Programming Project #2
CS4352 Operating Systems

Due Date: 4/22, 11:59 p.m. Please submit via Blackboard. Late submissions are accepted till 4/29, 11:59 p.m., with 10% penalty each day.

In this programming project, you are asked to implement a simplified version of the 'find' utility on Linux/Unix-like system. This assignment assists you for better understanding of file systems design, how to use the Linux's system calls, and enhancing programming skills and experience with programming on a Unix-like environment.

Description

The find utility is used to locate files on a Unix or Linux system. find will search any set of directories you specify for files that match the supplied search criteria. You can search for files by name, owner, group, type, permissions, date, and other criteria. The search is recursive in that it will search all subdirectories too. The syntax looks like this:

$find *where-to-look criteria what-to-do*

Requirements

You need to implement the following functionalities (#4 is for an extra credit). You can compare the output of your program with the output of the standard find utility provided on Linux.

1. find where-to-look
2. find where-to-look criteria
    a. find where-to-look -name <specified name>
    b. find where-to-look -mmin <specified number of minutes>
    c. find where-to-look -inum <specified i-node number>
3. find where-to-look criteria -delete
4. find where-to-look criteria -exec command **(an optional extra credit)**
5. Develop a Makefile to automate the compilation process.

Details of Each Functionality

1. find where-to-look

This will display the pathnames of all files in the specified directory and all subdirectories. e.g. (if no directory specified, the default is the current working directory)

$ find Document

You will get the output like:

Document/file1
Document/file2
Document/subfolder/file3

2. find where-to-look criteria

2.1 find where-to-look -name <specified name>

This will search the specified directory (where-to-look) and all subdirectories for any files named <specified name> and display their pathnames. e.g.

$ find Document –name foo

Here we are using the criterion -name with the argument foo to tell find to perform a name search for the filename foo. The output might look like this:

Document/wpollock/foo
Document/ua02/foo
Document/foo

If find doesn't locate any matching files, it produces no output.

2.2 find where-to-look -mmin <specified number of minutes>
This will find those files modified with the specified number of minutes ago
You can specify a number "n" to mean exactly n, "-n" to mean less than n, and "+n" to mean more than n.

$ find Document -mmin -10

This is used to locate files modified less than 10 minutes ago

2.3 find where-to-look -inum <specified i-node number>
Find a file that has i-node number n.

$ find Document -inum n

3. find where-to-look criteria -delete
This is an example of usage "find where-to-look criteria what-to-do". This will find files with specified criteria and delete them; e.g.

$ find Document -name foo -delete
$ find Document -mmin -10 -delete

Extra credit:

4. find where-to-look criteria -exec command
This will find files with specified criteria and execute the specified command; e.g.

$ find Document -name foo -exec cat (this should find the file with a name "foo" in the specified directory and output the content of the file by executing the "cat" command on the file; this should be equivalent to "$ find Document -name foo -exec cat {} \; " on the Oak machine)

$ find Document -name foo -exec rm (this should find the file with a name "foo" in the specified directory and delete the file; this should be equivalent to "$ find Document -name foo -exec rm {} \; " on the Oak machine)

$ find Document -name foo -exec mv <a new name> (this should find the file with a name "foo" in the specified directory and rename to a new name; this should be equivalent to

"$ find Document -name foo -exec mv {} <a new name> \;" on the Oak machine)

If you are able to implement and support the above three commands (cat, rm, mv), you can score the extra credit.

Sample Codes and Hints

Source code samples: Please checkout source code samples with executing the following command on the Oak machine:

```
git clone https://github.com/githubyongchen/OS.git
```

If you have already checked out a copy of the repo earlier, you can run the following command to update to the latest source code repo:

```
git pull
```

Before you start, you can create a testing directory under your home directory, this will help you debugging your code and better understanding the routine of each function.
Follow the steps below to create a test directory:

1	mkdir testdir
2	cd testdir
3	touch test1
4	touch test2
5	mkdir dir1
6	mkdir dir3
7	cd dir1
8	touch test3
9	touch test4
10	mkdir dir2
11	cd dir2
12	touch test5
13	touch test6
14	cd ..
15	cd ..
16	cd dir3
17	touch test7
18	touch test8

Table 1, Create a Test Directory

After typing the above 18 commands in Table. 1, you will have a simple directory structure. Then, by typing:

```
cd

    tree testdir
```

you will get a tree of all the files, as shown in the below:

```
testdir/
|-- dir1
|   |-- dir2
|   |   |-- test5
|   |   `-- test6
|   |-- test3
|   `-- test4
```

```
|-- dir3
|   |-- test7
|   `-- test8
|-- test1
`-- test2
```

Figure 1. Tree structure of all the files in a directory

This is a typical directory structure in Linux file system. From the Fig.1, we can see that there are files and ***subdirectory within a directory***. Therefore you can imagine that the find utility is probably a ***recursive*** routine.

