, gettext(
766                                         " do you wish to overwr"
767                                         "ite %s (%s or %s)? "),
768                                         ofname, yesstr, nostr);
769                                     (void) fflush(stderr);
770                                     for (cin = 0; cin < LINE_MAX; cin++)
771                                             line[cin] = 0;
772                                     (void) read(2, line, LINE_MAX);

774                                     if (yes_check(line) == 0) {
775                                             (void) fprintf(stderr,
776                                                 gettext(
777                                                 "\tnot overwri"
778                                                 "tten\n"));
779                                             continue;
780                                     }
781                             } else {
782                                     /*
783                                      * XPG4: Assertion 1009
784                                      * Standard input is not
```

```
785                                      * terminal, and no '-f',
786                                      * and file exists.
787                                      */
788
789                                     (void) fprintf(stderr, gettext(
790                                         "%s: File exists, -f not"
791                                         " specified, and ru"
792                                         "nning in the backgro"
793                                         "und.\n"), *fileptr);
794                                     perm_stat = 1;
795                                     continue;
796                             }
797                     }
798             }
799             if (!use_stdout) {
800                     if ((pathconf(ofname, _PC_XATTR_EXISTS) == 1) ||
801                         (saflg && sysattr_support(ofname,
802                         _PC_SATTR_EXISTS) == 1)) {
803                             (void) unlink(ofname);
804                     }
805                     /* Open output file */
806                     if (freopen(ofname, "w", outp) == NULL) {
807                             perror(ofname);
808                             perm_stat = 1;
809                             continue;
810                     }
811                     precious = 0;
812                     if (!quiet) {
813                             (void) fprintf(stderr, "%s: ",
814                                 *fileptr);
815                             newline_needed = 1;
816                     }
817             } else if (!quiet && !do_decomp) {
818                     (void) fprintf(stderr, "%s: ",
819                         *fileptr);
820                     newline_needed = 1;
821             }
822
823             /* Actually do the compression/decompression */
824
825             if ((jmpval = setjmp(env)) == 0) {
826                     /* We'll see how things go */
827 #ifndef DEBUG
828                     if (do_decomp == 0)  {
829                             compress();
830                     } else {
831                             decompress();
832                     }
833 #else
834                     if (do_decomp == 0)  {
835                             compress();
836                     } else if (debug == 0)  {
837                             decompress();
838                     } else {
839                             printcodes();
840                     }
841
842                     if (verbose) {
843                             dump_tab();
844                     }
845 #endif
846             } else {
847                     /*
848                      * Things went badly - clean up and go on.
849                      * jmpval's values break down as follows:
850                      *   1 == message determined by ferror() values.
```



```
851                      *   2 == input problem message needed.
852                      *   3 == output problem message needed.
853                      */
854
855                     if (ferror(inp) || jmpval == 2) {
856                             if (do_decomp) {
857                                     (void) fprintf(stderr, gettext(
858                                         "uncompress: %s: corrupt"
859                                         " input\n"), *fileptr);
860                             } else {
861                                     perror(*fileptr);
862                             }
863                     }
864
865                     if (ferror(outp) || jmpval == 3) {
866                             /* handle output errors */
867
868                             if (use_stdout) {
869                                     perror("");
870                             } else {
871                                     perror(ofname);
872                             }
873                     }
874
875                     if (ofname[0] != '\0') {
876                             if (unlink(ofname) < 0)  {
877                                     perror(ofname);
878                             }
879
880                             ofname[0] = '\0';
881                     }
882
883                     perm_stat = 1;
884                     continue;
885             }
886
887             /* Things went well */
888
889             if (!use_stdout) {
890                     /* Copy stats */
891                     copystat(*fileptr, &statbuf, ofname);
892                     precious = 1;
893                     if (newline_needed) {
894                             (void) putc('\n', stderr);
895                     }
896                     /*
897                      * Print the info. for unchanged file
898                      * when no -v
899                      */
900
901                     if (didnt_shrink) {
902                             if (!force && perm_stat == 0) {
903                                     if (quiet) {
904                                             (void) fprintf(stderr, gettext(
905                                                 "%s: -- file "
906                                                 "unchanged\n"),
907                                                 *fileptr);
908                                     }
909
910                                     perm_stat = 2;
911                             }
912                     }
913             } else {
914                     if (didnt_shrink && !force && perm_stat == 0) {
915                             perm_stat = 2;
916                     }
```

```
 918                        if (newline_needed) {
 919                                (void) fprintf(stderr, "\n");
 920                        }
 921                }
 922        }       /* for */

 924        return (perm_stat);
 925 }
_____unchanged_portion_omitted_

1781 static void
1782 cl_hash(count_int hsize)               /* reset code table */
1783 {
1784        count_int *htab_p = htab+hsize;
1785        long i;
1786        long m1 = -1;

1788        i = hsize - 16;
1789        do {                            /* might use Sys V memset(3) here */
1790                *(htab_p-16) = m1;
1791                *(htab_p-15) = m1;
1792                *(htab_p-14) = m1;
1793                *(htab_p-13) = m1;
1794                *(htab_p-12) = m1;
1795                *(htab_p-11) = m1;
1796                *(htab_p-10) = m1;
1797                *(htab_p-9) = m1;
1798                *(htab_p-8) = m1;
1799                *(htab_p-7) = m1;
1800                *(htab_p-6) = m1;
1801                *(htab_p-5) = m1;
1802                *(htab_p-4) = m1;
1803                *(htab_p-3) = m1;
1804                *(htab_p-2) = m1;
1805                *(htab_p-1) = m1;
1806                htab_p -= 16;
1807        } while ((i -= 16) >= 0);

1809        for (i += 16; i > 0; i--)
1810                *--htab_p = m1;
1811 }
_____unchanged_portion_omitted_
```

```
**********************************************************
  142664 Thu Feb 28 11:26:02 2019
new/usr/src/cmd/dfs.cmds/sharemgr/commands.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg  Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
______unchanged_portion_omitted_
```

```
**********************************************************
    2642 Thu Feb 28 11:26:02 2019
new/usr/src/cmd/echo/echo.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
/*
 * 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) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <string.h>
#include <locale.h>

int
main(int argc, char *argv[])
{

        register char   *cp;
        register int    i, wd;
        int     j;
        wchar_t         wc;
        int             b_len;
        char            *ep;

        (void) setlocale(LC_ALL, "");

        if (--argc == 0) {
                (void) putchar('\n');
                if (fflush(stdout) != 0)
                        return (1);
                return (0);
        }

        for (i = 1; i <= argc; i++) {
                for (cp = argv[i], ep = cp + (int)strlen(cp);
```



```
                    cp < ep; cp += b_len) {
                        if ((b_len = mbtowc(&wc, cp, MB_CUR_MAX)) <= 0) {
                                (void) putchar(*cp);
                                b_len = 1;
                                continue;
                        }

                        if (wc != '\\') {
                                (void) putwchar(wc);
                                continue;
                        }

                        cp += b_len;
                        b_len = 1;
                        switch (*cp) {
                                case 'a':       /* alert - XCU4 */
                                        (void) putchar('\a');
                                        continue;

                                case 'b':
                                        (void) putchar('\b');
                                        continue;

                                case 'c':
                                        if (fflush(stdout) != 0)
                                                return (1);
                                        return (0);

                                case 'f':
                                        (void) putchar('\f');
                                        continue;

                                case 'n':
                                        (void) putchar('\n');
                                        continue;

                                case 'r':
                                        (void) putchar('\r');
                                        continue;

                                case 't':
                                        (void) putchar('\t');
                                        continue;

                                case 'v':
                                        (void) putchar('\v');
                                        continue;

                                case '\\':
                                        (void) putchar('\\');
                                        continue;
                                case '0':
                                        j = wd = 0;
                                        while ((*++cp >= '0' && *cp <= '7') &&
                                            j++ < 3) {
                                                wd <<= 3;
                                                wd |= (*cp - '0');
                                        }
                                        (void) putchar(wd);
                                        --cp;
                                        continue;

                                default:
                                        cp--;
                                        (void) putchar(*cp);
                        }
```

```
 126                 }
 127                 (void) putchar(i == argc? '\n': ' ');
 128                 if (fflush(stdout) != 0)
 129                         return (1);
 130         }
 131         return (0);
 132 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   21938 Thu Feb 28 11:26:02 2019
new/usr/src/cmd/fmthard/fmthard.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg  Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */


/*
 *
 *      Portions of this source code were provided by International
 *      Computers Limited (ICL) under a development agreement with AT&T.
 */

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

/*
 * Copyright (c) 2018, Joyent, Inc.
 */
/*
 * Sun Microsystems version of fmthard:
 *
 * Supports the following arguments:
 *
 *      -i              Writes VTOC to stdout, rather than disk
 *      -q              Quick check: exit code 0 if VTOC ok
 *      -d <data>       Incremental changes to the VTOC
 *      -n <vname>      Change volume name to <vname>
 *      -s <file>       Read VTOC information from <file>, or stdin ("-")
 *      -u <state>      Reboot after writing VTOC, according to <state>:
 *                              boot: AD_BOOT (standard reboot)
 *                              firm: AD_IBOOT (interactive reboot)
 *
 * Note that fmthard cannot write a VTOC on an unlabeled disk.
 * You must use format or SunInstall for this purpose.
 * (NOTE: the above restriction only applies on Sparc systems).
 *
 * The primary motivation for fmthard is to duplicate the
 * partitioning from disk to disk:
 *
```



```
 *      prtvtoc /dev/rdsk/c0t0d0s2 | fmthard -s - /dev/rdsk/c0t1d0s2
 */

#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/int_limits.h>
#include <sys/stat.h>
#include <sys/uadmin.h>
#include <sys/open.h>
#include <sys/vtoc.h>
#include <sys/dkio.h>
#include <sys/isa_defs.h>
#include <sys/efi_partition.h>

#if defined(_SUNOS_VTOC_16)
#include <sys/dklabel.h>
#endif

#include <sys/sysmacros.h>

#ifndef SECSIZE
#define SECSIZE                 DEV_BSIZE
#endif  /* SECSIZE */

/*
 * Internal functions.
 */
extern  int     main(int, char **);
static  void    display(struct dk_geom *, struct extvtoc *, char *);
static  void    display64(struct dk_gpt *,  char *);
static  void    insert(char *, struct extvtoc *);
static  void    insert64(char *, struct dk_gpt *);
static  void    load(FILE *, struct dk_geom *, struct extvtoc *);
static  void    load64(FILE *, int fd, struct dk_gpt **);
static  void    usage(void);
static  void    validate(struct dk_geom *, struct extvtoc *);
static  void    validate64(struct dk_gpt *);
static  int     vread(int, struct extvtoc *, char *);
static  void    vread64(int, struct dk_gpt **, char *);
static  void    vwrite(int, struct extvtoc *, char *);
static  void    vwrite64(int, struct dk_gpt *, char *);

/*
 * Static variables.
 */
static char     *delta;         /* Incremental update */
static short    eflag;          /* force write of an EFI label */
static short    iflag;          /* Prints VTOC w/o updating */
static short    qflag;          /* Check for a formatted disk */
static short    uflag;          /* Exit to firmware after writing  */
                                /* new vtoc and reboot. Used during */
                                /* installation of core floppies */
static diskaddr_t       lastlba = 0;    /* last LBA on 64-bit VTOC */

#if defined(sparc)
static char     *uboot = "boot";

#elif defined(i386)
/* use installgrub(1M) to install boot blocks */
static char *uboot = "";
```

```
126 #else
127 #error No platform defined.
128 #endif  /* various platform-specific definitions */

130 static char     *ufirm = "firm";
131 static int              sectsiz;
132 #if defined(_SUNOS_VTOC_16)
133 static struct extvtoc   disk_vtoc;
134 #endif  /* defined(_SUNOS_VTOC_16) */

136 int
137 main(int argc, char **argv)
138 {
139         int             fd;
140         int             c;
141         char            *dfile;
142         char            *vname;
143         struct stat     statbuf;
144 #if defined(_SUNOS_VTOC_8)
145         struct extvtoc  disk_vtoc;
146 #endif  /* defined(_SUNOS_VTOC_8) */
147         struct dk_gpt   *disk_efi;
148         struct dk_geom  disk_geom;
149         struct dk_minfo minfo;
150         int             n;


153         disk_efi = NULL;
154         dfile = NULL;
155         vname = NULL;
156 #if defined(sparc)
157         while ((c = getopt(argc, argv, "ed:u:in:qs:")) != EOF)

159 #elif defined(i386)
160         while ((c = getopt(argc, argv, "ed:u:in:qb:p:s:")) != EOF)

162 #else
163 #error No platform defined.
164 #endif
165                 switch (c) {
166 #if defined(i386)
167                 case 'p':
168                 case 'b':
169                         (void) fprintf(stderr,
170                             "fmthard: -p and -b no longer supported."
171                             " Use installgrub(1M) to install boot blocks\n");
172                         break;
173 #endif  /* defined(i386) */

175                 case 'd':
176                         delta = optarg;
177                         break;
178                 case 'e':
179                         ++eflag;
180                         break;
181                 case 'i':
182                         ++iflag;
183                         break;
184                 case 'n':
185                         vname = optarg;
186                         break;
187                 case 'q':
188                         ++qflag;
189                         break;
190                 case 's':
191                         dfile = optarg;
```



```
192                         break;
193                 case 'u':
194                         if (strcmp(uboot, optarg) == 0)
195                                 ++uflag;
196                         else if (strcmp(ufirm, optarg) == 0)
197                                 uflag = 2;

199                         break;
200                 default:
201                         usage();
202                 }


205         if (argc - optind != 1)
206                 usage();

208         if (stat(argv[optind], (struct stat *)&statbuf) == -1) {
209                 (void) fprintf(stderr,
210                     "fmthard:  Cannot stat device %s\n",
211                     argv[optind]);
212                 exit(1);
213         }

215         if ((statbuf.st_mode & S_IFMT) != S_IFCHR) {
216                 (void) fprintf(stderr,
217                     "fmthard:  %s must be a raw device.\n",
218                     argv[optind]);
219                 exit(1);
220         }

222         if ((fd = open(argv[optind], O_RDWR|O_NDELAY)) < 0) {
223                 (void) fprintf(stderr, "fmthard:  Cannot open device %s - %s\n",
224                     argv[optind], strerror(errno));
225                 exit(1);
226         }

228         if (ioctl(fd, DKIOCGMEDIAINFO, &minfo) == 0) {
229                 sectsiz = minfo.dki_lbsize;
230         }

232         if (sectsiz == 0) {
233                 sectsiz = SECSIZE;
234         }

236         /*
237          * Get the geometry information for this disk from the driver
238          */
239         if (!eflag && ioctl(fd, DKIOCGGEOM, &disk_geom)) {
240 #ifdef DEBUG
241                 perror("DKIOCGGEOM failed");
242 #endif /* DEBUG */
243                 if (errno == ENOTSUP) {
244                         /* disk has EFI labels */
245                         eflag++;
246                 } else {
247                         (void) fprintf(stderr,
248                             "%s: Cannot get disk geometry\n", argv[optind]);
249                         (void) close(fd);
250                         exit(1);
251                 }
252         }

254         /*
255          * Read the vtoc on the disk
256          */
257         if (!eflag) {
```

```
258                 if (vread(fd, &disk_vtoc, argv[optind]) == 1)
259                         eflag++;
260         }
261         if (eflag && ((dfile == NULL) || qflag)) {
262                 vread64(fd, &disk_efi, argv[optind]);
263         }

265         /*
266          * Quick check for valid disk: 0 if ok, 1 if not
267          */
268         if (qflag) {
269                 (void) close(fd);
270                 if (!eflag) {
271                         exit(disk_vtoc.v_sanity == VTOC_SANE ? 0 : 1);
272                 } else {
273                         exit(disk_efi->efi_version <= EFI_VERSION102 ? 0 : 1);
274                 }
275         }

277         /*
278          * Incremental changes to the VTOC
279          */
280         if (delta) {
281                 if (!eflag) {
282                         insert(delta, &disk_vtoc);
283                         validate(&disk_geom, &disk_vtoc);
284                         vwrite(fd, &disk_vtoc, argv[optind]);
285                 } else {
286                         insert64(delta, disk_efi);
287                         validate64(disk_efi);
288                         vwrite64(fd, disk_efi, argv[optind]);
289                 }
290                 (void) close(fd);
291                 exit(0);
292         }

294         if (!dfile && !vname)
295                 usage();

