ECE357, Computer Operating Systems – Problem Set #2, Problem 3

Nikola Janjušević

October 1, 2018

Example Usage

```
nikopj@t480s:~/Documents/OS/hw2$ ./rrdir.out ./
 14418750      4 drwxrwxr-x   2 nikopj    nikopj        4096 Oct  1 15:52 ./report
 14418756      4 -rw-rw-r--   1 nikopj    nikopj          68 Oct  1 15:52 ./report/report.aux
 14418757     68 -rw-rw-r--   1 nikopj    nikopj       67765 Oct  1 15:52 ./report/report.pdf
 14418696     28 -rw-rw-r--   1 nikopj    nikopj       25856 Oct  1 15:52 ./report/report.log
 14418752      4 -rw-rw-r--   1 nikopj    nikopj         858 Oct  1 15:52 ./report/report.tex
 14418751    128 -rw-rw-r--   1 nikopj    nikopj      130675 Oct  1 15:52 ./report/report.synctex.gz
        2      1 drwxr-xr-x   6 root      root          1024 Sep 11 22:14 ./mountpoint
     1282      1 d-w-rwxr-x   2 root      root          1024 Sep 11 22:16 ./mountpoint/dir2
     1287      1 -r--rwx--x   1 123       456             13 Sep 11 22:16 ./mountpoint/dir2/file3
     1288      1 -rw-r--r--   2 root      root            38 Sep 11 22:16 ./mountpoint/dir2/file4
     1281      1 drwxr-xr-x   2 bin       root          1024 Sep 11 22:22 ./mountpoint/dir1
     1289      0 p---rw-rw-   1 root      root             0 Sep 11 22:19 ./mountpoint/dir1/pipe1
     1290      0 crw-r--r--   1 root      root             0 Sep 11 22:22 ./mountpoint/dir1/dev1
     1285      1 -rw-r--r--   1 root      root            15 Jan  1 00:00 ./mountpoint/dir1/file2
     1284      1 -rw-r--r--   1 root      root            11 Sep 26 13:13 ./mountpoint/dir1/file1
     1283      1 d--xrwxr-x   2 root      uucp          1024 Sep 11 22:22 ./mountpoint/dir3
     1291      0 brw-r--r--   1 root      root             0 Sep 11 22:22 ./mountpoint/dir3/dev2
     1286      0 lrwxrwxrwx   1 root      root            13 Sep 11 22:16 ./mountpoint/dir3/link1 -> ../dir2/file2
     1288      1 -rw-r--r--   2 root      root            38 Sep 11 22:16 ./mountpoint/dir3/file5
       11     12 drwx------   2 root      root         12288 Sep 11 22:14 ./mountpoint/lost+found
Cannot open directory ./mountpoint/lost+found: Permission denied
 14418743      0 lrwxrwxrwx   1 nikopj    nikopj          20 Oct  1 11:39 ./symlink -> ../MINICAT/minicat.c
 14418413      8 -rw-r--r--   1 nikopj    nikopj        5912 Oct  1 15:38 ./rrdir.c
 14418693      4 -rw-r--r--   1 nikopj    nikopj         462 Sep 30 22:09 ./test.c
 14418329     20 -rwxr-xr-x   1 nikopj    nikopj       19256 Oct  1 15:38 ./a.out
 14418758     16 -rwxr-xr-x   1 nikopj    nikopj       13664 Oct  1 15:53 ./rrdir.out
 15728705  10240 -rw-rw-r--   1 nikopj    nikopj    10485760 Oct  1 14:47 ./testfs.img
nikopj@t480s:~/Documents/OS/hw2$ ls
a.out  mountpoint  report  rrdir.c  rrdir.out  symlink  test.c  testfs.img
nikopj@t480s:~/Documents/OS/hw2$
```

Program: rrdir.c

```
1  /*
2  Nikola Janjusevic 2018
3  rrdir.c -- recursively traverses directory, printing out meta data stats
4  */
5
6  #include <dirent.h>
7  #include <stdio.h>
8  #include <string.h>
9  #include <errno.h>
10 #include <sys/types.h>
11 #include <sys/stat.h>
12 #include <unistd.h>
13 #include <libgen.h>
14 #include <stdlib.h>
15 #include <pwd.h>
16 #include <grp.h>
17 #include <time.h>
18
19 #define BUFFSIZE 256
20
21 // recursively read directories (and print stats)