But let's see what we missed in a directory,

ls –al –R testdir

you will see something like:

```
jialin@jaln:~$ ls -al -R testdir/
testdir/:
total 24
drwxr-xr-x  4 jialin jialin  4096 2013-04-14 00:53 .
drwxr-xr-x 73 jialin jialin 12288 2013-04-14 01:50 ..
drwxr-xr-x  3 jialin jialin  4096 2013-04-14 00:52 dir1
drwxr-xr-x  2 jialin jialin  4096 2013-04-14 00:53 dir4
-rw-r--r--  1 jialin jialin     0 2013-04-14 00:52 test1
-rw-r--r--  1 jialin jialin     0 2013-04-14 00:52 test2

testdir/dir1:
total 12
drwxr-xr-x 3 jialin jialin 4096 2013-04-14 00:52 .
drwxr-xr-x 4 jialin jialin 4096 2013-04-14 00:53 ..
drwxr-xr-x 2 jialin jialin 4096 2013-04-14 00:53 dir2
-rw-r--r-- 1 jialin jialin    0 2013-04-14 00:52 test3
-rw-r--r-- 1 jialin jialin    0 2013-04-14 00:52 test4

testdir/dir1/dir2:
total 8
drwxr-xr-x 2 jialin jialin 4096 2013-04-14 00:53 .
drwxr-xr-x 3 jialin jialin 4096 2013-04-14 00:52 ..
-rw-r--r-- 1 jialin jialin    0 2013-04-14 00:53 test5
-rw-r--r-- 1 jialin jialin    0 2013-04-14 00:53 test6

testdir/dir4:
total 8
drwxr-xr-x 2 jialin jialin 4096 2013-04-14 00:53 .
drwxr-xr-x 4 jialin jialin 4096 2013-04-14 00:53 ..
-rw-r--r-- 1 jialin jialin    0 2013-04-14 00:53 test7
-rw-r--r-- 1 jialin jialin    0 2013-04-14 00:53 test8
```

Figure 2. All Contents in A Directory

Notice that there are some hidden files starting with . or ..

Be careful about that!

Studying the basic directory structure and the contents in a directory is a good start for you to implement the find utility. The following code (Table 2) recursively prints all the file names in a directory, you may need to learn and pick up some useful system calls from the codes and then implement other find utilities.

```
0  /*
1  *A function that recursively print all file names
2  *Input: directory name, i.e., char * sub_dir
```

```
3   *Output: all file names
4   */
5   void read_sub (char* sub_dir)
6   {
7     DIR *sub_dp=opendir(sub_dir);//open a directory stream
8     struct dirent * sub_dirp;//define
9     struct stat buf;//define a file status structure
10    char temp1[]=".";
11    char temp2[]="..";
12    char temp3[]="/";
13    if(sub_dp!=NULL)
14    //check whether the directory stream is opened successfully
15      {
16         // read one entry each time
17         while((sub_dirp=readdir(sub_dp))!=NULL)
18          {
19           //print the first entry, a file or a subdirectory
20           printf("%s\n",sub_dirp->d_name);
21
22        //check whether the first entry is a subdirectory
23          char * temp =sub_dirp->d_name;
24
25          //to avoid recursively searching . and .. in the directory.
26          if(strcmp(temp,temp1)!=0&&strcmp(temp,temp2)!=0)
27          {
28  char *temp_sub=temp3;
29  temp_sub=strcat(temp_sub,temp);
30           //now you add the / in front of the entry's name
31  char* temp_full_path=malloc(sizeof(char)*2000);
32  temp_full_path=strcpy(temp_full_path,sub_dir);
33         strcat(temp_full_path,temp_sub);
34           //now you get a full path, e.g., testdir/dir1 or testdir/test1
35
36         // try to open
37         DIR * subsubdp=opendir(temp_full_path);
38  //if not null, means we find a subdirectory, otherwise, its just a file
39         if(subsubdp!=NULL){
40        //close the stream, because we will reopen it in the recursive call.
41  closedir(subsubdp);
42          read_sub(temp_full_path);//call the recursive function call.
43          }
44        }
45     }//end of while loop
46        closedir(sub_dp);//close the steam
47     }
48     else
49     {
50         printf("cannot open directory\n");
51         exit(2);
52     }
53  }
```

Table 2. Sample Codes for Printing All File Names

There are several system calls you need to know in your program.

