Yacine Manseur

09/25/2015

Operating Systems

Problem Set 2

**Example Output:**

Yacine@Yacine-Laptop ~/OS/OS/PS2

$ ./find.exe ~/OS/OS/PS2

2484ebdd/146366987889631739 -rwsr-xr-x 1 Administrators None 6832 2015-09-25 20:44 /home/Yacine/OS/OS/PS2/find.c

2484ebdd/60235645016128151 -rwsr-xr-x 1 Yacine None 72896 2015-09-25 20:40 /home/Yacine/OS/OS/PS2/find.exe

2484ebdd/10414574138492885 -rwsr-xr-x 1 Yacine None 1380 2015-09-25 18:42 /home/Yacine/OS/OS/PS2/find.exe.stackdump

2484ebdd/7318349394676125 -rwsr-xr-x 1 Administrators None 0 2015-09-23 21:43 /home/Yacine/OS/OS/PS2/testPath/doc1.txt

2484ebdd/4503599627569576 -rwsr-xr-x 1 Administrators None 0 2015-09-23 21:44 /home/Yacine/OS/OS/PS2/testPath/doc2.txt

2484ebdd/9007199254940074 -rwsr-xr-x 1 Administrators None 0 2015-09-23 21:44 /home/Yacine/OS/OS/PS2/testPath/doc3.txt

2484ebdd/14636698789152214 drwsr-xr-x 1 Administrators None 0 2015-09-23 21:44 /home/Yacine/OS/OS/PS2/testPath

Yacine@Yacine-Laptop ~/OS/OS/PS2

$ ./find.exe -m 1000 ~/OS/OS/PS2

2484ebdd/10414574138492885 -rwsr-xr-x 1 Yacine None 1380 2015-09-25 18:42 /home/Yacine/OS/OS/PS2/find.exe.stackdump

2484ebdd/7318349394676125 -rwsr-xr-x 1 Administrators None 0 2015-09-23 21:43 /home/Yacine/OS/OS/PS2/testPath/doc1.txt

2484ebdd/4503599627569576 -rwsr-xr-x 1 Administrators None 0 2015-09-23 21:44 /home/Yacine/OS/OS/PS2/testPath/doc2.txt

2484ebdd/9007199254940074 -rwsr-xr-x 1 Administrators None 0 2015-09-23 21:44 /home/Yacine/OS/OS/PS2/testPath/doc3.txt

2484ebdd/14636698789152214 drwsr-xr-x 1 Administrators None 0 2015-09-23 21:44 /home/Yacine/OS/OS/PS2/testPath

Yacine@Yacine-Laptop ~/OS/OS/PS2

$ ./find.exe -m -1000 ~/OS/OS/PS2

2484ebdd/146366987889631739 -rwsr-xr-x 1 Administrators None 6832 2015-09-25 20:44 /home/Yacine/OS/OS/PS2/find.c

2484ebdd/60235645016128151 -rwsr-xr-x 1 Yacine None 72896 2015-09-25 20:40 /home/Yacine/OS/OS/PS2/find.exe

Yacine@Yacine-Laptop ~/OS/OS/PS2

$ ./find.exe -u Yacine ~/OS/OS/PS2

2484ebdd/60235645016128151 -rwsr-xr-x 1 Yacine None 72896 2015-09-25 20:40 /home/Yacine/OS/OS/PS2/find.exe

2484ebdd/10414574138492885 -rwsr-xr-x 1 Yacine None 1380 2015-09-25 18:42 /home/Yacine/OS/OS/PS2/find.exe.stackdump

Yacine@Yacine-Laptop ~/OS/OS/PS2

$ ./find.exe -u 1002 ~/OS/OS/PS2

2484ebdd/60235645016128151 -rwsr-xr-x 1 Yacine None 72896 2015-09-25 20:40 /home/Yacine/OS/OS/PS2/find.exe

2484ebdd/10414574138492885 -rwsr-xr-x 1 Yacine None 1380 2015-09-25 18:42 /home/Yacine/OS/OS/PS2/find.exe.stackdump

Yacine@Yacine-Laptop ~/OS/OS/PS2

$ ./find.exe -u Administrators ~/OS/OS/PS2

2484ebdd/146366987889631739 -rwsr-xr-x 1 Administrators None 6832 2015-09-25 20:44 /home/Yacine/OS/OS/PS2/find.c