22 int rrdir(char *dn);
23
24 int main(int argc, char **argv)
25 {
26   char *path;
27   if(argc == 2)
28   {
29     path = argv[1];
30     // remove any leading slashes for nicer printing
31     while(path[strlen(path)-1] == '/')
32       path[strlen(path)-1] = '\0';
33     rrdir(argv[1]);
34   }
35   else
36     printf("Usage: %s [staring directory]\n", argv[0]);
37   return 0;
38 }
39
40 // returns char corresponding to filetype given by st_mode from stat struct
41 char charftype(mode_t st_mode)
42 {
43   switch( st_mode & S_IFMT )
44   {
45     case S_IFSOCK: return 's';
46     case S_IFLNK:  return 'l';
47     case S_IFREG:  return '-';
48     case S_IFBLK:  return 'b';
49     case S_IFDIR:  return 'd';
50     case S_IFCHR:  return 'c';
51     case S_IFIFO:  return 'p';
52     default:       return '?';
53   }
54 }
55
56 // returns string of file permission in rwx format (puts in dest)
57 // (for user, group, and others)
58 char *strfperm(mode_t st_mode, char *dest)
59 {
60   char octperm[4];
61   char *cptr;
62   sprintf(octperm, "%03o", st_mode & ~S_IFMT);
63
64   int i;
65   for(i=0; i<3; i++)
66   {
67     cptr = dest + 3*i;
68     switch(octperm[i])
69     {
70       case '0': strcpy(cptr, "---");
71         break;
```

```
72        case '1': strcpy(cptr, "--x");
73          break;
74        case '2': strcpy(cptr, "-w-");
75          break;
76        case '4': strcpy(cptr, "r--");
77          break;
78        case '5': strcpy(cptr, "r-x");
79          break;
80        case '6': strcpy(cptr, "rw-");
81          break;
82        case '7': strcpy(cptr, "rwx");
83          break;
84        default:  strcpy(cptr, "???");
85          break;
86      }
87    }
88    return dest;
89  }
90
91  // puts name corresponding to uid into dest string.
92  // if correspondence found, uid number is put into dest string.
93  // -1 returned upon error, 0 upon success
94  int strfuid(uid_t st_uid, char *dest)
95  {
96    struct passwd *stpw;
97    errno = 0; // reset errno first as per readdir's man page suggestion
98    if( !(stpw = getpwuid(st_uid)) )
99    {
100     if( errno == 0 )  // no uid - name translation
101     {
102       snprintf(dest, BUFFSIZE, "%d", st_uid);
103       return 0;
104     }
105     else
106     {
107       fprintf(stderr, "Error getting UID name from %d:%s\n", st_uid, \
108         strerror(errno));
109     }
110     return -1;
111   }
112   else
113     snprintf(dest, BUFFSIZE, "%s", stpw->pw_name);
114   return 0;
115 }
116
117 // same as strfuid except with gid
118 int strfgid(uid_t st_gid, char *dest)
119 {
120   struct group *stgr;
121   errno = 0; // reset errno first as per readdir's man page suggestion
122   if( !(stgr = getgrgid(st_gid)) )
123   {
124     if( errno == 0 )  // no uid - name translation
125     {
126       snprintf(dest, BUFFSIZE, "%d", st_gid);
127     }
128     else
129     {
130       fprintf(stderr, "Error getting UID name from %d:%s\n", st_gid, \
131         strerror(errno));
132     }
133     return -1;
134   }
135   else
136     snprintf(dest, BUFFSIZE, "%s", stgr->gr_name);
137   return 0;
138 }
139
140 char *strflnk(char *path, char *dest)
141 {
142   ssize_t len;
143   if( !(len = readlink(path, dest, PATH_MAX)) )
144     fprintf(stderr, "Error reading link %s:%s\n", path, strerror(errno));