297         /*
298          * Read new VTOC from stdin or data file
299          */
300         if (dfile) {
301                 if (strcmp(dfile, "-") == 0) {
302                         if (!eflag)
303                                 load(stdin, &disk_geom, &disk_vtoc);
304                         else
305                                 load64(stdin, fd, &disk_efi);
306                 } else {
307                         FILE *fp;
308                         if ((fp = fopen(dfile, "r")) == NULL) {
309                                 (void) fprintf(stderr, "Cannot open file %s\n",
310                                     dfile);
311                                 (void) close(fd);
312                                 exit(1);
313                         }
314                         if (!eflag)
315                                 load(fp, &disk_geom, &disk_vtoc);
316                         else
317                                 load64(fp, fd, &disk_efi);
318                         (void) fclose(fp);
319                 }
320         }

322         /*
323          * Print the modified VTOC, rather than updating the disk
```



```
324          */
325         if (iflag) {
326                 if (!eflag)
327                         display(&disk_geom, &disk_vtoc, argv[optind]);
328                 else
329                         display64(disk_efi, argv[optind]);
330                 (void) close(fd);
331                 exit(0);
332         }

334         if (vname) {
335                 n = MIN(strlen(vname) + 1, LEN_DKL_VVOL);
336                 if (!eflag) {
337                         (void) memcpy(disk_vtoc.v_volume, vname, n);
338                 } else {
339                         for (c = 0; c < disk_efi->efi_nparts; c++) {
340                                 if (disk_efi->efi_parts[c].p_tag ==
341                                     V_RESERVED) {
342                                 (void) memcpy(&disk_efi->efi_parts[c].p_name,
343                                     vname, n);
344                                 }
345                         }
346                 }

348         }
349         /*
350          * Write the new VTOC on the disk
351          */
352         if (!eflag) {
353                 validate(&disk_geom, &disk_vtoc);
354                 vwrite(fd, &disk_vtoc, argv[optind]);
355         } else {
356                 validate64(disk_efi);
357                 vwrite64(fd, disk_efi, argv[optind]);
358         }

360         /*
361          * Shut system down after writing a new vtoc to disk
362          * This is used during installation of core floppies.
363          */
364         if (uflag == 1)
365                 (void) uadmin(A_REBOOT, AD_BOOT, 0);
366         else if (uflag == 2)
367                 (void) uadmin(A_REBOOT, AD_IBOOT, 0);

369         (void) printf("fmthard:  New volume table of contents now in place.\n");

371         return (0);
372 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   21471 Thu Feb 28 11:26:03 2019
new/usr/src/cmd/fruadm/fruadm.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg   Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * 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) 2014 Gary Mills
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright (c) 2018, Joyent, Inc.
 */

#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libintl.h>
#include <libfru.h>
#include <errno.h>
#include <math.h>
#include <alloca.h>
#include <assert.h>
#include <sys/systeminfo.h>

#define NUM_OF_SEGMENT  1
#define SEGMENT_NAME_SIZE       2

#define FD_SEGMENT_SIZE 2949

static char  *command, *customer_data = NULL, *frupath = NULL, **svcargv;

/* DataElement supported in the customer operation */
static  char    *cust_data_list[] = {"Customer_DataR"};

/* DataElement supported in the service operation */
static  char    *serv_data_list[] = {"InstallationR", "ECO_CurrentR"};

/* currently supported segment name */
static  char    *segment_name[] = {"FD"};

static int   found_frupath = 0, list_only = 0, recursive = 0,
```



```
    service_mode = 0, svcargc, update = 0;


static void
usage(void)
{
        (void) fprintf(stderr,
            gettext("Usage:  %s [ -l ] | [ [ -r ] frupath [ text ] ]\n"),
            command);
}
```

*unchanged_portion_omitted_*

```
**********************************************************
   32780 Thu Feb 28 11:26:03 2019
new/usr/src/cmd/getconf/getconf.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

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

/*
 * Copyright 1985, 1993 by Mortice Kern Systems Inc.  All rights reserved.
 *
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 * getconf -- POSIX.2 compatible utility to query configuration specific
 *            parameters.
 *         -- XPG4 support added June/93
 *
 *         -- XPG5 support added Dec/97
 *
 *         -- XPG6 support added May/2003
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <limits.h>
#include <locale.h>
#include <libintl.h>
#include <assert.h>

extern size_t confstr(int, char *, size_t);

static int aflag = 0;
```



```
#define INVAL_ARG       "getconf: Invalid argument (%s)\n"
#define INVAL_PATHARG   "getconf: Invalid argument (%s or %s)\n"

/*
 *  Notes:
 *  The sctab.value field is defined to be a long.
 *  There are some values that are "unsigned long"; these values
 *  can be stored in a "long" field but when output, must be printed
 *  as an unsigned value. Thus, these values must have UNSIGNED_VALUE bit
 *  set in sctab.flag field.
 *
 *  There are 2 different ways to indicate a symbol is undefined:
 *     1) sctab.flag = UNDEFINED
 *     2) or sctab.value = -1 (and if !UNDEFINED and !UNSIGNED_VALUE)
 *
 *  There are a group of symbols (e.g CHAR_MIN, INT_MAX, INT_MIN, LONG_BIT ...)
 *  which we may set to -1 if they are not pre-defined in a system header file.
 *  This is used to indicate that these symbols are "undefined".
 *  We are assuming that these symbols cannot reasonably have a value of -1
 *  if they were defined in a system header file.
 *  (Unless of course -1 can be used to indicate "undefined" for that symbol)
 */

static struct sctab {
        long value;
        char *name;
        enum { SELFCONF, SYSCONF, PATHCONF, CONFSTR } type;
        int flag;
/* bit fields for sctab.flag member */
#define NOFLAGS         0       /* no special indicators */
#define UNDEFINED       1       /* value is known undefined at compile time */
#define UNSIGNED_VALUE  2       /* value is an unsigned */
} sctab[] = {
_____unchanged_portion_omitted_

int
main(int argc, char **argv)
{
        register struct sctab *scp;
        int c;
        int exstat = 0;

        (void) setlocale(LC_ALL, "");
#if     !defined(TEXT_DOMAIN)   /* Should be defined by cc -D */
#define TEXT_DOMAIN "SYS_TEST"
#endif
        (void) textdomain(TEXT_DOMAIN);

        while ((c = getopt(argc, argv, "av:")) != -1)
                switch (c) {
                case 'a':
                        aflag = 1;
                        break;
                case 'v':
                        /*
                         * Unix98 adds the -v option to allow
                         * programming 'specifications' to be
                         * indicated, for present purposes
                         * the specification isn't really
                         * doing anything of value, so for
                         * the moment getopt just processes the
                         * option value and argv[1] is adjusted.
                         *
                         * At some later date we might want to
                         * do specification processing at this
                         * point.
```

```
923                          */
924                         break;
925                 case '?':
926                         return (usage());
927                 }
928         argc -= optind-1;
929         argv += optind-1;

931         if ((aflag && argc >= 2) || (!aflag && argc < 2))
932                 return (usage());
933         if (aflag) {

935 #define TabStop         8
936 #define RightColumn     32
937 #define DefPathName     "."

939                 /*
940                  * sort the table by the "name" field
941                  * so we print it in sorted order
942                  */
943                 qsort(&sctab[0], (sizeof (sctab) /
944                     sizeof (struct sctab)) - 1,
939         qsort(&sctab[0], (sizeof (sctab)/sizeof (struct sctab))-1,
945                     sizeof (struct sctab), namecmp);

947                 /*
948                  * print all the known symbols and their values
949                  */
950                 for (scp = &sctab[0]; scp->name != NULL; ++scp) {
951                         int stat;

953                         /*
954                          * create 2 columns:
955                          *   variable name in the left column,
956                          *   value in the right column.
957                          * The right column starts at a tab stop.
958                          */
959                         (void) printf("%s:\t", scp->name);

961                         c = strlen(scp->name) + 1;
962                         c = (c+TabStop) / TabStop, c *= TabStop;
963                         for (; c < RightColumn; c += TabStop)
964                                 (void) putchar('\t');

966                         /*
967                          * for pathconf() related variables that require
968                          * a pathname, use "."
969                          */
970                         stat = getconf(scp, scp->type == PATHCONF ? 3 : 2,
971                             scp->name, DefPathName);

973                         exstat |= stat;

975                         /*
976                          * if stat != 0, then an error message was already
977                          * printed in getconf(),
978                          * so don't need to print one here
979                          */
980                 }
981         } else {

983                 /*
984                  * find the name of the specified variable (argv[1])
985                  * and print its value.
986                  */
987                 for (scp = &sctab[0]; scp->name != NULL; ++scp)
```



```
988                         if (strcmp(argv[1], scp->name) == 0) {
989                                 exstat = getconf(scp, argc, argv[1], argv[2]);
990                                 break;
991                         }

993                 /*
994                  * if at last entry in table, then the user specified
995                  * variable is invalid
996                  */
997                 if (scp->name == NULL) {
998                         errno = EINVAL;
999                         (void) fprintf(stderr, gettext(INVAL_ARG), argv[1]);
1000                        return (1);
1001                }
1002        }
1003        return (exstat);
1004 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   28860 Thu Feb 28 11:26:03 2019
new/usr/src/cmd/infocmp/infocmp.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg   Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * 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 (c) 2019, Joyent, Inc.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"       /* SVr4.0 1.13  */

/*
    NAME
        infocmp - compare terminfo descriptions, or dump a terminfo
        description

    AUTHOR
        Tony Hansen, February 23, 1984.
*/

#include "curses.h"
#include "term.h"
#include "print.h"
#include <fcntl.h>
#include <stdlib.h>

/* externs from libcurses */
extern char *boolnames[];
extern char *boolcodes[];
extern char *boolfnames[];
extern char *numnames[];
extern char *numcodes[];
extern char *numfnames[];
extern char *strnames[];
extern char *strcodes[];
extern char *strfnames[];
```



```
extern char ttytype[];
extern int tgetflag();
extern int tgetnum();
extern char *tgetstr();

/* externs from libc */
extern void exit();
extern void qsort();
extern char *getenv();
extern int getopt();
extern int optind;
extern char *optarg;
extern char *strncpy(), *strcpy();
extern int strcmp(), strlen();

/* data structures for this program */

struct boolstruct {
    char *infoname;                     /* the terminfo capability name */
    char *capname;                      /* the termcap capability name */
    char *fullname;                     /* the long C variable name */
    char *secondname;                   /* the use= terminal w/ this value */
    char val;                           /* the value */
    char secondval;                     /* the value in the use= terminal */
    char changed;                       /* a use= terminal changed the value */
    char seenagain;                     /* a use= terminal had this entry */
    };

struct numstruct {
    char *infoname;                     /* ditto from above */
    char *capname;
    char *fullname;
    char *secondname;
    short val;
    short secondval;
    char changed;
    char seenagain;
    };

struct strstruct {
    char *infoname;                     /* ditto from above */
    char *capname;
    char *fullname;
    char *secondname;
    char *val;
    char *secondval;
    char changed;
    char seenagain;
    };

/* globals for this file */
char *progname;                 /* argv[0], the name of the program */
static struct boolstruct *ibool; /* array of char information */
static struct numstruct *num;   /* array of number information */
static struct strstruct *str;   /* array of string information */
static char *used;              /* usage statistics */
static int numbools;            /* how many booleans there are */
static int numnums;             /* how many numbers there are */
static int numstrs;             /* how many strings there are */
#define TTYLEN 255
static char *firstterm;         /* the name of the first terminal */
static char *savettytype;       /* the synonyms of the first terminal */
static char _savettytype[TTYLEN+1]; /* the place to save those names */
static int devnull;             /* open("/dev/null") for setupterm */
#define trace stderr            /* send trace messages to stderr */
```

```
124 /* options */
125 static int verbose = 0;         /* debugging printing level */
126 static int diff = 0;            /* produce diff listing, the default */
127 static int common = 0;          /* produce common listing */
128 static int neither = 0;         /* list caps in neither entry */
129 static int use = 0;             /* produce use= comparison listing */
130 static enum printtypes printing /* doing any of above printing at all */
131         = pr_none;
132 enum { none, by_database, by_terminfo, by_longnames, by_cap }
133     sortorder = none;           /* sort the fields for printing */
134 static char *term1info, *term2info;     /* $TERMINFO settings */
135 static int Aflag = 0, Bflag = 0;        /* $TERMINFO was set with -A/-B */

137 #define EQUAL(s1, s2)   (((s1 == NULL) && (s2 == NULL)) || \
138                         ((s1 != NULL) && (s2 != NULL) && \
139                         (strcmp(s1, s2) == 0)))

141 static void sortnames();
142 int numcompare(const void *, const void *);
143 int boolcompare(const void *, const void *);
144 int strcompare(const void *, const void *);
145 static void check_nth_terminal(char *, int);

147 void
148 badmalloc()
149 {
150         (void) fprintf(stderr, "%s: malloc is out of space!\n", progname);
151         exit(-1);
152 }

154 /*
155     Allocate and initialize the global data structures and variables.
156 */
157 void
158 allocvariables(int argc, int firstoptind)
159 {
160         register int i, nullseen;

162         /* find out how many names we are dealing with */
163         for (numbools = 0; boolnames[numbools]; numbools++)
164                 ;
165         for (numnums = 0; numnames[numnums]; numnums++)
166                 ;
167         for (numstrs = 0; strnames[numstrs]; numstrs++)
168                 ;

170         if (verbose) {
171                 (void) fprintf(trace, "There are %d boolean capabilities.\n",
172                     numbools);
173                 (void) fprintf(trace, "There are %d numeric capabilities.\n",
174                     numnums);
175                 (void) fprintf(trace, "There are %d string capabilities.\n",
176                     numstrs);
177         }

179         /* Allocate storage for the names and their values */
180         ibool = (struct boolstruct  *) malloc((unsigned) numbools *
181             sizeof (struct boolstruct));
182         num = (struct numstruct *) malloc((unsigned) numnums *
183             sizeof (struct numstruct));
184         str = (struct strstruct *) malloc((unsigned) numstrs *
185             sizeof (struct strstruct));

187         /* Allocate array to keep track of which names have been used. */
188         if (use) {
188         if (use)
```



```
189                 used = (char *) malloc((unsigned) (argc - firstoptind) *
190                     sizeof (char));
191         }

193         if ((ibool == NULL) || (num == NULL) || (str == NULL) ||
194             (use && (used == NULL)))
195                 badmalloc();

197         /* Fill in the names and initialize the structures. */
198         nullseen = FALSE;
199         for (i = 0; i < numbools; i++) {
200                 ibool[i].infoname = boolnames[i];
201                 ibool[i].capname = boolcodes[i];
202                 /* This is necessary until fnames.c is */
203                 /* incorporated into standard curses. */
204                 if (nullseen || (boolfnames[i] == NULL)) {
205                         ibool[i].fullname = "unknown_boolean";
206                         nullseen = TRUE;
207                 } else {
206                 } else
208                         ibool[i].fullname = boolfnames[i];
209                 }
210                 ibool[i].changed = FALSE;
211                 ibool[i].seenagain = FALSE;
212         }
213         nullseen = 0;
214         for (i = 0; i < numnums; i++) {
215                 num[i].infoname = numnames[i];
216                 num[i].capname = numcodes[i];
217                 if (nullseen || (numfnames[i] == NULL)) {
218                         ibool[i].fullname = "unknown_number";
219                         nullseen = TRUE;
220                 } else {
218                 } else
221                         num[i].fullname = numfnames[i];
222                 }
223                 num[i].changed = FALSE;
224                 num[i].seenagain = FALSE;
225         }
226         nullseen = 0;
227         for (i = 0; i < numstrs; i++) {
228                 str[i].infoname = strnames[i];
229                 str[i].capname = strcodes[i];
230                 if (nullseen || (strfnames[i] == NULL)) {
231                         str[i].fullname = "unknown_string";
232                         nullseen = TRUE;
233                 } else {
230                 } else
234                         str[i].fullname = strfnames[i];
235                 }
236                 str[i].changed = FALSE;
237                 str[i].seenagain = FALSE;
238         }
239 }
```

*unchanged_portion_omitted_*

```
521 /*
522         Set up the first terminal and save the values from it.
523 */
524 void
525 initfirstterm(char *term)
526 {
527         register int i;

529         if (verbose) {
525         if (verbose)
```

```
530                 (void) fprintf(trace, "setting up terminal type '%s'.\n",
531                     term);
532         }

534         (void) setupterm(term, devnull, (int *) 0);