2484ebdd/7318349394676125 -rwsr-xr-x 1 Administrators None 0 2015-09-23 21:43 /home/Yacine/OS/OS/PS2/testPath/doc1.txt

2484ebdd/4503599627569576 -rwsr-xr-x 1 Administrators None 0 2015-09-23 21:44 /home/Yacine/OS/OS/PS2/testPath/doc2.txt

2484ebdd/9007199254940074 -rwsr-xr-x 1 Administrators None 0 2015-09-23 21:44 /home/Yacine/OS/OS/PS2/testPath/doc3.txt

2484ebdd/14636698789152214 drwsr-xr-x 1 Administrators None 0 2015-09-23 21:44 /home/Yacine/OS/OS/PS2/testPath

**Appendix:**

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 2

// find.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <dirent.h>

#include <string.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <pwd.h>

#include <grp.h>

#include <time.h>

int find (int uid, int mtime, char\* currentDir)

{

DIR \*dirp;

struct dirent \*de;

struct stat st;

long nodeSize;

char \*sizeType;

char \*dirPath[1024];

if (!(dirp=opendir(currentDir)))

{

fprintf(stderr, "Cannot open directory %s: %s\n", currentDir, strerror(errno));

return -1;

}

while (de=readdir(dirp))

{

if (strcmp(de->d\_name, "..") == 0)

continue;

if(strcmp(de->d\_name, ".") == 0)

continue;

char nodePathName[255] = "\0";

// Get the path name of the node

if(strcmp(de->d\_name, ".") == 0)

{

strncat(nodePathName, currentDir, sizeof(nodePathName) + sizeof(currentDir));

}

else

{

strncat(nodePathName, currentDir, sizeof(nodePathName) + sizeof(currentDir));

strncat(nodePathName, "/", sizeof(nodePathName) + sizeof("/"));

strncat(nodePathName, de->d\_name, sizeof(nodePathName) + sizeof(de->d\_name));

}

if (de->d\_type == DT\_DIR)

find(uid, mtime, nodePathName);

// Get stats for current node

if (lstat(nodePathName, &st) == -1){

fprintf(stderr, "Error getting link status for %s\n", nodePathName);

continue;

}

// Omit entries that don't correspond to the uid if it was given

if (uid != -1 && uid != st.st\_uid)

continue;

// Omit entries if mtime is given

if(mtime > 0 && (time(NULL) - st.st\_mtime) < mtime)

continue;

else if (mtime < 0 && (time(NULL) - st.st\_mtime) > (-1\*mtime))

continue;

// Get owner of the node

struct passwd \*user = getpwuid(st.st\_uid);

char \*owner;

char \*groupOwner;

if (user->pw\_name)

owner = user->pw\_name;

else

sprintf(owner, "%d", &st.st\_uid);

// Get group owner of the node

struct group \*gid = getgrgid(st.st\_gid);

if (gid->gr\_name)

groupOwner = gid->gr\_name;

else

sprintf(groupOwner, "%d", &st.st\_gid);

// Get modification time

struct tm \*tm;

char timestamp[256];

tm = localtime(&st.st\_mtime);

strftime(timestamp, sizeof(timestamp), "%Y-%m-%d %H:%M", tm);

// Get the size of the node in bytes

// for block/char special device nodes print the raw device number in hexadecimal

if((de->d\_type == DT\_BLK) || de->d\_type == DT\_CHR)

{

sizeType = "%x";

nodeSize = st.st\_rdev;

}

else

{

sizeType = "%ld";

nodeSize = st.st\_size;

}

// Get file type

char ft = '-';

// Test for a block special file

if(S\_ISBLK(st.st\_mode))

ft = 'b';

// Test for a character special file

else if (S\_ISCHR(st.st\_mode))

ft = 'c';

// Test for a directory

else if (S\_ISDIR(st.st\_mode))

ft = 'd';

// Test for a pipe or FIFO special file

else if (S\_ISFIFO(st.st\_mode))

ft = 'p';

// Test for a regular file

else if (S\_ISREG(st.st\_mode))

ft = '-';

// Test for a symbolic link

else if (S\_ISLNK(st.st\_mode))

ft = 'l';