```

```
145     else
146     {
147       dest[len] = '\0';
148     }
149     return dest;
150   }
151
152   // prints path meta data to standard output
153   void printstats(char *path, struct stat *st)
154   {
155     char strperm[10];
156     char struid[BUFFSIZE];
157     char strgid[BUFFSIZE];
158     char strmtime[13];
159     char ft;
160     char strlnk[PATH_MAX];
161     char *lnksymb;
162
163     // get filetype character
164     if( (ft = charftype(st->st_mode)) == 'l' )
165     { // get symbolic link path if it exists
166       strflnk(path, strlnk);
167       lnksymb = "-> ";
168     }
169     else
170     { // otherwise make sure these don't print anything
171       *strlnk  = '\0';
172       lnksymb = "";
173     }
174
175     // get permissions string
176     strfperm(st->st_mode, strperm);
177
178     // get uid and gid names
179     strfuid(st->st_uid, struid);
180     strfgid(st->st_gid, strgid);
181
182     // get time string
183     struct tm *tmp = localtime(&(st->st_mtime));
184     strftime(strmtime, sizeof(strmtime), "%b %e %R", tmp);
185
186     printf(" %8ld %6ld %c%9s %3ld %-9s %-9s %8ld %s %s %s%s\n", \
187       st->st_ino, st->st_blocks/2, ft, strperm, st->st_nlink, struid, strgid, \
188       st->st_size, strmtime, path, lnksymb, strlnk);
189   }
190
191   // recursively read directory
192   int rrdir(char *dn)
193   {
194     DIR *dirp;
195     struct dirent *de;
196     char *path;
197
198     // initialize path for this fcn call
199     if( !(path = malloc(strlen(dn)+1)) )
200     {
201       fprintf(stderr, "Error in path malloc: %s\n", strerror(errno));
202       exit(-1);
203     }
204
205     // open directory before attempting to read
206     if( !(dirp=opendir(dn)) )
207     {
208       fprintf(stderr, "Cannot open directory %s: %s\n", dn, strerror(errno) );
209       return -1;
210     }
211
212     errno = 0; // reset errno first as per readdir's man page suggestion
213     while( de=readdir(dirp) ) // loop through directory entries
214     {
215       struct stat st;
216
217       // allocate memory for pathname
```

```
218       if ( !(path = realloc(path, strlen(dn) + strlen(de->d_name) + 2)) )
219       {
220         fprintf(stderr, "Error in path realloc: %s\n", strerror(errno));
221         exit(-1);
222       }
223       // append directory entry to path
224       sprintf(path, "%s/%s", dn, de->d_name);
225
226       // if de refers to . or .., don't print stats
227       if( strcmp(de->d_name, "..") == 0 || strcmp(de->d_name, ".") == 0 )
228         continue;
229
230       // get file meta data
231       if( lstat(path, &st) == -1 )
232       {
233         fprintf(stderr, "Error getting stats from %s: %s\n", path, \
234           strerror(errno));
235         continue;
236       }
237
238       // print meta data to stdout
239       printstats(path, &st);
240
241       // recursively explore child directories
242       if( S_ISDIR(st.st_mode) )
243         rrdir(path);
244       errno = 0;
245     }
246
247     if(errno)
248       fprintf(stderr, "Error reading directory %s: %s\n", dn, strerror(errno));
249     if( closedir(dirp) == -1 )
250       fprintf(stderr, "Error closing directory %s: %s\n", dn, strerror(errno));
251
252     free(path);
253   }
```