536         /* Save the name for later use. */
537         if (use) {
538                 register unsigned int length;
539                 savettytype = _savettytype;
540                 if ((length = strlen(ttytype)) >= TTYLEN) {
541                         savettytype = malloc(length);
542                         if (savettytype == NULL) {
543                                 (void) fprintf(stderr, "%s: malloc is out "
544                                     "of space\n", progname);
545                                 (void) strncpy(_savettytype, ttytype,
546                                     TTYLEN-1);
547                                 _savettytype[TTYLEN] = '\0';
548                                 savettytype = _savettytype;
549                         }
550                 } else {
545                 } else
551                         (void) strcpy(_savettytype, ttytype);
552                 }
553         }

555         if (printing != pr_none) {
556                 pr_heading(term, ttytype);
557                 pr_bheading();
558         }

560         /* Save the values for the first terminal. */
561         for (i = 0; i < numbools; i++) {
562                 if ((ibool[i].val = tgetflag(ibool[i].capname)) &&
563                     printing != pr_none) {
557                     printing != pr_none)
564                         pr_boolean(ibool[i].infoname, ibool[i].capname,
565                             ibool[i].fullname, 1);
566                 }

568                 if (verbose) {
560                 if (verbose)
569                         (void) fprintf(trace, "%s=%d.\n", ibool[i].infoname,
570                             ibool[i].val);
571                 }
572         }

574         if (printing != pr_none) {
575                 if (printing == pr_cap)
576                         pr_bcaps();
577                 pr_bfooting();
578                 pr_nheading();
579         }

581         for (i = 0; i < numnums; i++) {
582                 if (((num[i].val = tgetnum(num[i].capname)) > -1) &&
583                     printing != pr_none) {
574                     printing != pr_none)
584                         pr_number(num[i].infoname, num[i].capname,
585                             num[i].fullname, num[i].val);
586                 }

588                 if (verbose) {
577                 if (verbose)
589                         (void) fprintf(trace, "%s=%d.\n", num[i].infoname,
590                             num[i].val);
```



```
591                 }
592         }

594         if (printing != pr_none) {
595                 if (printing == pr_cap)
596                         pr_ncaps();
597                 pr_nfooting();
598                 pr_sheading();
599         }

601         for (i = 0; i < numstrs; i++) {
602                 str[i].val = tgetstr(str[i].capname, (char **)0);
603                 if ((str[i].val != NULL) && printing != pr_none) {
591                 if ((str[i].val != NULL) && printing != pr_none)
604                         pr_string(str[i].infoname, str[i].capname,
605                             str[i].fullname, str[i].val);
606                 }

608                 if (verbose) {
609                         (void) fprintf(trace, "%s='", str[i].infoname);
610                         PR(trace, str[i].val);
611                         (void) fprintf(trace, "'.\n");
612                 }
613         }

615         if (printing == pr_cap)
616                 pr_scaps();

618         if (printing != pr_none)
619                 pr_sfooting();
620 }

622 /*
623         Set up the n'th terminal.
624 */
625 static void
626 check_nth_terminal(char *nterm, int n)
627 {
628         register char boolval;
629         register short numval;
630         register char *strval;
631         register int i;

633         if (use)
634                 used[n] = FALSE;

636         if (verbose) {
622         if (verbose)
637                 (void) fprintf(trace, "adding in terminal type '%s'.\n",
638                     nterm);
639         }

641         (void) setupterm(nterm, devnull, (int *) 0);

643         if (printing != pr_none) {
644                 pr_heading(nterm, ttytype);
645                 pr_bheading();
646         }

648         if (diff || common || neither) {
649                 if (Aflag && Bflag)
650                         (void) printf("comparing %s (TERMINFO=%s) to %s "
651                             "(TERMINFO=%s).\n",
652                         firstterm, term1info, nterm, term2info);
653                 else if (Aflag)
654                         (void) printf("comparing %s (TERMINFO=%s) to %s.\n",
```

```
655                             firstterm, term1info, nterm);
656                 else if (Bflag)
657                         (void) printf("comparing %s to %s (TERMINFO=%s).\n",
658                             firstterm, nterm, term2info);
659                 else
660                         (void) printf("comparing %s to %s.\n",
661                             firstterm, nterm);
662                 (void) printf("    comparing booleans.\n");
663         }

665         /* save away the values for the nth terminal */
666         for (i = 0; i < numbools; i++) {
667                 boolval = tgetflag(ibool[i].capname);
668                 if (use) {
669                         if (ibool[i].seenagain) {
670                         /*
671                         ** We do not have to worry about this impossible case
672                         ** since booleans can have only two values: true and
673                         ** false.
674                         ** if (boolval && (boolval != ibool[i].secondval))
675                         **  {
676                         **  (void) fprintf(trace, "use= order dependency"
677                         **  "found:\n");
678                         **  (void) fprintf(trace, "    %s: %s has %d, %s has"
679                         **   " %d.\n",
680                         **      ibool[i].capname, ibool[i].secondname,
681                         **      ibool[i].secondval, nterm, boolval);
682                         **  }
683                         */
684                         } else {
685                                 if (boolval == TRUE) {
686                                         ibool[i].seenagain = TRUE;
687                                         ibool[i].secondval = boolval;
688                                         ibool[i].secondname = nterm;
689                                         if (ibool[i].val != boolval)
690                                                 ibool[i].changed = TRUE;
691                                         else
692                                                 used[n] = TRUE;
693                                 }
694                         }
695                 }
696                 if (boolval) {
697                         if (printing != pr_none) {
682                         if (printing != pr_none)
698                                 pr_boolean(ibool[i].infoname, ibool[i].capname,
699                                     ibool[i].fullname, 1);
700                         }

702                         if (common && (ibool[i].val == boolval))
703                                 (void) printf("\t%s= T.\n", ibool[i].infoname);
704                 } else if (neither && !ibool[i].val) {
687                 } else if (neither && !ibool[i].val)
705                         (void) printf("\t!%s.\n", ibool[i].infoname);
706                 }
707                 if (diff && (ibool[i].val != boolval))
708                         (void) printf("\t%s: %c:%c.\n", ibool[i].infoname,
709                             ibool[i].val?'T':'F', boolval?'T':'F');
710                 if (verbose) {
692                 if (verbose)
711                         (void) fprintf(trace, "%s: %d:%d, changed=%d, "
712                             "seen=%d.\n", ibool[i].infoname, ibool[i].val,
713                             boolval, ibool[i].changed, ibool[i].seenagain);
714                 }
715         }

717         if (printing != pr_none) {
```



```
718                 if (printing == pr_cap)
719                         pr_bcaps();
720                 pr_bfooting();
721                 pr_nheading();
722         }

724         if (diff || common || neither)
725                 (void) printf("    comparing numbers.\n");

727         for (i = 0; i < numnums; i++) {
728                 numval = tgetnum(num[i].capname);
729                 if (use) {
730                         if (num[i].seenagain) {
731                                 if ((numval > -1) &&
732                                     (numval != num[i].secondval)) {
733                                         (void) fprintf(stderr,
734                                             "%s: use = order dependency "
735                                             "found:\n", progname);
736                                         (void) fprintf(stderr, "    %s: %s "
737                                             "has %d, %s has %d.\n",
738                                             num[i].capname, num[i].secondname,
739                                             num[i].secondval, nterm, numval);
740                                 }
741                         } else {
742                                 if (numval > -1) {
743                                         num[i].seenagain = TRUE;
744                                         num[i].secondval = numval;
745                                         num[i].secondname = nterm;
746                                         if ((numval > -1) &&
747                                             (num[i].val != numval))
748                                                 num[i].changed = TRUE;
749                                         else
750                                                 used[n] = TRUE;
751                                 }
752                         }
753                 }
754                 if (numval > -1) {
755                         if (printing != pr_none) {
736                         if (printing != pr_none)
756                                 pr_number(num[i].infoname, num[i].capname,
757                                     num[i].fullname, numval);
758                         }

760                         if (common && (num[i].val == numval)) {
739                         if (common && (num[i].val == numval))
761                                 (void) printf("\t%s= %d.\n", num[i].infoname,
762                                     numval);
763                         }

765                 } else if (neither && (num[i].val == -1)) {
742                         } else if (neither && (num[i].val == -1))
766                         (void) printf("\t!%s.\n", num[i].infoname);
767                 }

769                 if (diff && (num[i].val != numval)) {
744                         if (diff && (num[i].val != numval))
770                         (void) printf("\t%s: %d:%d.\n",
771                             num[i].infoname, num[i].val, numval);
772                 }

774                 if (verbose) {
747                         if (verbose)
775                         (void) fprintf(trace, "%s: %d:%d, "
776                             "changed = %d, seen = %d.\n",
777                             num[i].infoname, num[i].val, numval,
778                             num[i].changed, num[i].seenagain);
```

```
779                 }
780         }

782         if (printing != pr_none) {
783                 if (printing == pr_cap)
784                         pr_ncaps();
785                 pr_nfooting();
786                 pr_sheading();
787         }

789         if (diff || common || neither)
790                 (void) printf("    comparing strings.\n");

792         for (i = 0; i < numstrs; i++) {
793                 strval = tgetstr(str[i].capname, (char **)0);
794                 if (use) {
795                         if (str[i].seenagain && (strval != NULL)) {
796                                 if (!EQUAL(strval, str[i].secondval)) {
797                                         (void) fprintf(stderr,
798                                             "use= order dependency"
799                                             "  found:\n");
800                                         (void) fprintf(stderr,
801                                             "    %s: %s has '",
802                                             str[i].capname, str[i].secondname);
803                                         PR(stderr, str[i].secondval);
804                                         (void) fprintf(stderr,
805                                             "', %s has '", nterm);
806                                         PR(stderr, strval);
807                                         (void) fprintf(stderr, "'.\n");
808                                 }
809                         } else {
810                                 if (strval != NULL) {
811                                         str[i].seenagain = TRUE;
812                                         str[i].secondval = strval;
813                                         str[i].secondname = nterm;
814                                         if (!EQUAL(str[i].val, strval))
815                                                 str[i].changed = TRUE;
816                                         else
817                                                 used[n] = TRUE;
818                                 }
819                         }
820                 }
821                 if (strval != NULL) {
822                         if (printing != pr_none) {
794                         if (printing != pr_none)
823                                 pr_string(str[i].infoname, str[i].capname,
824                                     str[i].fullname, strval);
825                         }

827                         if (common && EQUAL(str[i].val, strval)) {
828                                 (void) printf("\t%s= '", str[i].infoname);
829                                 PR(stdout, strval);
830                                 (void) printf("'.\n");
831                         }
832                 } else if (neither && (str[i].val == NULL))
833                         (void) printf("\t!%s.\n", str[i].infoname);
834                 if (diff && !EQUAL(str[i].val, strval)) {
835                         (void) printf("\t%s: '", str[i].infoname);
836                         PR(stdout, str[i].val);
837                         (void) printf("','");
838                         PR(stdout, strval);
839                         (void) printf("'.\n");
840                 }
841                 if (verbose) {
842                         (void) fprintf(trace, "%s: '", str[i].infoname);
843                         PR(trace, str[i].val);
```



```
844                         (void) fprintf(trace, "':'");
845                         PR(trace, strval);
846                         (void) fprintf(trace, "',changed=%d,seen=%d.\n",
847                             str[i].changed, str[i].seenagain);
848                 }
849         }

851         if (printing == pr_cap)
852                 pr_scaps();

854         if (printing != pr_none)
855                 pr_sfooting();

857         return;
858 }

860 /*
861         A capability gets an at-sign if it no longer exists, but
862         one of the relative entries contains a value for it.
863         It gets printed if the original value is not seen in ANY
864         of the relative entries, or if the FIRST relative entry that has
865         the capability gives a DIFFERENT value for the capability.
866 */
867 void
868 dorelative(int firstoptind, int argc, char **argv)
869 {
870         register int i;

872         /* turn off printing of termcap and long names */
873         pr_init(pr_terminfo);

875         /* print out the entry name */
876         pr_heading((char *)0, savettytype);

878         pr_bheading();

880         /* Print out all bools that are different. */
881         for (i = 0; i < numbools; i++) {
882                 if (!ibool[i].val && ibool[i].changed) {
851         for (i = 0; i < numbools; i++)
852                 if (!ibool[i].val && ibool[i].changed)
883                         pr_boolean(ibool[i].infoname, (char *)0,
884                             (char *)0, -1);
885                 } else if (ibool[i].val && (ibool[i].changed ||
886                     !ibool[i].seenagain)) {
855                 else if (ibool[i].val && (ibool[i].changed ||
856                     !ibool[i].seenagain))
887                         pr_boolean(ibool[i].infoname, (char *)0, (char *)0, 1);
888                 }
889         }

891         pr_bfooting();
892         pr_nheading();

894         /* Print out all nums that are different. */
895         for (i = 0; i < numnums; i++) {
896                 if (num[i].val < 0 && num[i].changed) {
863         for (i = 0; i < numnums; i++)
864                 if (num[i].val < 0 && num[i].changed)
897                         pr_number(num[i].infoname, (char *)0, (char *)0, -1);
898                 } else if (num[i].val >= 0 && (num[i].changed ||
899                     !num[i].seenagain)) {
866                 else if (num[i].val >= 0 && (num[i].changed ||
867                     !num[i].seenagain))
900                         pr_number(num[i].infoname, (char *)0,
901                             (char *)0, num[i].val);
```

```
902                 }
903         }

905         pr_nfooting();
906         pr_sheading();

908         /* Print out all strs that are different. */
909         for (i = 0; i < numstrs; i++) {
910                 if (str[i].val == NULL && str[i].changed) {
875         for (i = 0; i < numstrs; i++)
876                 if (str[i].val == NULL && str[i].changed)
911                         pr_string(str[i].infoname, (char *)0, (char *)0,
912                             (char *)0);
913                 } else if ((str[i].val != NULL) &&
914                     (str[i].changed || !str[i].seenagain)) {
915                         pr_string(str[i].infoname,
916                             (char *)0, (char *)0, str[i].val);
917                 }
918         }
879                 else if ((str[i].val != NULL) &&
880                     (str[i].changed || !str[i].seenagain))
881         pr_string(str[i].infoname, (char *)0, (char *)0, str[i].val);

920         pr_sfooting();

922         /* Finish it up. */
923         for (i = firstoptind; i < argc; i++) {
924                 if (used[i - firstoptind]) {
886         for (i = firstoptind; i < argc; i++)
887                 if (used[i - firstoptind])
925                         (void) printf("\tuse=%s,\n", argv[i]);
926                 } else {
889                 else
927                         (void) fprintf(stderr,
928                             "%s: 'use=%s' did not add anything to the "
929                             "description.\n", progname, argv[i]);
930                 }
931         }
932 }

934 void
935 local_setenv(char *termNinfo)
936 {
937         extern char **environ;
938         static char *newenviron[2] = { 0, 0 };
939         static unsigned int termsize = BUFSIZ;
940         static char _terminfo[BUFSIZ];
941         static char *terminfo = &_terminfo[0];
942         register int termlen;

944         if (termNinfo && *termNinfo) {
945                 if (verbose) {
906                 if (verbose)
946                         (void) fprintf(trace, "setting TERMINFO=%s.\n",
947                             termNinfo);
948                 }

950                 termlen = strlen(termNinfo);
951                 if (termlen + 10 > termsize) {
952                         termsize = termlen + 20;
953                         terminfo = (char *) malloc(termsize * sizeof (char));
954                 }
955                 if (terminfo == (char *) NULL)
956                         badmalloc();
957                 (void) sprintf(terminfo, "TERMINFO=%s", termNinfo);
958                 newenviron[0] = terminfo;
```



```
959         } else
960                 newenviron[0] = (char *) 0;
961         environ = newenviron;
962 }

964 int
965 main(int argc, char **argv)
966 {
967         int i, c, firstoptind;
968         char *tempargv[2];
969         char *term = getenv("TERM");

971         term1info = term2info = getenv("TERMINFO");
972         progname = argv[0];

