```
**********************************************************
   50603 Wed Mar 27 12:06:34 2013
new/usr/src/cmd/mv/mv.c
3619 cp -p clobbers permissions/ownership following symbolic links
**********************************************************
```

```
/*
 * 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 2013 Nexenta Systems, Inc. All rights reserved.
 */

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

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

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

/*
 * Combined mv/cp/ln command:
 *      mv file1 file2
 *      mv dir1 dir2
 *      mv file1 ... filen dir1
 */
#include <sys/time.h>
#include <signal.h>
#include <locale.h>
#include <stdarg.h>
#include <sys/acl.h>
#include <libcmdutils.h>
#include <aclutils.h>
#include "getresponse.h"

#define FTYPE(A)        (A.st_mode)
#define FMODE(A)        (A.st_mode)
#define UID(A)          (A.st_uid)
```



```
#define GID(A)          (A.st_gid)
#define IDENTICAL(A, B) (A.st_dev == B.st_dev && A.st_ino == B.st_ino)
#define ISDIR(A)        ((A.st_mode & S_IFMT) == S_IFDIR)
#define ISDOOR(A)       ((A.st_mode & S_IFMT) == S_IFDOOR)
#define ISLNK(A)        ((A.st_mode & S_IFMT) == S_IFLNK)
#define ISREG(A)        (((A).st_mode & S_IFMT) == S_IFREG)
#define ISDEV(A)        ((A.st_mode & S_IFMT) == S_IFCHR || \
                        (A.st_mode & S_IFMT) == S_IFBLK || \
                        (A.st_mode & S_IFMT) == S_IFIFO)
#define ISSOCK(A)       ((A.st_mode & S_IFMT) == S_IFSOCK)

#define DELIM   '/'
#define EQ(x, y)        (strcmp(x, y) == 0)
#define FALSE   0
#define MODEBITS (S_ISUID|S_ISGID|S_ISVTX|S_IRWXU|S_IRWXG|S_IRWXO)
#define TRUE 1

static char             *dname(char *);
static int              lnkfil(char *, char *);
static int              cpymve(char *, char *);
static int              chkfiles(char *, char **);
static int              rcopy(char *, char *);
static int              chk_different(char *, char *);
static int              chg_time(char *, struct stat);
static int              chg_mode(char *, uid_t, gid_t, mode_t);
static int              copydir(char *, char *);
static int              copyspecial(char *);
static int              getrealpath(char *, char *);
static void             usage(void);
static void             Perror(char *);
static void             Perror2(char *, char *);
static int              use_stdin(void);
static int              copyattributes(char *, char *);
static int              copy_sysattr(char *, char *);
static tree_node_t      *create_tnode(dev_t, ino_t);

static struct stat      s1, s2, s3, s4;
static int              cpy = FALSE;
static int              mve = FALSE;
static int              lnk = FALSE;
static char             *cmd;
static int              silent = 0;
static int              fflg = 0;
static int              iflg = 0;
static int              pflg = 0;
static int              Rflg = 0;       /* recursive copy */
static int              rflg = 0;       /* recursive copy */
static int              sflg = 0;
static int              Hflg = 0;       /* follow cmd line arg symlink to dir */
static int              Lflg = 0;       /* follow symlinks */
static int              Pflg = 0;       /* do not follow symlinks */
static int              atflg = 0;
static int              attrsilent = 0;
static int              targetexists = 0;
static int              cmdarg;         /* command line argument */
static avl_tree_t       *stree = NULL;  /* source file inode search tree */
static acl_t            *s1acl;
static int              saflg = 0;      /* 'cp' extended system attr. */
static int              srcfd = -1;
static int              targfd = -1;
static int              sourcedirfd = -1;
static int              targetdirfd = -1;
static DIR              *srcdirp = NULL;
static int              srcattrfd = -1;
static int              targattrfd = -1;
static struct stat      attrdir;
```

```
129 /* Extended system attributes support */

131 static int open_source(char  *);
132 static int open_target_srctarg_attrdirs(char  *, char *);
133 static int open_attrdirp(char *);
134 static int traverse_attrfile(struct dirent *, char *, char *, int);
135 static void rewind_attrdir(DIR *);
136 static void close_all();


139 int
140 main(int argc, char *argv[])
141 {
142         int c, i, r, errflg = 0;
143         char target[PATH_MAX];
144         int (*move)(char *, char *);

146         /*
147          * Determine command invoked (mv, cp, or ln)
148          */

150         if (cmd = strrchr(argv[0], '/'))
151                 ++cmd;
152         else
153                 cmd = argv[0];

155         /*
156          * Set flags based on command.