1. DIR *opendir(const char **name*) at line 7, 37
   The opendir() function opens a directory stream corresponding to the directory name, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.
2. struct dirent *readdir(DIR *dirp) at line 17

The readdir() function returns a pointer to a 'dirent' structure representing the next directory entry in the directory stream pointed to by 'dirp'. It returns NULL on reaching the end of the directory stream or if an error occurred.

3. The 'dirent' structure defines a file system independent directory entry, which contains information common to directory entries in different file system types. The dirent structure is shown in Figure 3 as follows:

```
struct dirent {
    ino_t          d_ino;       /* inode number */
    off_t          d_off;       /* offset to the next dirent */
    unsigned short d_reclen;    /* length of this record */
    unsigned char  d_type;      /* type of file; not supported
                                   by all file system types */
    char           d_name[256]; /* filename */
};
```

Figure 3. Strucutre of Dirent

Note that we have used the **d_name** in the sample codes at line 20 and 23.

4. Another system call you need is stat()
   int stat(const char *path, struct stat *buf);
   This system call returns information about a file. No permissions are required on the file itself.Stat is also a struct in Linux system. The structure of stat is shown in the Figure 4:

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

Figure 4. Structure of Stat

- Why do we need the stat() call and stat struct? Since you are required to implement a find utility like

  $ find where-to-look -inum <specified **i-node** number>

  The stat contains the information of **i-node** and other needed stuff.
- How to use it? Recall that we have the path name in line 20, and we defined a file status structure in line 9. Then to print the file size in bytes, we can use the following codes:

```
struct stat buf;//define a file status structure
if(stat(sub_dirp->d_name,&buf)==0)
printf("%d ", (int)buf.st_size);
```

Figure 5. Print File Status

5. Remove() will be needed
   To implement the 'find where-to-look criteria –delete' function, you will need remove():

   remove(file_name);

6. For parsing options and arguments, you can directly manipulate *argv*, or you can consider using *getopt()*/*getopt_long()*/*argp_parse()* supplied by the GNU C library. A sample code *getopt.c* is provided as an example of showing how *getopt()* is used. You can always Google to find more details of these command line options and arguments parsing functions.

Expected Submission:

You should submit a single tarball/zipped file through the Blackboard containing the following, and please name your submission file starting with LastName_FirstName_Project#2.

- Source codes for requirements #1, #2, and #3 (and #4 if you take this extra credit).
- A Makefile to automate the compilation process.
- Output files for your test cases.

Grading Criteria:

Please note that, if needed, we may request an in-person, 5-10 mins quick demo from you.

Percentage %	Criteria
10%	Inline comments to briefly describe your code
20%	Implement the 'find where-to-look' functionality and have correct results
30%	Implement the 'find where-to-look criteria' functionality and have correct results
20%	Implement the 'find where-to-look criteria -delete' functionality and have correct results
10% (extra credit)	Implement the 'find where-to-look criteria -exec command' functionality and have correct results
10%	Carry out test cases to evaluate all functionalities implemented (you can be creative in designing test cases)
10%	Correctness and features of Makefile (minimum features of automating compilation and cleaning up)

Reference Materials:

- Linux system programming: Book: *Linux System Programming*
  Online:
  Tutorial for Beginners, http://www.ee.surrey.ac.uk/Teaching/Unix/
  Advanced Linux Programming, http://www.advancedlinuxprogramming.com/alp-folder/advanced-linux-programming.pdf
- Linux man pages, http://linux.die.net/man/
- Stackoverflow, http://stackoverflow.com/
- Codewiki, http://codewiki.wikidot.com/start
- Makefile**:**
  - http://www.gnu.org/software/make/manual/make.pdf or
  - http://www.gnu.org/software/make/manual/html_node/index.html