974         /* parse options */
975         while ((c = getopt(argc, argv, "ducnILCvV1rw:s:A:B:")) != EOF)
976                 switch (c) {
977                         case 'v':       verbose++;
978                                         break;
979                         case '1':       pr_onecolumn(1);
980                                         break;
981                         case 'w':       pr_width(atoi(optarg));
982                                         break;
983                         case 'd':       diff++;
984                                         break;
985                         case 'c':       common++;
986                                         break;
987                         case 'n':       neither++;
988                                         break;
989                         case 'u':       use++;
990                                         break;
991                         case 'L':       pr_init(printing = pr_longnames);
992                                         break;
993                         case 'I':       pr_init(printing = pr_terminfo);
994                                         break;
995                         case 'C':       pr_init(printing = pr_cap);
996                                         break;
997                         case 'A':       term1info = optarg; Aflag++;
998                                         break;
999                         case 'B':       term2info = optarg; Bflag++;
1000                                        break;
1001                        case 'r':       pr_caprestrict(0);
1002                                        break;
1003                        case 's':
1004                                if (strcmp(optarg, "d") == 0)
1005                                        sortorder = by_database;
1006                                else if (strcmp(optarg, "i") == 0)
1007                                        sortorder = by_terminfo;
1008                                else if (strcmp(optarg, "l") == 0)
1009                                        sortorder = by_longnames;
1010                                else if (strcmp(optarg, "c") == 0)
1011                                        sortorder = by_cap;
1012                                else
1013                                        goto usage;
1014                                break;
1015                        case 'V':
1016                                (void) printf("%s: version %s\n", progname,
1017                                    "@(#)curses:screen/infocmp.c        1.13");
1018                                exit(0);
1019                        case '?':
1020                                usage:
1021                                (void) fprintf(stderr,
1022                                    "usage: %s [-ducn] [-ILC] [-1Vv] "
1023                                    "[-s d|i|l|c] [-A directory] "
1024                                    "[-B directory] term-names ...\n",
```

```
1025                                    progname);
1026                                (void) fprintf(stderr, "\t-d\tprint "
1027                                    "differences (the default for >1 "
1028                                    "term-name)\n");
1029                                (void) fprintf(stderr, "\t-u\tproduce "
1030                                    "relative description\n");
1031                                (void) fprintf(stderr, "\t-c\tprint common "
1032                                    "entries\n");
1033                                (void) fprintf(stderr, "\t-n\tprint entries "
1034                                    "in neither\n");
1035                                (void) fprintf(stderr, "\t-I\tprint terminfo "
1036                                    "entries (the default for 1 term-name)\n");
1037                                (void) fprintf(stderr, "\t-C\tprint termcap "
1038                                    "entries\n");
1039                                (void) fprintf(stderr, "\t-L\tprint long C "
1040                                    "variable names\n");
1041                                (void) fprintf(stderr, "\t-1\tsingle column "
1042                                    "output\n");
1043                                (void) fprintf(stderr, "\t-V\tprint program "
1044                                    "version\n");
1045                                (void) fprintf(stderr, "\t-v\tverbose "
1046                                    "debugging output\n");
1047                                (void) fprintf(stderr, "\t-s\tchange sort "
1048                                    "order\n");
1049                                (void) fprintf(stderr, "\t-A\tset $TERMINFO "
1050                                    "for first term-name\n");
1051                                (void) fprintf(stderr, "\t-B\tset $TERMINFO "
1052                                    "for other term-names\n");
1053                                exit(-1);
1054                        }

1056                argc -= optind;
1057                argv += optind;
1058                optind = 0;

1060                /* Default to $TERM for -n, -I, -C and -L options. */
1061                /* This is done by faking argv[][], argc and optind. */
1062                if (neither && (argc == 0 || argc == 1)) {
1063                        if (argc == 0)
1064                                tempargv[0] = term;
1065                        else
1066                                tempargv[0] = argv[optind];
1067                        tempargv[1] = term;
1068                        argc = 2;
1069                        argv = tempargv;
1070                        optind = 0;
1071                } else if ((printing != pr_none) && (argc == 0)) {
1072                        tempargv[0] = term;
1073                        argc = 1;
1074                        argv = tempargv;
1075                        optind = 0;
1076                }

1078                /* Check for enough names. */
1079                if ((use || diff || common) && (argc <= 1)) {
1080                        (void) fprintf(stderr,
1081                            "%s: must have at least two terminal names for a "
1082                            "comparison to be done.\n", progname);
1083                        goto usage;
1084                }

1086                /* Set the default of diff -d or print -I */
1087                if (!use && (printing == pr_none) && !common && !neither) {
1088                        if (argc == 0 || argc == 1) {
1089                                if (argc == 0) {
1090                                        tempargv[0] = term;
```



```
1091                                        argc = 1;
1092                                        argv = tempargv;
1093                                        optind = 0;
1094                                }
1095                                pr_init(printing = pr_terminfo);
1096                        } else {
1055                        } else
1097                                diff++;
1098                        }
1099                }

1101                /* Set the default sorting order. */
1102                if (sortorder == none) {
1060                if (sortorder == none)
1103                        switch ((int) printing) {
1104                                case (int) pr_cap:
1105                                        sortorder = by_cap; break;
1106                                case (int) pr_longnames:
1107                                        sortorder = by_longnames; break;
1108                                case (int) pr_terminfo:
1109                                case (int) pr_none:
1110                                        sortorder = by_terminfo; break;
1111                        }
1112                }

1114                firstterm = argv[optind++];
1115                firstoptind = optind;

1117                allocvariables(argc, firstoptind);
1118                sortnames();

1120                devnull = open("/dev/null", O_RDWR);
1121                local_setenv(term1info);
1122                initfirstterm(firstterm);
1123                local_setenv(term2info);
1124                for (i = 0; optind < argc; optind++, i++)
1125                        check_nth_terminal(argv[optind], i);

1127                if (use)
1128                        dorelative(firstoptind, argc, argv);

1130                return (0);
1131 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    9962 Thu Feb 28 11:26:03 2019
new/usr/src/cmd/mdb/common/modules/genunix/ldi.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * 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 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/dditypes.h>
#include <sys/ddi_impldefs.h>
#include <sys/ddipropdefs.h>
#include <sys/modctl.h>
#include <sys/file.h>
#include <sys/sunldi_impl.h>

#include <mdb/mdb_modapi.h>
#include <mdb/mdb_ks.h>

#include "ldi.h"

/*
 * ldi handle walker structure
 */
typedef struct lh_walk {
        struct ldi_handle       **hash; /* current bucket pointer       */
        struct ldi_handle       *lhp;   /* ldi handle pointer           */
        size_t                  index;  /* hash table index             */
        struct ldi_handle       buf;    /* buffer used for handle reads */
} lh_walk_t;
```

*_____unchanged_portion_omitted_*

```
int
ldi_handle(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
{
        int                     ident = 0;
```



```
        int                     refs = 1;

        if (mdb_getopts(argc, argv,
            'i', MDB_OPT_SETBITS, TRUE, &ident) != argc)
                return (DCMD_USAGE);

        if (ident)
                refs = 0;

        /* Determine if there is an ldi handle address */
        if (!(flags & DCMD_ADDRSPEC)) {
                if (mdb_walk_dcmd("ldi_handle", "ldi_handle",
                    argc, argv) == -1) {
                        mdb_warn("can't walk ldi handles");
                        return (DCMD_ERR);
                } return (DCMD_OK);
                }
                return (DCMD_OK);
        }

        /* display the header line */
        if (DCMD_HDRSPEC(flags))
                ldi_handle_header(refs, ident);

        /* display the ldi handle */
        if (ldi_handle_print(addr, ident, refs))
                return (DCMD_ERR);

        return (DCMD_OK);
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   18753 Thu Feb 28 11:26:04 2019
new/usr/src/cmd/ndmpadm/ndmpadm_main.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg   Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 * BSD 3 Clause License
 *
 * Copyright (c) 2007, The Storage Networking Industry Association.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *      - Redistributions of source code must retain the above copyright
 *        notice, this list of conditions and the following disclaimer.
 *
 *      - Redistributions in binary form must reproduce the above copyright
 *        notice, this list of conditions and the following disclaimer in
 *        the documentation and/or other materials provided with the
 *        distribution.
 *
 *      - Neither the name of The Storage Networking Industry Association (SNIA)
 *        nor the names of its contributors may be used to endorse or promote
 *        products derived from this software without specific prior written
 *        permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
#include <assert.h>
#include <ctype.h>
#include <libgen.h>
#include <libintl.h>
#include <locale.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <door.h>
#include <sys/mman.h>
#include <libndmp.h>
#include "ndmpadm.h"

typedef enum {
        HELP_GET_CONFIG,
        HELP_SET_CONFIG,
```



```
        HELP_SHOW_DEVICES,
        HELP_SHOW_SESSIONS,
        HELP_KILL_SESSIONS,
        HELP_ENABLE_AUTH,
        HELP_DISABLE_AUTH
} ndmp_help_t;
______unchanged_portion_omitted_
```

```
**********************************************************
    7440 Thu Feb 28 11:26:04 2019
new/usr/src/cmd/pathchk/pathchk.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
/*
 * 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) 1994 by Sun Microsystems, Inc.
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 * Copyright 1991, 1992 by Mortice Kern Systems Inc.  All rights reserved.
 *
 * Standards Conformance :
 *      P1003.2/D11.2
 *
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"
/*
 * Original ident string for reference
 * ident        "$Id: pathchk.c,v 1.29 1994/05/24 15:51:19 mark Exp $"
 */

#include <locale.h>
#include <libintl.h>
#include <limits.h>
#include <sys/stat.h>
#include <fcntl.h>              /* for creat() prototype */
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <unistd.h>
#include <stdlib.h>

/*
 * These are the characters in the portable filename character set defined
 * in POSIX P1003.2.
 */
static  char    portfsset[] = \
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789._-";
```



```
#ifndef M_FSDELIM
#define M_FSDELIM(c)    ((c) == '/')
#endif

static char *nametoolong = "%s: component too long.\n";
static char *pathtoolong = "%s: pathname too long.\n";
static char *notsrch = "%s: Not searchable.\n";
static char *badchar = "%s: Nonportable character '%c' (%#02X) found.\n";
static char *badbyte = "%s: Nonportable byte %#02X found.\n";

static char *pathconfprob = "pathchk: warning: \
                            pathconf(\"%s\", %s) returns '%s'. Using %s = %d\n";


static int printWarnings = 1;

static int checkpathname(char *, int);
static void usage(void);

/*
 * mainline for pathchk
 */
int
main(int argc, char **argv)
{
        int c;
        int errors;
        int pflag = 0;

        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)
#define TEXT_DOMAIN "SYS_TEST"
#endif
        (void) textdomain(TEXT_DOMAIN);


        while ((c = getopt(argc, argv, "pw")) != EOF) {
                switch (c) {
                case 'p':
                        pflag = 1;
                        break;

                case 'w':
                        /* turn off warning messages */
                        printWarnings = 0;
                        break;

                default:
                        usage();
                }
        }

        argv += optind;

        if (*argv == 0) {
                usage();
                /* NOTREACHED */
        }

        errors = 0;
        while (*argv) {
                errors += checkpathname(*argv, pflag);
                argv += 1;
        }
```

```
 125         return (errors);
 126 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   37389 Thu Feb 28 11:26:04 2019
new/usr/src/cmd/pcitool/pcitool_ui.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg   Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * 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) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Milan Jurik. All rights reserved.
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 * This is the user interface module for the pcitool.  It checks commandline
 * arguments and options and stores them in a pcitool_uiargs_t structure passed
 * back to the rest of the program for processing.
 *
 * Please see pcitool_usage.c for a complete commandline description.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/inttypes.h>
#include <sys/types.h>
#include <sys/param.h>
#include <strings.h>
#include <errno.h>
#include <sys/pci.h>

#include <sys/pci_tools.h>

#include "pcitool_ui.h"

/*
 * Uncomment the following for useful debugging / development options for this
 * module only.
 */

/* #define      DEBUG   1               */
/* #define      STANDALONE      1       */

#define         DEVNAME_START_PCI       "/pci"
#define         DEVNAME_START_NIU       "/niu"
```



```
/* Default read/write size when -s not specified. */
#define DEFAULT_SIZE    4

/* For get_value64 */
#define HEX_ONLY        B_TRUE
#define BASE_BY_PREFIX  B_FALSE

#define BITS_PER_BYTE   8

/*
 * This defines which main options can be specified by the user.
 * Options with colons after them require arguments.
 */
static char *opt_string = ":n:d:i:m:p:rw:o:s:e:b:vaqlcxgy";

/* This defines options used singly and only by themselves (no nexus). */
static char *no_dev_opt_string = "ahpqv";

static void print_bad_option(char *argv[], int optopt, char *optarg);
static boolean_t get_confirmation(void);
static int get_value64(char *value_str, uint64_t *value, boolean_t hex_only);
static int parse_nexus_opts(char *input, uint64_t *flags_arg, uint8_t *bank_arg,
    uint64_t *base_addr_arg);
static int extract_bdf_arg(char *cvalue, char *fld, uint64_t fld_flag,
    uint64_t *all_flags, uint8_t *ivalue);
static int extract_bdf(char *value, char **bvalue_p, char **dvalue_p,
    char **fvalue_p);
static int parse_device_opts(char *input, uint64_t *flags_arg,
    uint8_t *bus_arg, uint8_t *device_arg, uint8_t *func_arg,
    uint8_t *bank_arg);
static int parse_ino_opts(char *input, uint64_t *flags_arg,
    uint32_t *cpu_arg, uint8_t *ino_arg);
static int parse_msi_opts(char *input, uint64_t *flags_arg, uint16_t *msi_arg);
static int parse_intr_set_opts(char *input, uint64_t *flags_arg,
    uint32_t *cpu_arg);
static int parse_probeone_opts(char *input, uint64_t *flags_arg,
    uint8_t *bus_arg, uint8_t *device_arg, uint8_t *func_arg);

#ifdef DEBUG
void dump_struct(pcitool_uiargs_t *dump_this);
#endif

/* Exported functions. */

/*
 * Main commandline argument parsing routine.
 *
 * Takes argc and argv straight from the commandline.
 * Returns a pcitool_uiargs_t with flags of options specified, and values
 * associated with them.
 */
int
get_commandline_args(int argc, char *argv[], pcitool_uiargs_t *parsed_args)
{
        int c;                          /* Current option being processed. */
        boolean_t error = B_FALSE;
        boolean_t confirm = B_FALSE;
        uint64_t recv64;

        /* Needed for getopt(3C) */
        extern char *optarg;    /* Current commandline string. */
        extern int optind;      /* Index of current commandline string. */
        extern int optopt;      /* Option (char) which is missing an operand. */
        extern int opterr;      /* Set to 0 to disable getopt err reporting. */

        opterr = 0;
```

```
127         bzero(parsed_args, sizeof (pcitool_uiargs_t));

129         /* No args.  probe mode accounting for bus ranges, nonverbose. */
130         if (argc == 1) {
131                 usage(argv[0]);
132                 parsed_args->flags = 0;
133                 return (SUCCESS);
134         }

136         /* 1st arg is not a device name. */
137         if ((strstr(argv[1], DEVNAME_START_PCI) != argv[1]) &&
138             (strstr(argv[1], DEVNAME_START_NIU) != argv[1])) {

140                 /* Default is to probe all trees accounting for bus ranges. */
141                 parsed_args->flags = PROBEALL_FLAG | PROBERNG_FLAG;

143                 /* Loop thru the options until complete or an error is found. */
144                 while (((c = getopt(argc, argv, no_dev_opt_string)) != -1) &&
145                     (error == B_FALSE)) {

147                         switch (c) {

149                         /* Help requested. */
150                         case 'h':
151                                 usage(argv[0]);
152                                 parsed_args->flags = 0;
153                                 return (SUCCESS);

155                         case 'p':
156                                 /* Take default probe mode */
157                                 break;

159                         case 'a':
160                                 /*
161                                  * Enable display of ALL bus numbers.
162                                  *
163                                  * This takes precidence over PROBERNG as -a
164                                  * is explicitly specified.
165                                  */
166                                 parsed_args->flags &= ~PROBERNG_FLAG;
167                                 break;

169                         case 'q':
170                                 parsed_args->flags |= QUIET_FLAG;
171                                 break;

173                         /* Verbose mode for full probe. */
174                         case 'v':
175                                 parsed_args->flags |= VERBOSE_FLAG;
176                                 break;

178                         default:
179                                 error = B_TRUE;
180                                 break;
181                         }
182                 }

184                 /* Check for values straggling at the end of the command. */
185                 if (optind != argc) {
186                         (void) fprintf(stderr, "%s: Unrecognized parameter "
187                             "at the end of the command.\n", argv[0]);
188                         error = B_TRUE;
189                 }

191                 if (error) {
```



```
192                         print_bad_option(argv, optopt, optarg);
193                         return (FAILURE);
194                 }

196                 return (SUCCESS);
197         }

199         /* Device node specified on commandline. */

201         /* Skip argv[1] before continuing below. */
202         optind++;

204         /* Loop through the options until complete or an error is found. */
205         while (((c = getopt(argc, argv, opt_string)) != -1) &&
206             (error == B_FALSE)) {