157          */

159         (void) setlocale(LC_ALL, "");
160 #if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
161 #define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
162 #endif
163         (void) textdomain(TEXT_DOMAIN);
164         if (init_yes() < 0) {
165                 (void) fprintf(stderr, gettext(ERR_MSG_INIT_YES),
166                     strerror(errno));
167                 exit(3);
168         }

170         if (EQ(cmd, "mv"))
171                 mve = TRUE;
172         else if (EQ(cmd, "ln"))
173                 lnk = TRUE;
174         else if (EQ(cmd, "cp"))
175                 cpy = TRUE;
176         else {
177                 (void) fprintf(stderr,
178                     gettext("Invalid command name (%s); expecting "
179                     "mv, cp, or ln.\n"), cmd);
180                 exit(1);
181         }

183         /*
184          * Check for options:
185          *      cp  -r|-R [-H|-L|-P] [-fip@/] file1 [file2 ...] target
186          *      cp [-fiprR@/] file1 [file2 ...] target
187          *      ln [-f] [-n] [-s] file1 [file2 ...] target
188          *      ln [-f] [-n] [-s] file1 [file2 ...]
189          *      mv [-f|i] file1 [file2 ...] target
190          *      mv [-f|i] dir1 target
191          */

193         if (cpy) {
```



```
194                 while ((c = getopt(argc, argv, "fHiLpPrR@/")) != EOF)
195                         switch (c) {
196                         case 'f':
197                                 fflg++;
198                                 break;
199                         case 'i':
200                                 iflg++;
201                                 break;
202                         case 'p':
203                                 pflg++;
204 #ifdef XPG4
205                                 attrsilent = 1;
206                                 atflg = 0;
207                                 saflg = 0;
208 #else
209                                 if (atflg == 0)
210                                         attrsilent = 1;
211 #endif
212                                 break;
213                         case 'H':
214                                 /*
215                                  * If more than one of -H, -L, or -P are
216                                  * specified, only the last option specified
217                                  * determines the behavior.
218                                  */
219                                 Lflg = Pflg = 0;
220                                 Hflg++;
221                                 break;
222                         case 'L':
223                                 Hflg = Pflg = 0;
224                                 Lflg++;
225                                 break;
226                         case 'P':
227                                 Lflg = Hflg = 0;
228                                 Pflg++;
229                                 break;
230                         case 'R':
231                                 /*
232                                  * The default behavior of cp -R|-r
233                                  * when specified without -H|-L|-P
234                                  * is -L.
235                                  */
236                                 Rflg++;
237                                 /*FALLTHROUGH*/
238                         case 'r':
239                                 rflg++;
240                                 break;
241                         case '@':
242                                 atflg++;
243                                 attrsilent = 0;
244 #ifdef XPG4
245                                 pflg = 0;
246 #endif
247                                 break;
248                         case '/':
249                                 saflg++;
250                                 attrsilent = 0;
251 #ifdef XPG4
252                                 pflg = 0;
253 #endif
254                                 break;
255                         default:
256                                 errflg++;
257                         }

259                 /* -R or -r must be specified with -H, -L, or -P */
```

```
260                 if ((Hflg || Lflg || Pflg) && !(Rflg || rflg)) {
261                         errflg++;
262                 }

264         } else if (mve) {
265                 while ((c = getopt(argc, argv, "fis")) != EOF)
266                         switch (c) {
267                         case 'f':
268                                 silent++;
269 #ifdef XPG4
270                                 iflg = 0;
271 #endif
272                                 break;
273                         case 'i':
274                                 iflg++;
275 #ifdef XPG4
276                                 silent = 0;
277 #endif
278                                 break;
279                         default:
280                                 errflg++;
281                         }
282         } else { /* ln */
283                 while ((c = getopt(argc, argv, "fns")) != EOF)
284                         switch (c) {
285                         case 'f':
286                                 silent++;
287                                 break;
288                         case 'n':
289                                 /* silently ignored; this is the default */
290                                 break;
291                         case 's':
292                                 sflg++;
293                                 break;
294                         default:
295                                 errflg++;
296                         }
297         }

299         /*
300          * For BSD compatibility allow - to delimit the end of
301          * options for mv.
302          */
303         if (mve && optind < argc && (strcmp(argv[optind], "-") == 0))
304                 optind++;

306         /*
307          * Check for sufficient arguments
308          * or a usage error.
309          */

311         argc -= optind;
312         argv  = &argv[optind];

314         if ((argc < 2 && lnk != TRUE) || (argc < 1 && lnk == TRUE)) {
315                 (void) fprintf(stderr,
316                     gettext("%s: Insufficient arguments (%d)\n"),
317                     cmd, argc);
318                 usage();
319         }

321         if (errflg != 0)
322                 usage();

324         /*
325          * If there is more than a source and target,
```



```
326          * the last argument (the target) must be a directory
327          * which really exists.
