CS-392 Systems Programming — Spring 2023

Assignment 3 · Permission Finder

Lecturer: Shudong Hao — *Date: See Canvas*

1 Objective

For this assignment, you will be creating a program that finds files with a specified set of permissions. This program will recursively search for files whose permissions match the user specified permissions string under a specific directory.

Permissions strings are to be formatted similarly to how the command `ls` formats them. In UNIX systems, the leftmost character specifies the type of file (`d` for directory, `l` for symlink, *etc*). The permission string passed as command-line argument will only contain the rightmost 9 characters, such as `rwxr--r--`.

2 Implementation Specifics

You will be traversing the directory tree using the function `readdir()` by first opening a directory by `opendir()`, and for every file, checking the permissions on it using `stat()`. The function `stat()` allows you to check what type of file it is (regular file, directory, symlink, block special, *etc*), and handle each accordingly.

2.1 Step 1: Validate Input

You can invoke the program like the following:

```
./pfind <directory> <pstring>
```

You can safely assume the input of the command is always like this, so no need to check `argc`. You can also assume `directory` will always be a real directory that exists, not a regular file. The only thing you need to check from the command is `pstring` (see below).

2.2 Step 2: Verify and Resolve Permissions String

You will be required to ensure that the permissions string is in proper format. That is, each of the 9 characters must either be a dash (`-`) or one of the characters `rwx`, in the proper position.

Some examples of valid permissions strings:

```
rwxrwxrwx
---------
rw-r--r--
rwx------
```

Some examples of invalid permissions strings:

```
abcdefghi
xrwxrwxrw (notice it's the right characters in the wrong places)
---rrr---
-
rwxrwxrwxrwxrwxrwxrwxrwxrwx
```

If an invalid permissions string is passed (denoted here as `pstring`), the following error message should be printed to standard error, and an exit status of `EXIT_FAILURE` should be returned:

```
Error: Permissions string 'pstring' is invalid.
```

The `'pstring'` in the output above should be replaced by whatever the user typed.

2.3 Step 3: Recursively Navigate Directory Tree

Get yourself comfortable navigating the directory tree using `opendir()`, `readdir()`, and `closedir()`. Each time you `readdir()`, try calling `stat()` on that file, and reading the permissions, the filename, *etc*. Then, start matching the permissions against the target.

2.4 Step 4: Put It All Together

Your general program flow should be as follows:

(1) Initialize the program by checking the permission string is valid;

(2) Get the target permissions from the permissions string;

(3) Start recursing through the directories, printing out files you encounter where the permissions match the target permissions.

3 Example Executions

For these examples, I will be operating on a directory tree with the following format:

```
test_dir
├── subdir1
│   ├── file1
│   └── file2
└── subdir2
    ├── file1
    └── file2
```

The files will have the following permissions:

```
test_dir:
drwxrwxrwx subdir1
drwxr-xr-x subdir2

test_dir/subdir1:
---------- file1
-rw-r--r-- file2

test_dir/subdir2:
-rw-r--r-- file1
-rw-r--r-- file2
```

3.1 Sample Run

```
$ ./pfind test_dir badpermis
Error: Permissions string 'badpermis' is invalid.

$ ./pfind test_dir rw-r--r--
/home/user/test_dir/subdir1/file2
/home/user/test_dir/subdir2/file2
/home/user/test_dir/subdir2/file1

$ ./pfind test_dir --x--x--x
<no output>
```

Note in the last case it literally means "no output", so do not print a string `no output`!

If I create a new directory `danger_dir` with permissions `---------` inside my home directory, and try to run `pfind` on it, it will produce the following output:

```
$ ./pfind ~/danger_dir --x--x--x
Error: Cannot open directory '/home/user/danger_dir'. Permission denied.
```

Good to Know

- The output order of matched files does not matter;
- You only need to print out matching regular files; no need to print out matching directories;
- In the 3rd test in the sample run, it literally means no output at all. Do not print `<no output>`!

Deliverable

Submit a single `pfind.c`. Do not zip it. Your code must compile successfully to receive credits.