208                 switch (c) {

210                 /* Nexus */
211                 case 'n':
212                         if (parsed_args->flags & (LEAF_FLAG |
213                             NEXUS_FLAG | INTR_FLAG | PROBE_FLAGS)) {
214                                 (void) fprintf(stderr, "%s: -n set with "
215                                     "-d, -p or -i or is set twice\n", argv[0]);
216                                 error = B_TRUE;
217                                 break;
218                         }
219                         parsed_args->flags |= NEXUS_FLAG;
220                         if (parse_nexus_opts(optarg, &parsed_args->flags,
221                             &parsed_args->bank, &parsed_args->base_address) !=
222                             SUCCESS) {
223                                 (void) fprintf(stderr,
224                                     "%s: Error parsing -n options\n", argv[0]);
225                                 error = B_TRUE;
226                                 break;
227                         }
228                         break;

230                 /* Device (leaf node) */
231                 case 'd':
232                         if (parsed_args->flags & (LEAF_FLAG |
233                             NEXUS_FLAG | INTR_FLAG | PROBE_FLAGS)) {
234                                 (void) fprintf(stderr, "%s: -d set with "
235                                     "-n, -p or -i or is set twice\n", argv[0]);
236                                 error = B_TRUE;
237                                 break;
238                         }
239                         parsed_args->flags |= LEAF_FLAG;
240                         if (parse_device_opts(optarg, &parsed_args->flags,
241                             &parsed_args->bus, &parsed_args->device,
242                             &parsed_args->function,
243                             &parsed_args->bank) != SUCCESS) {
244                                 (void) fprintf(stderr,
245                                     "%s: Error parsing -d options\n", argv[0]);
246                                 error = B_TRUE;
247                                 break;
248                         }
249                         break;

251                 /* Interrupt */
252                 case 'i':
253                         if (parsed_args->flags & (LEAF_FLAG |
254                             NEXUS_FLAG | INTR_FLAG | PROBE_FLAGS)) {
255                                 (void) fprintf(stderr, "%s: -i set with -m, "
256                                     "-n, -d or -p or is set twice\n", argv[0]);
257                                 error = B_TRUE;
```

```
258                             break;
259                     }
260                     parsed_args->flags |= INTR_FLAG;

262                     /* parse input to get ino value. */
263                     if (parse_ino_opts(optarg, &parsed_args->flags,
264                         &parsed_args->old_cpu,
265                         &parsed_args->intr_ino) != SUCCESS) {
266                             (void) fprintf(stderr,
267                                 "%s: Error parsing interrupt options\n",
268                                 argv[0]);
269                             error = B_TRUE;
270                     }
271                     break;
272             /* Interrupt */
273             case 'm':
274                     if (parsed_args->flags & (LEAF_FLAG |
275                         NEXUS_FLAG | INTR_FLAG | PROBE_FLAGS)) {
276                             (void) fprintf(stderr, "%s: -m set with -i, "
277                                 "-n, -d or -p or is set twice\n", argv[0]);
278                             error = B_TRUE;
279                             break;
280                     }
281                     parsed_args->flags |= INTR_FLAG;

283                     /* parse input to get msi value. */
284                     if (parse_msi_opts(optarg, &parsed_args->flags,
285                         &parsed_args->intr_msi) != SUCCESS) {
286                             (void) fprintf(stderr,
287                                 "%s: Error parsing interrupt options\n",
288                                 argv[0]);
289                             error = B_TRUE;
290                     }
291                     break;
292             /* Probe */
293             case 'p':
294                     if (parsed_args->flags & (LEAF_FLAG |
295                         NEXUS_FLAG | INTR_FLAG | PROBE_FLAGS)) {
296                             (void) fprintf(stderr, "%s: -p set with "
297                                 "-n, -d or -i or is set twice\n", argv[0]);
298                             error = B_TRUE;
299                             break;
300                     }

302                     /* Process -p with no dedicated options to it. */
303                     if (optarg[0] == '-') {
304                             optind--;

306                             /* Probe given tree observing ranges */
307                             parsed_args->flags |=
308                                 (PROBETREE_FLAG | PROBERNG_FLAG);
309                             continue;
310                     }

312                     /* parse input to get ino value. */
313                     if (parse_probeone_opts(optarg, &parsed_args->flags,
314                         &parsed_args->bus, &parsed_args->device,
315                         &parsed_args->function) != SUCCESS) {
316                             (void) fprintf(stderr,
317                                 "%s: Error parsing probe options\n",
318                                 argv[0]);
319                             error = B_TRUE;
320                     } else {
321                             /*
322                              * parse_probeone_opts found options to
323                              * set up bdf.
```



```
324                              */
325                             parsed_args->flags |= PROBEDEV_FLAG;
326                     }
327                     break;

329             /* Probe all busses */
330             case 'a':
331                     /* Must follow -p, and -p must have no bdf. */
332                     if (!(parsed_args->flags & PROBETREE_FLAG)) {
333                             error = B_TRUE;
334                             break;
335                     }

337                     parsed_args->flags &= ~PROBERNG_FLAG;
338                     break;

340             /* Read */
341             case 'r':
342                     if (!(parsed_args->flags &
343                         (LEAF_FLAG | NEXUS_FLAG | INTR_FLAG))) {
344                             error = B_TRUE;
345                             break;
346                     }

348                     /*
349                      * Allow read and write to be set together for now,
350                      * since this means write then read back for device and
351                      * nexus accesses.  Check for this and disallow with
352                      * interrupt command later.
353                      */
354                     parsed_args->flags |= READ_FLAG;
355                     break;

357             /* Write */
358             case 'w':
359                     if (!(parsed_args->flags &
360                         (LEAF_FLAG | NEXUS_FLAG | INTR_FLAG))) {
361                             error = B_TRUE;
362                             break;
363                     }
364                     if (parsed_args->flags & WRITE_FLAG) {
365                             (void) fprintf(stderr, "%s: -w set twice\n",
366                                 argv[0]);
367                             error = B_TRUE;
368                             break;
369                     }

371                     /*
372                      * For device and nexus, get a single register value
373                      * to write.
374                      */
375                     if (parsed_args->flags & (NEXUS_FLAG | LEAF_FLAG)) {
376                             parsed_args->flags |= WRITE_FLAG;
377                             if (get_value64(optarg,
378                                 &parsed_args->write_value, HEX_ONLY) !=
379                                 SUCCESS) {
380                                     (void) fprintf(stderr,
381                                         "%s: Error reading value to "
382                                         "write.\n", argv[0]);
383                                     error = B_TRUE;
384                                     break;
385                             }

387                     /* For interrupt,  parse input to get cpu value. */
388                     } else if (parsed_args->flags & INTR_FLAG) {
389                             parsed_args->flags |= WRITE_FLAG;
```

```
390                                 if (parse_intr_set_opts(optarg,
391                                     &parsed_args->flags,
392                                     &parsed_args->intr_cpu) != SUCCESS) {
393                                         (void) fprintf(stderr, "%s: Error "
394                                             "parsing interrupt options.\n",
395                                             argv[0]);
396                                         error = B_TRUE;
397                                         break;
398                                 }

400                         } else {
401                                 error = B_TRUE;
402                                 break;
403                         }
404                         break;

406                 /* Offset */
407                 case 'o':
408                         if (!(parsed_args->flags & (LEAF_FLAG | NEXUS_FLAG))) {
409                                 error = B_TRUE;
410                                 break;
411                         }
412                         if (parsed_args->flags & OFFSET_FLAG) {
413                                 (void) fprintf(stderr, "%s: -o set twice\n",
414                                     argv[0]);
415                                 error = B_TRUE;
416                                 break;
417                         }
418                         parsed_args->flags |= OFFSET_FLAG;
419                         if (get_value64(optarg, &recv64, HEX_ONLY) != SUCCESS) {
420                                 (void) fprintf(stderr,
421                                     "%s: Error in offset argument\n", argv[0]);
422                                 error = B_TRUE;
423                                 break;
424                         }
425                         parsed_args->offset = (uint32_t)recv64;
426                         if (parsed_args->offset != recv64) {
427                                 (void) fprintf(stderr, "%s: Offset argument "
428                                     "too large for 32 bits\n", argv[0]);
429                                 error = B_TRUE;
430                                 break;
431                         }
432                         break;

434                 /* Size */
435                 case 's':
436                         if (!(parsed_args->flags & (LEAF_FLAG | NEXUS_FLAG))) {
437                                 error = B_TRUE;
438                                 break;
439                         }
440                         if (parsed_args->flags & SIZE_FLAG) {
441                                 (void) fprintf(stderr, "%s: -s set twice\n",
442                                     argv[0]);
443                                 error = B_TRUE;
444                                 break;
445                         }
446                         parsed_args->flags |= SIZE_FLAG;
447                         if (get_value64(optarg, &recv64, HEX_ONLY) != SUCCESS) {
448                                 (void) fprintf(stderr,
449                                     "%s: Error in size argument\n", argv[0]);
450                                 error = B_TRUE;
451                                 break;
452                         }
453                         switch (recv64) {
454                         case 1:
455                         case 2:
```



```
456                         case 4:
457                         case 8:
458                                 break;
459                         default:
460                                 error = B_TRUE;
461                                 (void) fprintf(stderr,
462                                     "%s: Error in size argument\n", argv[0]);
463                                 break;
464                         }
465                         parsed_args->size |= (uint8_t)recv64;
466                         break;

468                 /* Endian. */
469                 case 'e':
470                         if (!(parsed_args->flags & (LEAF_FLAG | NEXUS_FLAG))) {
471                                 error = B_TRUE;
472                                 break;
473                         }
474                         if (parsed_args->flags & ENDIAN_FLAG) {
475                                 (void) fprintf(stderr, "%s: -e set twice\n",
476                                     argv[0]);
477                                 error = B_TRUE;
478                                 break;
479                         }
480                         parsed_args->flags |= ENDIAN_FLAG;

482                         /* Only a single character allowed. */
483                         if (optarg[1] != '\0') {
484                                 (void) fprintf(stderr,
485                                     "%s: Error in endian argument\n", argv[0]);
486                                 error = B_TRUE;
487                                 break;
488                         }

490                         switch (optarg[0]) {
491                         case 'b':
492                                 parsed_args->big_endian = B_TRUE;
493                                 break;
494                         case 'l':
495                                 break;
496                         default:
497                                 (void) fprintf(stderr,
498                                     "%s: Error in endian argument\n", argv[0]);
499                                 error = B_TRUE;
500                                 break;
501                         }
502                         break;

504                 /* (Byte)dump */
505                 case 'b':
506                         if (!(parsed_args->flags & (LEAF_FLAG | NEXUS_FLAG))) {
507                                 error = B_TRUE;
508                                 break;
509                         }
510                         if (parsed_args->flags & BYTEDUMP_FLAG) {
511                                 (void) fprintf(stderr, "%s: -b set twice\n",
512                                     argv[0]);
513                                 error = B_TRUE;
514                                 break;
515                         }
516                         parsed_args->flags |= BYTEDUMP_FLAG;
517                         if (get_value64(optarg, &recv64, HEX_ONLY) != SUCCESS) {
518                                 (void) fprintf(stderr, "%s: Error in "
519                                     "bytedump argument\n", argv[0]);
520                                 error = B_TRUE;
521                                 break;
```

```
522                         }
523                         parsed_args->bytedump_amt = (uint32_t)recv64;
524                         if (parsed_args->bytedump_amt != recv64) {
525                                 (void) fprintf(stderr, "%s: Bytedump amount "
526                                     "too large for 32 bits\n", argv[0]);
527                                 error = B_TRUE;
528                                 break;
529                         }
530                         break;

532                 /* Verbose. */
533                 case 'v':
534                         parsed_args->flags |= VERBOSE_FLAG;
535                         break;

537                 /*
538                  * Quiet - no errors reported as messages.
539                  * (Status still returned by program, however.)
540                  */
541                 case 'q':
542                         parsed_args->flags |= QUIET_FLAG;
543                         break;

545                 /* Loop. */
546                 case 'l':
547                         parsed_args->flags |= LOOP_FLAG;
548                         break;

550                 /*
551                  * Dump characters with bytedump (-b).
552                  * Show controller info with -i.
553                  */
554                 case 'c':
555                         if (parsed_args->flags & BYTEDUMP_FLAG) {
556                                 parsed_args->flags |= CHARDUMP_FLAG;

558                         } else if (parsed_args->flags & INTR_FLAG) {
559                                 parsed_args->flags |= SHOWCTLR_FLAG;

561                         } else {
562                                 error = B_TRUE;
563                         }
564                         break;

566                 /* Continue on errors with bytedump (-b). */
567                 case 'x':
568                         if (!(parsed_args->flags & BYTEDUMP_FLAG)) {
569                                 error = B_TRUE;
570                                 break;
571                         }
572                         parsed_args->flags |= ERRCONT_FLAG;
573                         break;

575                 case 'g':
576                         if (!(parsed_args->flags & INTR_FLAG)) {
577                                 error = B_TRUE;
578                                 break;
579                         }
580                         parsed_args->flags |= SETGRP_FLAG;
581                         break;

583                 /* Take -y as confirmation and don't ask (where applicable). */
584                 case 'y':
585                         confirm = B_TRUE;
586                         break;
```



```
588                 /* Option without operand. */
589                 case ':':
590                         switch (optopt) {
591                         case 'p':
592                                 /* Allow -p without bdf spec. */
593                                 parsed_args->flags |=
594                                     (PROBETREE_FLAG | PROBERNG_FLAG);
595                                 break;
596                         default:
597                                 error = B_TRUE;
598                                 break;
599                         }
600                         break;

602                 /* Unrecognized option. */
603                 case '?':
604                         error = B_TRUE;
605                         break;
606                 }
607         }

609         /*
610          * Commandline has been parsed.  Check for errors which can be checked
611          * only after commandline parsing is complete.
612          */

614         if (!error) {

616                 /* Check for values straggling at the end of the command. */
617                 if (optind != argc) {
618                         (void) fprintf(stderr, "%s: Unrecognized parameter "
619                             "at the end of the command.\n", argv[0]);
620                         print_bad_option(argv, optopt, optarg);
621                         return (FAILURE);
622                 }

624                 /* No args other than nexus.  Default to probing that nexus */
625                 if (!(parsed_args->flags &
626                     (LEAF_FLAG | NEXUS_FLAG | INTR_FLAG | PROBE_FLAGS))) {
627                         usage(argv[0]);
628                         parsed_args->flags = 0;
629                         return (SUCCESS);
630                 }

632                 /*
633                  * Don't allow any options other than all-bus, verbose or
634                  * quiet with probe command.  Set default probe flags if nexus
635                  * or leaf options are not specified.
636                  */
637                 if (parsed_args->flags & (PROBETREE_FLAG | PROBEALL_FLAG)) {
638                         if (parsed_args->flags &
639                             ~(PROBE_FLAGS | QUIET_FLAG | VERBOSE_FLAG))
640                                 error = B_TRUE;
641                 }

643                 /*
644                  * Allow only read, write, quiet and verbose flags for
645                  * interrupt command.  Note that INO_SPEC_FLAG and CPU_SPEC_FLAG
646                  * get set for interrupt command.
647                  */
648                 if (parsed_args->flags & INTR_FLAG) {
649                         if (parsed_args->flags &
650                             ~(INTR_FLAG | VERBOSE_FLAG | QUIET_FLAG |
651                             READ_FLAG | WRITE_FLAG | SHOWCTLR_FLAG |
652                             SETGRP_FLAG | INO_ALL_FLAG | INO_SPEC_FLAG |
653                             MSI_ALL_FLAG | MSI_SPEC_FLAG | CPU_SPEC_FLAG)) {
```