328          */

330         if (argc > 2) {
331                 if (stat(argv[argc-1], &s2) < 0) {
332                         (void) fprintf(stderr,
333                             gettext("%s: %s not found\n"),
334                             cmd, argv[argc-1]);
335                         exit(2);
336                 }

338                 if (!ISDIR(s2)) {
339                         (void) fprintf(stderr,
340                             gettext("%s: Target %s must be a directory\n"),
341                             cmd, argv[argc-1]);
342                         usage();
343                 }
344         }

346         if (strlen(argv[argc-1]) >= PATH_MAX) {
347                 (void) fprintf(stderr,
348                     gettext("%s: Target %s file name length exceeds PATH_MAX"
349                     " %d\n"), cmd, argv[argc-1], PATH_MAX);
350                 exit(78);
351         }

353         if (argc == 1) {
354                 if (!lnk)
355                         usage();
356                 (void) strcpy(target, ".");
357         } else {
358                 (void) strcpy(target, argv[--argc]);
359         }

361         /*
362          * Perform a multiple argument mv|cp|ln by
363          * multiple invocations of cpymve() or lnkfil().
364          */
365         if (lnk)
366                 move = lnkfil;
367         else
368                 move = cpymve;

370         r = 0;
371         for (i = 0; i < argc; i++) {
372                 stree = NULL;
373                 cmdarg = 1;
374                 r += move(argv[i], target);
375         }

377         /*
378          * Show errors by nonzero exit code.
379          */

381         return (r?2:0);
382 }
```

*_____unchanged_portion_omitted_*

```
1334 /*
1335  * chg_time()
1336  *
1337  * Try to preserve modification and access time.
1338  * If 1) pflg is not set, or 2) pflg is set and this is the Solaris version,
1339  * don't report a utimensat() failure.
1340  * If this is the XPG4 version and utimensat fails, if 1) pflg is set (cp -p)
```

```
 * or 2) we are doing a mv, print a diagnostic message; arrange for a non-zero
 * exit status only if pflg is set.
 * utimensat(2) is being used to achieve granularity in nanoseconds
 * (if supported by the underlying file system) while setting file times.
 */
static int
chg_time(char *to, struct stat ss)
{
        struct timespec times[2];
        int rc;

        times[0] = ss.st_atim;
        times[1] = ss.st_mtim;

        rc = utimensat(AT_FDCWD, to, times,
            ISLNK(s1) ? AT_SYMLINK_NOFOLLOW : 0);
        rc = utimensat(AT_FDCWD, to, times, 0);
#ifdef XPG4
        if ((pflg || mve) && rc != 0) {
                (void) fprintf(stderr,
                    gettext("%s: cannot set times for %s: "), cmd, to);
                perror("");
                if (pflg)
                        return (1);
        }
#endif

        return (0);

}

/*
 * chg_mode()
 *
 * This function is called upon "cp -p" or mv across filesystems.
 *
 * Try to preserve the owner and group id.  If chown() fails,
 * only print a diagnostic message if doing a mv in the XPG4 version;
 * try to clear S_ISUID and S_ISGID bits in the target.  If unable to clear
 * S_ISUID and S_ISGID bits, print a diagnostic message and arrange for a
 * non-zero exit status because this is a security violation.
 * Try to preserve permissions.
 * If this is the XPG4 version and chmod() fails, print a diagnostic message
 * and arrange for a non-zero exit status.
 * If this is the Solaris version and chmod() fails, do not print a
 * diagnostic message or exit with a non-zero value.
 */
static int
chg_mode(char *target, uid_t uid, gid_t gid, mode_t mode)
{
        int clearflg = 0; /* controls message printed upon chown() error */
        struct stat st;

        /* Don't change mode if target is symlink */
        if (lstat(target, &st) == 0 && ISLNK(st))
                return (0);

        if (chown(target, uid, gid) != 0) {
#ifdef XPG4
                if (mve) {
                        (void) fprintf(stderr, gettext("%s: cannot change"
                            " owner and group of %s: "), cmd, target);
                        perror("");
                }
#endif
                if (mode & (S_ISUID | S_ISGID)) {
```



```
                        /* try to clear S_ISUID and S_ISGID */
                        mode &= ~S_ISUID & ~S_ISGID;
                        ++clearflg;
                }
        }
        if (chmod(target, mode) != 0) {
                if (clearflg) {
                        (void) fprintf(stderr, gettext(
                            "%s: cannot clear S_ISUID and S_ISGID bits in"
                            " %s: "), cmd, target);
                        perror("");
                        /* cp -p should get non-zero exit; mv should not */
                        if (pflg)
                                return (1);
                }
#ifdef XPG4
                else {
                        (void) fprintf(stderr, gettext(
                        "%s: cannot set permissions for %s: "), cmd, target);
                        perror("");
                        /* cp -p should get non-zero exit; mv should not */
                        if (pflg)
                                return (1);
                }
#endif
        }
        return (0);

}
```

unchanged_portion_omitted_