CS 392 Spring 2024 Systems Programming Homework 3

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 right-most 9 characters, such as `rwxr--r--`.

2 Task 1: Validating 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).

You will be required to ensure that the permission 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 permission 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.

3 Task 2: Finding Files with Specified Permission

Once the permission string has been validated, 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.

If a file's permission matches the `<pstring>`, you will print the absolute path to that file. You only need to print out matching regular files; no need to print out matching directories. Also note that in the output, the order of matched files does not matter.

4 Tester

To help you with testing, we provided a tester file `tester_pfind_arm` for ARM-64 machines, and `tester_pfind_x86` for X86-64 machines. In the following, we use `tester_pfind_arm` as an example:

```
$ ./tester_pfind_arm -h
```

By default, the tester will assume the source code file `pfind.c` is in the same directory, so it will compile `./pfind.c`. However, if you want to specify a path to the code, you can use `-f` flag:

```
$ ./tester_pfind_arm -f <PATH_TO_PFIND.C> -t <TASK>
```

To help you examine the output, we added a feature called **paging**, which shows you one test case at a time. To continue to the next test case, press any key. If you don't want to use this feature but rather see the output once for all, use `-c` flag (c for "continuously"), such as:

```
$ ./tester_pfind_arm -t all -c
```

If you are working only with the tester, you don't need to compile your `pfind.c` – the tester does it for you! It compiles your code every time before testing.

5 Grading

The homework will be graded based on a total of 100 points:

- Task 1 (30 pts): 10 test cases in total, **3** points each;
- Task 2 (70 pts): 10 test cases in total, **7** points each.

After accumulating points from the testing above, we will inspect your code and apply deductibles listed below. The lowest score is 0, so no negative scores.

- **-100:** the code does not compile, or executes with run-time error;
- **-100:** the code is generated by AI, and/or through reverse engineering on the tester;
- **-50:** the testing directory is hardcoded;
- **-50:** did not use all of the following structure (including related function calls):
  - `DIR*`, `struct dirent*`, and `struct stat`;
- **-50:** did not use any of the permission macros, such as `S_IRUSR`, *etc*.
- **-30:** memory leak through valgrind (only "definitely lost" category);

- **-10:** no pledge and/or name in C file.

Earlybird Extra Credit: 2% of extra credit will be given if the homework is finished two days before the deadline. For specific policy, see syllabus.

Deliverable

Submit a single `pfind.c`.