```
654                             (void) fprintf(stderr, "%s: -v, -q, -r, -w, -c "
655                                 "-g are only options allowed with "
656                                 "interrupt command.\n", argv[0]);
657                             error = B_TRUE;
658                     }
659
660                     /* Need cpu and ino values for interrupt set command. */
661                     if ((parsed_args->flags & WRITE_FLAG) &&
662                         !(parsed_args->flags & CPU_SPEC_FLAG) &&
663                         !((parsed_args->flags & INO_SPEC_FLAG) ||
664                         (parsed_args->flags & MSI_SPEC_FLAG))) {
665                             (void) fprintf(stderr,
666                                 "%s: Both cpu and ino/msi must be "
667                                 "specified explicitly for interrupt "
668                                 "set command.\n", argv[0]);
669                             error = B_TRUE;
670                     }
671
672                     /* Intr write and show ctlr flags are incompatible. */
673                     if ((parsed_args->flags &
674                         (WRITE_FLAG + SHOWCTLR_FLAG)) ==
675                         (WRITE_FLAG + SHOWCTLR_FLAG)) {
676                             (void) fprintf(stderr,
677                                 "%s: -w and -c are incompatible for "
678                                 "interrupt command.\n", argv[0]);
679                             error = B_TRUE;
680                     }
681
682                     /* Intr setgrp flag valid only for intr writes. */
683                     if ((parsed_args->flags & (WRITE_FLAG + SETGRP_FLAG)) ==
684                         SETGRP_FLAG) {
685                             (void) fprintf(stderr,
686                                 "%s: -g is incompatible with -r "
687                                 "for interrupt command.\n", argv[0]);
688                             error = B_TRUE;
689                     }
690
691                     /*
692                      * Disallow read & write together in interrupt command.
693                      */
694                     if ((parsed_args->flags & (WRITE_FLAG | READ_FLAG)) ==
695                         (WRITE_FLAG | READ_FLAG)) {
696                             (void) fprintf(stderr, "%s: Only one of -r and "
697                                 "-w can be specified in "
698                                 "interrupt command.\n", argv[0]);
699                             error = B_TRUE;
700                     }
701             }
702
703             /* Bytedump incompatible with some other options. */
704             if ((parsed_args->flags & BYTEDUMP_FLAG) &&
705                 (parsed_args->flags &
706                 (WRITE_FLAG | PROBE_FLAGS | INTR_FLAG))) {
707                     (void) fprintf(stderr,
708                         "%s: -b is incompatible with "
709                         "another specified option.\n", argv[0]);
710                     error = B_TRUE;
711             }
712
713             if (parsed_args->flags & (LEAF_FLAG | NEXUS_FLAG)) {
714
715                     if (!(parsed_args->flags & SIZE_FLAG)) {
716                             parsed_args->size = DEFAULT_SIZE;
717                     }
718                     if ((parsed_args->flags & WRITE_FLAG) &&
719                         parsed_args->size < sizeof (uint64_t) &&
```



```
720                             (parsed_args->write_value >>
721                             (parsed_args->size * BITS_PER_BYTE))) {
722                                 (void) fprintf(stderr,
723                                     "%s: Data to write is larger than "
724                                     "specified size.\n", argv[0]);
725                                 error = B_TRUE;
726                         }
727
728                 } else { /* Looping is compatible only with register cmds. */
729
730                         if (parsed_args->flags & LOOP_FLAG) {
731                                 (void) fprintf(stderr, "%s: -l is incompatible "
732                                     "with given command.\n", argv[0]);
733                                 error = B_TRUE;
734                         }
735                 }
736
737                 /* Call out an erroneous -y and then ignore it. */
738                 if ((confirm) && (!(parsed_args->flags & BASE_SPEC_FLAG))) {
739                                 (void) fprintf(stderr,
740                                     "%s: -y is incompatible with given command."
741                                     "  Ignoring.\n", argv[0]);
742                 }
743         }
744
745         /* Now fill in the defaults and other holes. */
746         if (!(error)) {
747                 if (!(parsed_args->flags & (READ_FLAG | WRITE_FLAG))) {
748                         parsed_args->flags |= READ_FLAG;
749                 }
750
751                 if (parsed_args->flags & (LEAF_FLAG | NEXUS_FLAG)) {
752                         if (!(parsed_args->flags & ENDIAN_FLAG)) {
753                                 parsed_args->big_endian = B_FALSE;
754                         }
755                 }
756
757                 if (parsed_args->flags & BASE_SPEC_FLAG) {
758                         if (!confirm) {
759                                 confirm = get_confirmation();
760                         }
761                         if (!confirm) {
762                                 parsed_args->flags &= ~ALL_COMMANDS;
763                         }
764                 }
765
766                 /*
767                  * As far as other defaults are concerned:
768                  *   Other fields: bus, device, function, offset, default to
769                  *   zero.
770                  */
771
772         } else {        /* An error occurred. */
773
774                 print_bad_option(argv, optopt, optarg);
775         }
776         return (error);
777 }
_____unchanged_portion_omitted_
```

```
**********************************************************
   38501 Thu Feb 28 11:26:04 2019
new/usr/src/cmd/pg/pg.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
/*
 * 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) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2016 by Delphix. All rights reserved.
 * Copyright (c) 2018, Joyent, Inc.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

#include <signal.h>
#include <setjmp.h>
#include <sys/types.h>
#include <sys/dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <ctype.h>
#include <stdio.h>
#include <wchar.h>
#include <curses.h>
#include <term.h>
#include <errno.h>
#include <stdlib.h>
#include <regexpr.h>
#include <limits.h>
#include <locale.h>
#include <wctype.h> /* iswprint() */
#include <string.h>
#include <unistd.h>
#include <wait.h>
#include <libw.h>
#include <regexpr.h>


/*
 *      pg -- paginator for crt terminals
 *
 *      Includes the ability to display pages that have
 *      already passed by. Also gives the user the ability
```



```
 *      to search forward and backwards for regular expressions.
 *      This works for piped input by copying to a temporary file,
 *      and resolving backreferences from there.
 *
 *      Note:   The reason that there are so many commands to do
 *              the same types of things is to try to accommodate
 *              users of other paginators.
 */

#define LINSIZ  1024
#define QUIT    '\034'
#define BOF     (EOF - 1)       /* Begining of File */
#define STOP    (EOF - 2)
#define PROMPTSIZE      256

/*
 * Function definitions
 */
static  void    lineset(int);
static  char    *setprompt();
static  int     set_state(int *, wchar_t, char *);
static  void    help();
static  void    copy_file(FILE *, FILE *);
static  void    re_error(int);
static  void    save_input(FILE *);
static  void    save_pipe();
static  void    newdol(FILE *);
static  void    erase_line(int);
static  void    kill_line();
static  void    doclear();
static  void    sopr(char *, int);
static  void    prompt(char *);
static  void    error(char *);
static  void    terminit();
static  void    compact();
static  off_t   getaline(FILE *);
static  int     mrdchar();
static  off_t   find(int, off_t);
static  int     search(char *, off_t);
static  FILE    *checkf(char *);
static  int     skipf(int);
static  int     readch();
static  int     ttyin();
static  int     number();
static  int     command(char *);
static  int     screen(char *);
static  int     fgetputc();
static  char    *pg_strchr();


struct line {                   /* how line addresses are stored */
        off_t   l_addr;         /* file offset */
        off_t   l_no;           /* line number in file */
};
______unchanged_portion_omitted_
```

```
**********************************************************
   18858 Thu Feb 28 11:26:04 2019
new/usr/src/cmd/sbdadm/sbdadm.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg   Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * 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 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2012 Milan Jurik. All rights reserved.
 * Copyright (c) 2018, Joyent, Inc.
 */
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <libintl.h>
#include <errno.h>
#include <string.h>
#include <assert.h>
#include <getopt.h>
#include <strings.h>
#include <ctype.h>
#include <libnvpair.h>
#include <locale.h>

#include <cmdparse.h>
#include <sys/stmf_defines.h>
#include <libstmf.h>
#include <sys/stmf_sbd_ioctl.h>

#define MAX_LU_LIST     8192
#define LU_LIST_MAX_RETRIES 3
#define GUID_INPUT  32

#define VERSION_STRING_MAJOR        "1"
#define VERSION_STRING_MINOR        "0"
#define VERSION_STRING_MAX_LEN      10


char *cmdName;

static char *getExecBasename(char *);
```



```
int delete_lu(int argc, char *argv[], cmdOptions_t *options,
    void *callData);
int create_lu(int argc, char *argv[], cmdOptions_t *options, void *callData);
int list_lus(int argc, char *argv[], cmdOptions_t *options, void *callData);
int modify_lu(int argc, char *argv[], cmdOptions_t *options, void *callData);
int import_lu(int argc, char *argv[], cmdOptions_t *options, void *callData);
static int callModify(char *, stmfGuid *, uint32_t, const char *, const char *);
int print_lu_attr(stmfGuid *);
void print_guid(uint8_t *g, FILE *f);
void print_attr_header();

optionTbl_t options[] = {
        { "disk-size", required_argument, 's',
                        "Size with <none>/k/m/g/t/p/e modifier" },
        { "keep-views", no_arg, 'k',
                        "Dont delete view entries related to the LU" },
        { NULL, 0, 0 }
};
```

*unchanged_portion_omitted_*

```
**********************************************************
    7885 Thu Feb 28 11:26:05 2019
new/usr/src/cmd/sgs/ar/common/main.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

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

  24 /*
  25  * Copyright (c) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
  26  */

  28 /*
  29  * Copyright (c) 2018, Joyent, Inc.
  30  */

  32 #include "inc.h"
  33 #include "conv.h"

  35 /*
  36  * Forward declarations
  37  */
  38 static void setup(int, char **, Cmd_info *);
  39 static void setcom(Cmd_info *, Cmd_func);
  40 static void usage(void);
  41 static void sigexit(int sig);
  42 static int notfound(Cmd_info *);
  43 static void check_swap();

  45 const char *
  46 _ar_msg(Msg mid)
  47 {
  48         return (gettext(MSG_ORIG(mid)));
  49 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   21123 Thu Feb 28 11:26:05 2019
new/usr/src/cmd/sgs/m4/common/m4.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * 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) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright (c) 2011 Gary Mills
 */

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

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

#include        <signal.h>
#include        <unistd.h>
#include        <fcntl.h>
#include        "m4.h"

#if defined(__lint)
extern int yydebug;
#endif

#define match(c, s)     (c == *s && (!s[1] || inpmatch(s+1)))

static char     tmp_name[] = "/tmp/m4aXXXXX";
static wchar_t  prev_char;
static int mb_cur_max;

static void getflags(int *, char ***, int *);
static void initalloc(void);
static void expand(wchar_t **, int);
static void lnsync(FILE *);
static void fpath(FILE *);
static void puttok(wchar_t *);
static void error3(void);
static wchar_t itochr(int);
```



```
/*LINTED: E_STATIC_UNUSED*/
static wchar_t *chkbltin(wchar_t *);
static wchar_t *inpmatch(wchar_t *);
static void chkspace(char **, int *, char ***);
static void catchsig(int);
static FILE *m4open(char ***, char *, int *);
static void showwrap(void);
static void sputchr(wchar_t, FILE *);
static void putchr(wchar_t);
static void *xcalloc(size_t, size_t);
static wint_t myfgetwc(FILE *, int);
static wint_t myfputwc(wchar_t, FILE *);
static int myfeof(int);

int
main(int argc, char **argv)
{
        wchar_t t;
        int i, opt_end = 0;
        int sigs[] = {SIGHUP, SIGINT, SIGPIPE, 0};

#if defined(__lint)
        yydebug = 0;
#endif

        for (i = 0; sigs[i]; ++i) {
                if (signal(sigs[i], SIG_IGN) != SIG_IGN)
                        (void) signal(sigs[i], catchsig);
        }
        tempfile = mktemp(tmp_name);
        (void) close(creat(tempfile, 0));

        (void) setlocale(LC_ALL, "");

#if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
#define TEXT_DOMAIN "SYS_TEST"
#endif
        (void) textdomain(TEXT_DOMAIN);

        if ((mb_cur_max = MB_CUR_MAX) > 1)
                wide = 1;

        procnam = argv[0];
        getflags(&argc, &argv, &opt_end);
        initalloc();

        setfname("-");
        if (argc > 1) {
                --argc;
                ++argv;
                if (strcmp(argv[0], "-")) {
                        ifile[ifx] = m4open(&argv, "r", &argc);
                        setfname(argv[0]);
                }
        }

        for (;;) {
                token[0] = t = getchr();
                token[1] = EOS;

                if (t == WEOF) {
                        if (ifx > 0) {
                                (void) fclose(ifile[ifx]);
                                ipflr = ipstk[--ifx];
                                continue;
                        }
```

```
127                         getflags(&argc, &argv, &opt_end);

129                         if (argc <= 1)
130                                 /*
131                                  * If dowrap() has been called, the m4wrap
132                                  * macro has been processed, and a linked
133                                  * list of m4wrap strings has been created.
134                                  * The list starts at wrapstart.
135                                  */
136                                 if (wrapstart) {
137                                         /*
138                                          * Now that EOF has been processed,
139                                          * display the m4wrap strings.
140                                          */
141                                         showwrap();
142                                         continue;
143                                 } else
144                                         break;
145                         --argc;
146                         ++argv;

148                         if (ifile[ifx] != stdin)
149                                 (void) fclose(ifile[ifx]);

151                         if (strcmp(argv[0], "-"))
152                                 ifile[ifx] = m4open(&argv, "r", &argc);
153                         else
154                                 ifile[ifx] = stdin;

156                         setfname(argv[0]);
157                         continue;
158                 }

160                 if (is_alpha(t) || t == '_') {
161                         wchar_t *tp = token+1;
162                         int tlim = toksize;
163                         struct nlist    *macadd;  /* temp variable */

165                         while ((*tp = getchr()) != WEOF &&
166                             (is_alnum(*tp) || *tp == '_')) {
167                                 tp++;
168                                 if (--tlim <= 0)
169                                         error2(gettext(
170                                             "more than %d chars in word"),
171                                             toksize);
172                         }
173                         putbak(*tp);
174                         *tp = EOS;

176                         macadd = lookup(token);
177                         *Ap = (wchar_t *)macadd;
178                         if (macadd->def) {
179                                 if ((wchar_t *)(++Ap) >= astklm) {
180                                         --Ap;
181                                         error2(gettext(
182                                             "more than %d items on "
183                                             "argument stack"),
184                                             stksize);
185                                 }

187                                 if (Cp++ == NULL)
188                                         Cp = callst;

190                                 Cp->argp = Ap;
191                                 *Ap++ = op;
```



```
192                                 puttok(token);
193                                 stkchr(EOS);
194                                 t = getchr();
195                                 putbak(t);

197                                 if (t != '(')
198                                         pbstr(L"()");
199                                 else    /* try to fix arg count */
200                                         *Ap++ = op;

202                                 Cp->plev = 0;
203                         } else {
204                                 puttok(token);
205                         }
206                 } else if (match(t, lquote)) {
207                         int     qlev = 1;

209                         for (;;) {
210                                 token[0] = t = getchr();
211                                 token[1] = EOS;

213                                 if (match(t, rquote)) {
214                                         if (--qlev > 0)
215                                                 puttok(token);
216                                         else
217                                                 break;
218                                 } else if (match(t, lquote)) {
219                                         ++qlev;
220                                         puttok(token);
221                                 } else {
222                                         if (t == WEOF)
223                                                 error(gettext(
224                                                 "EOF in quote"));
225                                         putchr(t);
226                                 }
227                         }
228                 } else if (match(t, lcom) &&
229                     ((lcom[0] != L'#' || lcom[1] != L'\0') ||
230                     prev_char != '$')) {

232                         /*
233                          * Don't expand commented macro (between lcom and
234                          * rcom).
235                          * What we know so far is that we have found the
236                          * left comment char (lcom).
237                          * Make sure we haven't found '#' (lcom) immediately
238                          * preceded by '$' because we want to expand "$#".
239                          */

241                         puttok(token);
242                         for (;;) {
243                                 token[0] = t = getchr();
244                                 token[1] = EOS;
245                                 if (match(t, rcom)) {
246                                         puttok(token);
247                                         break;
248                                 } else {
249                                         if (t == WEOF)
250                                                 error(gettext(
251                                                 "EOF in comment"));
252                                         putchr(t);
253                                 }
254                         }
255                 } else if (Cp == NULL) {
256                         putchr(t);
257                 } else if (t == '(') {
```

```
258                         if (Cp->plev)
259                                 stkchr(t);
260                         else {
261                                 /* skip white before arg */
262                                 while ((t = getchr()) != WEOF && is_space(t))
263                                         ;
265                                 putbak(t);
266                         }
268                         ++Cp->plev;
269                 } else if (t == ')') {
270                         --Cp->plev;
272                         if (Cp->plev == 0) {
273                                 stkchr(EOS);
274                                 expand(Cp->argp, Ap-Cp->argp-1);
275                                 op = *Cp->argp;
276                                 Ap = Cp->argp-1;
278                                 if (--Cp < callst)
279                                         Cp = NULL;
280                         } else
281                                 stkchr(t);
282                 } else if (t == ',' && Cp->plev <= 1) {
283                         stkchr(EOS);
284                         *Ap = op;
286                         if ((wchar_t *)(++Ap) >= astklm) {
287                                 --Ap;
288                                 error2(gettext(
289                                     "more than %d items on argument stack"),
290                                     stksize);
291                         }
293                         while ((t = getchr()) != WEOF && is_space(t))
294                                 ;
296                         putbak(t);
297                 } else {
298                         stkchr(t);
299                 }
300         }
302         if (Cp != NULL)
303                 error(gettext(
304                 "EOF in argument list"));
306         delexit(exitstat, 1);
307         return (0);
308 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    3889 Thu Feb 28 11:26:05 2019
new/usr/src/cmd/syseventd/daemons/syseventd/sysevent_client.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * 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) 2000-2001 by Sun Microsystems, Inc.
 * All rights reserved.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * syseventd client interfaces
 */