// Test for a socket

else if (S\_ISSOCK(st.st\_mode))

ft = 's';

// Create string for type of node and its permissions mask

char \*filePermissions = malloc(11);

filePermissions[0] = ft;

filePermissions[1] = (st.st\_mode & S\_IRUSR) ? 'r' : '-';

filePermissions[2] = (st.st\_mode & S\_IWUSR) ? 'w' : '-';

filePermissions[3] = (st.st\_mode & S\_IXUSR) ? 'x' : '-';

filePermissions[4] = (st.st\_mode & S\_IRGRP) ? 'r' : '-';

filePermissions[5] = (st.st\_mode & S\_IWGRP) ? 'w' : '-';

filePermissions[6] = (st.st\_mode & S\_IXGRP) ? 'x' : '-';

filePermissions[7] = (st.st\_mode & S\_IROTH) ? 'r' : '-';

filePermissions[8] = (st.st\_mode & S\_IWOTH) ? 'w' : '-';

filePermissions[9] = (st.st\_mode & S\_IXOTH) ? 'x' : '-';

filePermissions[10] = '\0'; // Get rid of ugly weird characters

// Three special permissions on files and directories:

// Sticky bit

if(st.st\_mode & S\_ISVTX)

{

// Sticky bit and other execute are both set

if(st.st\_mode && S\_IXUSR)

filePermissions[9] = 't';

// Sticky bit is set, but other execute is not set

else

filePermissions[9] = 'T';

}

// SetGID

if(st.st\_mode & S\_ISGID)

{

// SGID and group execute are both set

if(st.st\_mode && S\_IXGRP)

filePermissions[6] = 's';

// SGID is set, but group execute is not set

else

filePermissions[6] = 'S';

}

// SetUID

if(st.st\_mode & S\_IXUSR)

{

// SUID and user execute are both set

if(st.st\_mode && 0100)

filePermissions[3] = 's';

// SUID is set, but user (owner) execute is not set

else

filePermissions[3] = 'S';

}

// Check if the file is a symbolic link

char symbolicLinkContent[1024] = "\0";

if(S\_ISLNK(st.st\_mode))

{

char buf[1024];

if(readlink(nodePathName, buf, sizeof(buf)) == -1)

{

fprintf(stderr, "Unable to read contents of symbolic link: %s\n", errno);

continue;

}

strcpy(symbolicLinkContent, buf);

}

char outputLine[256] = "%04x/%llu\t %s %ld %s\t %s\t ";

strcat(outputLine, sizeType);

strcat(outputLine, "\t %s %s");

printf(outputLine, st.st\_dev, st.st\_ino, filePermissions, st.st\_nlink, owner, groupOwner, nodeSize, timestamp, nodePathName);

if(strcmp(symbolicLinkContent, "\0") == 0)

printf("\n");

else

printf("-> %s\n", symbolicLinkContent);

}

closedir(dirp);

return 0;

}

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

{

int opt = -1;

int uid = -1;

struct passwd \*pwd;

int mtime = 0;

char\* startingPath = ".";

while ( (opt = getopt(argc, argv, "u:m:")) != -1)

{

switch (opt)

{

case 'u':

// Check if input is the name or the uid number

if (sscanf(optarg, "%d", &uid) == 1)

{

}

else

{

if((pwd = getpwnam(optarg)) == NULL)

{

fprintf(stderr, "User %s cannot be found.\n", optarg);

return -1;

}

uid = pwd->pw\_uid;

}

break;

case 'm':

if ( atoi(optarg) == 0)

{

fprintf(stderr, "Usage: incorrect argument. Must be an integer.\n");

return -1;

}

else

mtime = atoi(optarg);

break;

case '?':

if (optopt == 'u' || optopt == 'm')

{

fprintf(stderr, "Usage: -%c missing argument.\n", optopt);

return -1;

}

else

{

fprintf(stderr, "Usage: %s [-u user] [-m mtime]\n", argv[0]);

return -1;

}

break;

default:

fprintf(stderr, "Usage: %s [-u user] [-m mtime]\n", argv[0]);

return -1;

}

}

if (optind < argc)

{

startingPath = argv[optind];

}

if (find(uid, mtime, startingPath) == -1)

return -1;

return 0;

}