#include <stdio.h>
#include <sys/types.h>
#include <stdarg.h>
#include <stddef.h>
#include <stdlib.h>
#include <unistd.h>
#include <door.h>
#include <errno.h>
#include <strings.h>
#include <thread.h>
#include <pthread.h>
#include <synch.h>
#include <syslog.h>
#include <fcntl.h>
#include <stropts.h>
#include <locale.h>
#include <libsysevent.h>
#include <sys/stat.h>
#include <sys/sysevent.h>

#include "syseventd.h"
#include "message.h"

/*
```



```
 * sysevent_client.c - contains routines particular to syseventd client
 *                     management (addition and deletion).
 */

/* Global client table and lock */
struct sysevent_client *sysevent_client_tbl[MAX_SLM];
mutex_t client_tbl_lock;

/*
 * initialize_client_tbl - Initialize each client entry in the syseventd
 *                         client table.  Each entry in the client table
 *                         entry represents one shared-object (SLM) client.
 */
void
initialize_client_tbl()
{
        struct sysevent_client  *scp;
        int                     i;

        for (i = 0; i < MAX_SLM; ++i) {
                if ((scp = (struct sysevent_client *)malloc(
                        sizeof (struct sysevent_client))) == NULL)
                        goto init_error;

                if (mutex_init(&scp->client_lock, USYNC_THREAD, NULL) != 0)
                        goto init_error;

                scp->client_data = NULL;
                scp->client_num = i;
                scp->eventq = NULL;

                /* Clear all flags when setting UNLOADED */
                scp->client_flags = SE_CLIENT_UNLOADED;

                sysevent_client_tbl[i] = scp;
        }

        return;

init_error:
        syseventd_err_print(INIT_CLIENT_TBL_ERR);
        syseventd_exit(1);
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   17203 Thu Feb 28 11:26:05 2019
new/usr/src/cmd/ucodeadm/ucodeadm.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

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

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

#include <sys/types.h>
#include <sys/processor.h>
#include <sys/ucode.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <errno.h>
#include <syslog.h>
#include <time.h>
#include <ctype.h>
#include <assert.h>
#include <libgen.h>
#include <locale.h>
#include <libintl.h>

#define UCODE_OPT_INSTALL       0x0001
#define UCODE_OPT_UPDATE        0x0002
#define UCODE_OPT_VERSION       0x0004

static const char ucode_dev[] = "/dev/" UCODE_DRIVER_NAME;

static char     *cmdname;
```



```
static char     ucode_vendor_str[UCODE_MAX_VENDORS_NAME_LEN];
static char     ucode_install_path[] = UCODE_INSTALL_PATH;

static int      ucode_debug = 0;

static int ucode_convert_amd(const char *, uint8_t *, size_t);
static int ucode_convert_intel(const char *, uint8_t *, size_t);

static ucode_errno_t ucode_gen_files_amd(uint8_t *, int, char *);
static ucode_errno_t ucode_gen_files_intel(uint8_t *, int, char *);

static const struct ucode_ops ucode_ops[] = {
        { ucode_convert_intel, ucode_gen_files_intel, ucode_validate_intel },
        { ucode_convert_amd, ucode_gen_files_amd, ucode_validate_amd },
};
______unchanged_portion_omitted_
```

```
**********************************************************
   12700 Thu Feb 28 11:26:05 2019
new/usr/src/cmd/ul/ul.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg  Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
/*
 * Copyright 2000 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

/*
 * Copyright (c) 1980 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <stdio.h>
#include <locale.h>
#include <wctype.h>
#include <widec.h>
#include <euc.h>
#include <getwidth.h>
#include <limits.h>
#include <stdlib.h>
#include <curses.h>
#include <term.h>
#include <string.h>

#define IESC    L'\033'
#define SO      L'\016'
#define SI      L'\017'
#define HFWD    L'9'
#define HREV    L'8'
#define FREV    L'7'
#define CDUMMY  -1

#define NORMAL  000
#define ALTSET  001     /* Reverse */
#define SUPERSC 002     /* Dim */
#define SUBSC   004     /* Dim | Ul */
#define UNDERL  010     /* Ul */
#define BOLD    020     /* Bold */

#define MEMFCT  16
/*
 * MEMFCT is a number that is likely to be large enough as a factor for
 * allocating more memory and to be small enough so as not wasting memory
 */

int     must_use_uc, must_overstrike;
char    *CURS_UP, *CURS_RIGHT, *CURS_LEFT,
        *ENTER_STANDOUT, *EXIT_STANDOUT, *ENTER_UNDERLINE, *EXIT_UNDERLINE,
        *ENTER_DIM, *ENTER_BOLD, *ENTER_REVERSE, *UNDER_CHAR, *EXIT_ATTRIBUTES;

struct  CHAR    {
        char    c_mode;
```



```
        wchar_t c_char;
};
______unchanged_portion_omitted_
```

```
**********************************************************
    8007 Thu Feb 28 11:26:05 2019
new/usr/src/cmd/valtools/ckitem.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * 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 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */


/*
 * Copyright (c) 2018, Joyent, Inc.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <signal.h>
#include <valtools.h>
#include <stdlib.h>
#include <locale.h>
#include <libintl.h>
#include <limits.h>
#include <wchar.h>
#include "usage.h"
#include "libadm.h"

#define BADPID  (-2)
#define INVISMAXSIZE 36

static char     *prog;
static char     *deflt = NULL, *prompt = NULL, *error = NULL, *help = NULL;
static int      kpid = BADPID;
static int      signo;

static char     *label, **invis;
static int      ninvis = 0;
static int      max = 1;
```



```
static int      attr = CKALPHA;

#define MAXSIZE 128
#define LSIZE   1024
#define INTERR  \
        "%s: ERROR: internal error occurred while attempting menu setup\n"
#define MYOPTS \
        "\t-f file             #file containing choices\n" \
        "\t-l label            #menu label\n" \
        "\t-i invis [, ...]    #invisible menu choices\n" \
        "\t-m max              #maximum choices user may select\n" \
        "\t-n                  #do not sort choices alphabetically\n" \
        "\t-o                  #don't prompt if only one choice\n" \
        "\t-u                  #unnumbered choices\n"

static const char       husage[] = "Wh";
static const char       eusage[] = "We";

static void
usage(void)
{
        switch (*prog) {
        default:
                (void) fprintf(stderr,
                        gettext("usage: %s [options] [choice [...]]\n"), prog);
                (void) fprintf(stderr, gettext(OPTMESG));
                (void) fprintf(stderr, gettext(MYOPTS));
                (void) fprintf(stderr, gettext(STDOPTS));
                break;

        case 'h':
                (void) fprintf(stderr,
                        gettext("usage: %s [options] [choice [...]]\n"), prog);
                (void) fprintf(stderr, gettext(OPTMESG));
                (void) fprintf(stderr,
                        gettext("\t-W width\n\t-h help\n"));
                break;

        case 'e':
                (void) fprintf(stderr,
                        gettext("usage: %s [options] [choice [...]]\n"), prog);
                (void) fprintf(stderr, gettext(OPTMESG));
                (void) fprintf(stderr,
                        gettext("\t-W width\n\t-e error\n"));
                break;
        }
        exit(1);
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   15738 Thu Feb 28 11:26:06 2019
new/usr/src/cmd/write/write.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg  Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
/*
 * 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) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */


/*
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright (c) 2016 by Delphix. All rights reserved.
 * Copyright (c) 2018, Joyent, Inc.
 */

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

#include        <ctype.h>
#include        <string.h>
#include        <stdio.h>
#include        <signal.h>
#include        <sys/wait.h>
#include        <sys/types.h>
#include        <sys/stat.h>
#include        <sys/utsname.h>
#include        <stdlib.h>
#include        <unistd.h>
#include        <time.h>
#include        <utmpx.h>
#include        <pwd.h>
#include        <fcntl.h>
#include        <stdarg.h>
#include        <locale.h>
#include        <stdlib.h>
#include        <limits.h>
#include        <wctype.h>
#include        <errno.h>
#include        <syslog.h>

#define         TRUE    1
#define         FALSE   0
#define         FAILURE -1
```



```
#define         DATE_FMT        "%a %b %e %H:%M:%S"
#define         UTMP_HACK  /* work around until utmpx is world writable */
/*
 *      DATE-TIME format
 *  %a  abbreviated weekday name
 *  %b  abbreviated month name
 *  %e  day of month
 *  %H  hour - 24 hour clock
 *  %M  minute
 *  %S  second
 *
 */

static int permit1(int);
static int permit(char *);
static int readcsi(int, char *, int);
static void setsignals();
static void shellcmd(char *);
static void openfail();
static void eof();

static struct   utsname utsn;

static FILE     *fp;    /* File pointer for receipient's terminal */
static char *rterm, *receipient; /* Pointer to receipient's terminal & name */
static char *thissys;

int
main(int argc, char **argv)
{
        int i;
        struct utmpx *ubuf;
        static struct utmpx self;
        char ownname[sizeof (self.ut_user) + 1];
        static char rterminal[sizeof ("/dev/") + sizeof (self.ut_line)] =
            "/dev/";
        extern char *rterm, *receipient;
        char *terminal, *ownterminal, *oterminal;
        short count;
        extern FILE *fp;
        char input[134+MB_LEN_MAX];
        char *ptr;
        time_t tod;
        char time_buf[40];
        struct passwd *passptr;
        char badterm[20][20];
        int bad = 0;
        uid_t   myuid;
        char *bp;
        int n;
        wchar_t wc;
        int c;
        int newline;

        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)
#define TEXT_DOMAIN "SYS_TEST"
#endif
        (void) textdomain(TEXT_DOMAIN);

        while ((c = getopt(argc, argv, "")) != EOF)
                switch (c) {
                        case '?':
                                (void) fprintf(stderr, "Usage: write %s\n",
                                gettext("user_name [terminal]"));
                                exit(2);
```

```
126                 }
127         myuid = geteuid();
128         uname(&utsn);
129         thissys = utsn.nodename;

131 /*      Set "rterm" to location where receipient's terminal will go.    */

133         rterm = &rterminal[sizeof ("/dev/") - 1];
134         terminal = NULL;

136         if (--argc <= 0) {
137             (void) fprintf(stderr, "Usage: write %s\n",
138                 gettext("user_name [terminal]"));
139             exit(1);
140             }
141         else
142             {
143             receipient = *++argv;
144             }

146 /*      Was a terminal name supplied?  If so, save it.                  */

148         if (--argc > 1) {
149             (void) fprintf(stderr, "Usage: write %s\n",
150                 gettext("user_name [terminal]"));
151             exit(1);
152         } else {
153             terminal = *++argv;
154         }

156 /*      One of the standard file descriptors must be attached to a      */
157 /*      terminal in "/dev".                                             */

159         if ((ownterminal = ttyname(fileno(stdin))) == NULL &&
160             (ownterminal = ttyname(fileno(stdout))) == NULL &&
161             (ownterminal = ttyname(fileno(stderr))) == NULL) {
162                 (void) fprintf(stderr,
163                         gettext("I cannot determine your terminal name."
164                                 " No reply possible.\n"));
165                 ownterminal = "/dev/???";
166         }

168         /*
169          * Set "ownterminal" past the "/dev/" at the beginning of
170          * the device name.
171          */
172         oterminal = ownterminal + sizeof ("/dev/")-1;

174         /*
175          * Scan through the "utmpx" file for your own entry and the
176          * entry for the person we want to send to.
177          */
178         for (self.ut_pid = 0, count = 0; (ubuf = getutxent()) != NULL; ) {
179         /* Is this a USER_PROCESS entry? */

181             if (ubuf->ut_type == USER_PROCESS) {
182 /*      Is it our entry?  (ie.  The line matches ours?)                 */

184                 if (strncmp(&ubuf->ut_line[0], oterminal,
185                     sizeof (ubuf->ut_line)) == 0) self = *ubuf;

187 /*      Is this the person we want to send to?                          */

189                 if (strncmp(receipient, &ubuf->ut_user[0],
190                     sizeof (ubuf->ut_user)) == 0) {
191 /*      If a terminal name was supplied, is this login at the correct   */
```



```
192 /*      terminal?  If not, ignore.  If it is right place, copy over the */
193 /*      name.                                                           */

195                     if (terminal != NULL) {
196                         if (strncmp(terminal, &ubuf->ut_line[0],
197                             sizeof (ubuf->ut_line)) == 0) {
198                             strlcpy(rterm, &ubuf->ut_line[0],
199                                 sizeof (rterminal) - (rterm - rterminal));
200                             if (myuid && !permit(rterminal)) {
201                                 bad++;
202                                 rterm[0] = '\0';
203                             }
204                             }
205                     }

207 /*      If no terminal was supplied, then take this terminal if no      */
208 /*      other terminal has been encountered already.                    */

210                     else
211                     {
212 /*      If this is the first encounter, copy the string into            */
213 /*      "rterminal".                                                    */

215                         if (*rterm == '\0') {
216                             strlcpy(rterm, &ubuf->ut_line[0],
217                                 sizeof (rterminal) - (rterm - rterminal));
218                             if (myuid && !permit(rterminal)) {
219                                 if (bad < 20) {
220                                         strlcpy(badterm[bad++], rterm,
221                                             sizeof (badterm[bad++]));
222                                 }
223                                 rterm[0] = '\0';
224                             } else if (bad > 0) {
225                                 (void) fprintf(stderr,
226                                 gettext(
227                                 "%s is logged on more than one place.\n"
228         "You are connected to \"%s\".\nOther locations are:\n"),
229                                         receipient, rterm);
230                                 for (i = 0; i < bad; i++)
231                                     (void) fprintf(stderr, "%s\n", badterm[i]);
232                             }
233                         }

235 /*      If this is the second terminal, print out the first.  In all    */
236 /*      cases of multiple terminals, list out all the other terminals   */
237 /*      so the user can restart knowing what their choices are.         */

239                         else if (terminal == NULL) {
240                             if (count == 1 && bad == 0) {
241                                 (void) fprintf(stderr,
242                                 gettext(
243                                 "%s is logged on more than one place.\n"
244         "You are connected to \"%s\".\nOther locations are:\n"),
245                                         receipient, rterm);
246                             }
247                             fwrite(&ubuf->ut_line[0], sizeof (ubuf->ut_line),
248                                 1, stderr);
249                             (void) fprintf(stderr, "\n");
250                             }

252                         count++;
253                     }                   /* End of "else" */
254                     }                   /* End of "else if (strncmp" */
255             }                   /* End of "if (USER_PROCESS" */
256             }           /* End of "for(count=0" */
```

```
258 /*      Did we find a place to talk to?  If we were looking for a       */
259 /*      specific spot and didn't find it, complain and quit.            */

261         if (terminal != NULL && *rterm == '\0') {
262             if (bad > 0) {
263                 (void) fprintf(stderr, gettext("Permission denied.\n"));
264                 exit(1);
265                 } else {
266 #ifdef UTMP_HACK
267                 if (strlcat(rterminal, terminal, sizeof (rterminal)) >=
268                     sizeof (rterminal)) {
269                         (void) fprintf(stderr,
270                             gettext("Terminal name too long.\n"));
271                         exit(1);
272                 }
273                 if (self.ut_pid == 0) {
274                         if ((passptr = getpwuid(getuid())) == NULL) {
275                             (void) fprintf(stderr,
276                                 gettext("Cannot determine who you are.\n"));
277                             exit(1);
278                         }
279                         (void) strlcpy(&ownname[0], &passptr->pw_name[0],
280                             sizeof (ownname));
281                 } else {
282                         (void) strlcpy(&ownname[0], self.ut_user,
283                             sizeof (self.ut_user));
284                 }
285                 if (!permit(rterminal)) {
286                         (void) fprintf(stderr,
287                                 gettext("%s permission denied\n"), terminal);
288                         exit(1);
289                 }
290 #else
291                 (void) fprintf(stderr, gettext("%s is not at \"%s\".\n"),
292                         receipient, terminal);
293                 exit(1);
294 #endif  /* UTMP_HACK */
295             }
296             }

298 /*      If we were just looking for anyplace to talk and didn't find    */
299 /*      one, complain and quit.                                         */
300 /*      If permissions prevent us from sending to this person - exit    */

302         else if (*rterm == '\0') {
303             if (bad > 0)
304                 (void) fprintf(stderr, gettext("Permission denied.\n"));
305             else
306                 (void) fprintf(stderr,
307                         gettext("%s is not logged on.\n"), receipient);
308             exit(1);
309             }

311 /*      Did we find our own entry?                                      */

313         else if (self.ut_pid == 0) {
314 /*      Use the user id instead of utmp name if the entry in the        */
315 /*      utmp file couldn't be found.                                    */

317             if ((passptr = getpwuid(getuid())) == (struct passwd *)NULL) {
318                 (void) fprintf(stderr,
319                         gettext("Cannot determine who you are.\n"));
320                 exit(1);
321             }
322             strncpy(&ownname[0], &passptr->pw_name[0], sizeof (ownname));
323             }
```



```
324         else
325             {
326             strncpy(&ownname[0], self.ut_user, sizeof (self.ut_user));
327             }
328         ownname[sizeof (ownname)-1] = '\0';

330         if (!permit1(1))
331                 (void) fprintf(stderr,
332                 gettext("Warning: You have your terminal set to \"mesg -n\"."
333                     " No reply possible.\n"));
334 /*      Close the utmpx files.                                          */

336         endutxent();

338 /*      Try to open up the line to the receipient's terminal.           */

340         signal(SIGALRM, openfail);
341         alarm(5);
342         fp = fopen(&rterminal[0], "w");
343         alarm(0);

345 /*      Make sure executed subshell doesn't inherit this fd - close-on-exec */

347         if (fcntl(fileno(fp), F_SETFD, FD_CLOEXEC) < 0)  {
348                 perror("fcntl(F_SETFD)");
349                 exit(1);
350         }

352 /*      Catch signals SIGHUP, SIGINT, SIGQUIT, and SIGTERM, and send    */
353 /*      <EOT> message to receipient before dying away.                  */

355         setsignals(eof);

357 /*      Get the time of day, convert it to a string and throw away the  */
358 /*      year information at the end of the string.                      */

360         time(&tod);
361         (void) strftime(time_buf, sizeof (time_buf),
362             dcgettext(NULL, DATE_FMT, LC_TIME), localtime(&tod));

364         (void) fprintf(fp,
365         gettext("\n\007\007\007\tMessage from %s on %s (%s) [ %s ] ...\n"),
366             &ownname[0], thissys, oterminal, time_buf);
367         fflush(fp);
368         (void) fprintf(stderr, "\007\007");

370 /*      Get input from user and send to receipient unless it begins     */
371 /*      with a !, when it is to be a shell command.                     */
372         newline = 1;
373         while ((i = readcsi(0, &input[0], sizeof (input))) > 0) {
374                 ptr = &input[0];
375 /*      Is this a shell command?                                        */

377                 if ((newline) && (*ptr == '!'))
378                         shellcmd(++ptr);

380 /*      Send line to the receipient.                                    */

382                 else {
383                         if (myuid && !permit1(fileno(fp))) {
384                                 (void) fprintf(stderr,
385                         gettext("Can no longer write to %s\n"), rterminal);
386                                 break;
387                         }

389 /*
```

```
 390  * All non-printable characters are displayed using a special notation:
 391  * Control characters  shall be displayed using the two character
 392  * sequence of ^ (carat) and the ASCII character - decimal 64 greater
 393  * that the character being encoded - eg., a \003 is displayed ^C.
 394  * Characters with the eighth bit set shall be displayed using
 395  * the three or four character meta notation - e.g., \372 is
 396  * displayed M-z and \203 is displayed M-^C.
 397  */

 399                         newline = 0;
 400                         for (bp = &input[0]; --i >= 0; bp++) {
 401                         if (*bp == '\n') {
 402                                 newline = 1;
 403                                 putc('\r', fp);
 404                         }
 405                         if (*bp == ' ' ||
 406                                  *bp == '\t' || *bp == '\n' ||
 407                                  *bp == '\r' || *bp == '\013' ||
 408                                  *bp == '\007') {
 409                                         putc(*bp, fp);
 410                         } else if (((n = mbtowc(&wc, bp, MB_CUR_MAX)) > 0) &&
 411                                 iswprint(wc)) {
 412                                 for (; n > 0; --n, --i, ++bp)
 413                                         putc(*bp, fp);
 414                                 bp--, ++i;
 415                         } else {
 416                                 if (!isascii(*bp)) {
 417                                         fputs("M-", fp);
 418                                         *bp = toascii(*bp);
 419                                 }
 420                                 if (iscntrl(*bp)) {
 421                                         putc('^', fp);
 422 /*      add decimal 64 to the control character                 */
 423                                         putc(*bp + 0100, fp);
 424                                 }
 425                                 else
 426                                         putc(*bp, fp);
 427                         }
 428                         if (*bp == '\n')
 429                                 fflush(fp);
 430                         if (ferror(fp) || feof(fp)) {
 431                                 printf(gettext(
 432                                 "\n\007Write failed (%s logged out?)\n"),
 433                                 receipient);
 434                                 exit(1);
 435                         }
 436                         } /* for */
 437                         fflush(fp);
 438         } /* else */
 439         } /* while */

 441 /*      Since "end of file" received, send <EOT> message to receipient. */

 443         eof();
 444         return (0);
 445 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   15055 Thu Feb 28 11:26:06 2019
new/usr/src/cmd/zdump/zdump.c
10120 smatch indenting fixes for usr/src/cmd
Reviewed by: Gerg__ Doma <domag02@gmail.com>
Portions contributed by: Joyce McIntosh <joyce.mcintosh@nexenta.com>
**********************************************************
```

```
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 * zdump 7.24
 * Taken from elsie.nci.nih.gov to replace the existing Solaris zdump,
 * which was based on an earlier version of the elsie code.
 *
 * For zdump 7.24, the following changes were made to the elsie code:
 *   locale/textdomain/messages to match existing Solaris style.
 *   Solaris verbose mode is documented to display the current time first.
 *   cstyle cleaned code.
 *   removed old locale/textdomain code.
 */

static char     elsieid[] = "@(#)zdump.c        7.74";

/*
 * This code has been made independent of the rest of the time
 * conversion package to increase confidence in the verification it provides.
 * You can use this code to help in verifying other implementations.
 */

#include "stdio.h"      /* for stdout, stderr, perror */
#include "string.h"     /* for strcpy */
#include "sys/types.h"  /* for time_t */
#include "time.h"       /* for struct tm */
#include "stdlib.h"     /* for exit, malloc, atoi */
#include "locale.h"     /* for setlocale, textdomain */
#include "libintl.h"
#include <ctype.h>
#include "tzfile.h"     /* for defines */
#include <limits.h>

#ifndef ZDUMP_LO_YEAR
#define ZDUMP_LO_YEAR   (-500)
#endif /* !defined ZDUMP_LO_YEAR */

#ifndef ZDUMP_HI_YEAR
#define ZDUMP_HI_YEAR   2500
#endif /* !defined ZDUMP_HI_YEAR */

#ifndef MAX_STRING_LENGTH
#define MAX_STRING_LENGTH       1024
#endif /* !defined MAX_STRING_LENGTH */

#ifndef TRUE
#define TRUE            1
#endif /* !defined TRUE */

#ifndef FALSE
#define FALSE           0
#endif /* !defined FALSE */
```



```
#ifndef isleap_sum
/*
 * See tzfile.h for details on isleap_sum.
 */
#define isleap_sum(a, b)        isleap((a) % 400 + (b) % 400)
#endif /* !defined isleap_sum */

#ifndef SECSPERDAY
#define SECSPERDAY      ((long)SECSPERHOUR * HOURSPERDAY)
#endif
#define SECSPERNYEAR    (SECSPERDAY * DAYSPERNYEAR)
#define SECSPERLYEAR    (SECSPERNYEAR + SECSPERDAY)

#ifndef GNUC_or_lint
#ifdef lint
#define GNUC_or_lint
#else /* !defined lint */
#ifdef __GNUC__
#define GNUC_or_lint
#endif /* defined __GNUC__ */
#endif /* !defined lint */
#endif /* !defined GNUC_or_lint */

#ifndef INITIALIZE
#ifdef  GNUC_or_lint
#define INITIALIZE(x)   ((x) = 0)
#else /* !defined GNUC_or_lint */
#define INITIALIZE(x)
#endif /* !defined GNUC_or_lint */
#endif /* !defined INITIALIZE */

static time_t   absolute_min_time;
static time_t   absolute_max_time;
static size_t   longest;
static char     *progname;
static int      warned;

static char     *abbr(struct tm *);
static void     abbrok(const char *, const char *);
static long     delta(struct tm *, struct tm *);
static void     dumptime(const struct tm *);
static time_t   hunt(char *, time_t, time_t);
static void     setabsolutes(void);
static void     show(char *, time_t, int);
static void     usage(void);
static const char       *tformat(void);
static time_t   yeartot(long y);

#ifndef TYPECHECK
#define my_localtime    localtime
#else /* !defined TYPECHECK */
static struct tm *
my_localtime(tp)
time_t *tp;
{
        register struct tm *tmp;

        tmp = localtime(tp);
        if (tp != NULL && tmp != NULL) {
                struct tm       tm;
                register time_t t;

                tm = *tmp;
                t = mktime(&tm);
                if (t - *tp >= 1 || *tp - t >= 1) {
```

```
126                         (void) fflush(stdout);
127                         (void) fprintf(stderr, "\n%s: ", progname);
128                         (void) fprintf(stderr, tformat(), *tp);
129                         (void) fprintf(stderr, " ->");
130                         (void) fprintf(stderr, " year=%d", tmp->tm_year);
131                         (void) fprintf(stderr, " mon=%d", tmp->tm_mon);
132                         (void) fprintf(stderr, " mday=%d", tmp->tm_mday);
133                         (void) fprintf(stderr, " hour=%d", tmp->tm_hour);
134                         (void) fprintf(stderr, " min=%d", tmp->tm_min);
135                         (void) fprintf(stderr, " sec=%d", tmp->tm_sec);
136                         (void) fprintf(stderr, " isdst=%d", tmp->tm_isdst);
137                         (void) fprintf(stderr, " -> ");
138                         (void) fprintf(stderr, tformat(), t);
139                         (void) fprintf(stderr, "\n");
140                 }
141         }
142         return (tmp);
143 }
```

*unchanged_portion_omitted_*

```
178 int
179 main(argc, argv)
180 int     argc;
181 char    *argv[];
182 {
183         register int            i;
184         register int            c;
185         register int            vflag;
186         register char           *cutarg;
187         register long           cutloyear = ZDUMP_LO_YEAR;
188         register long           cuthiyear = ZDUMP_HI_YEAR;
189         register time_t         cutlotime;
190         register time_t         cuthitime;
191         time_t                  now;
192         time_t                  t;
193         time_t                  newt;
194         struct tm               tm;
195         struct tm               newtm;
196         register struct tm      *tmp;
197         register struct tm      *newtmp;

199         INITIALIZE(cutlotime);
200         INITIALIZE(cuthitime);

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

208         progname = argv[0];
209         for (i = 1; i < argc; ++i)
210                 if (strcmp(argv[i], "--version") == 0) {
211                         (void) printf("%s\n", elsieid);
212                         exit(EXIT_SUCCESS);
213                 }
214         vflag = 0;
215         cutarg = NULL;
216         while ((c = getopt(argc, argv, "c:v")) == 'c' || c == 'v')
217                 if (c == 'v')
218                         vflag = 1;
219                 else    cutarg = optarg;
220         if (c != EOF ||
221                 (optind == argc - 1 && strcmp(argv[optind], "=") == 0)) {
222                         usage();
223                         /* NOTREACHED */
```



```
224         }
225         if (vflag) {
226                 if (cutarg != NULL) {
227                         long    lo;
228                         long    hi;
229                         char    dummy;

231                         if (sscanf(cutarg, "%ld%c", &hi, &dummy) == 1) {
232                                 cuthiyear = hi;
233                         } else if (sscanf(cutarg, "%ld,%ld%c",
234                                 &lo, &hi, &dummy) == 2) {
235                                         cutloyear = lo;
236                                         cuthiyear = hi;
237                         } else {
238                                 (void) fprintf(stderr,
239                                         gettext("%s: wild -c argument %s\n"),
234 (void) fprintf(stderr, gettext("%s: wild -c argument %s\n"),
240                                         progname, cutarg);
241                                 exit(EXIT_FAILURE);
242                         }
243                 }
244                 setabsolutes();
245                 cutlotime = yeartot(cutloyear);
246                 cuthitime = yeartot(cuthiyear);
247         }
248         (void) time(&now);
249         longest = 0;
250         for (i = optind; i < argc; ++i)
251                 if (strlen(argv[i]) > longest)
252                         longest = strlen(argv[i]);

254         for (i = optind; i < argc; ++i) {
255                 static char     buf[MAX_STRING_LENGTH];
256                 static char     *tzp = NULL;

258                 (void) unsetenv("TZ");
259                 if (tzp != NULL)
260                         free(tzp);
261                 if ((tzp = malloc(3 + strlen(argv[i]) + 1)) == NULL) {
262                         perror(progname);
263                         exit(EXIT_FAILURE);
264                 }
265                 (void) strcpy(tzp, "TZ=");
266                 (void) strcat(tzp, argv[i]);
267                 if (putenv(tzp) != 0) {
268                         perror(progname);
269                         exit(EXIT_FAILURE);
270                 }
271                 if (!vflag) {
272                         show(argv[i], now, FALSE);
273                         continue;
274                 }

276 #if defined(sun)
277                 /*
278                  * We show the current time first, probably because we froze
279                  * the behavior of zdump some time ago and then it got
280                  * changed.
281                  */
282                 show(argv[i], now, TRUE);
283 #endif
284                 warned = FALSE;
285                 t = absolute_min_time;
286                 show(argv[i], t, TRUE);
287                 t += SECSPERHOUR * HOURSPERDAY;
288                 show(argv[i], t, TRUE);
```

```
289                 if (t < cutlotime)
290                         t = cutlotime;
291                 tmp = my_localtime(&t);
292                 if (tmp != NULL) {
293                         tm = *tmp;
294                         (void) strncpy(buf, abbr(&tm), sizeof (buf) - 1);
295                 }
296                 for (;;) {
297                         if (t >= cuthitime)
298                                 break;
299                         /* check if newt will overrun maximum time_t value */
300                         if (t > LONG_MAX - (SECSPERHOUR * 12))
301                                 break;
302                         newt = t + SECSPERHOUR * 12;
303                         if (newt >= cuthitime)
304                                 break;
305                         newtmp = localtime(&newt);
306                         if (newtmp != NULL)
307                                 newtm = *newtmp;
308                         if ((tmp == NULL || newtmp == NULL) ? (tmp != newtmp) :
309                                 (delta(&newtm, &tm) != (newt - t) ||
310                                 newtm.tm_isdst != tm.tm_isdst ||
311                                 strcmp(abbr(&newtm), buf) != 0)) {
312                                         newt = hunt(argv[i], t, newt);
313                                         newtmp = localtime(&newt);
314                                         if (newtmp != NULL) {
315                                                 newtm = *newtmp;
316                                                 (void) strncpy(buf,
317                                                         abbr(&newtm),
318                                                         sizeof (buf) - 1);
319                                         }
320                         }
321                         t = newt;
322                         tm = newtm;
323                         tmp = newtmp;
324                 }
325                 t = absolute_max_time;
326 #if defined(sun)
327                 show(argv[i], t, TRUE);
328                 t -= SECSPERHOUR * HOURSPERDAY;
329                 show(argv[i], t, TRUE);
330 #else /* !defined(sun) */
331                 t -= SECSPERHOUR * HOURSPERDAY;
332                 show(argv[i], t, TRUE);
333                 t += SECSPERHOUR * HOURSPERDAY;
334                 show(argv[i], t, TRUE);
335 #endif /* !defined(sun) */
336         }
337         if (fflush(stdout) || ferror(stdout)) {
338                 (void) fprintf(stderr, "%s: ", progname);
339                 (void) perror(gettext("Error writing standard output"));
340                 exit(EXIT_FAILURE);
341         }
342         return (EXIT_SUCCESS);
343 }
```

_____***unchanged_portion_omitted_***