CS 392: Homework Assignment 4
Due: March 27, 11:55pm

Philippos Mordohai
Department of Computer Science
Stevens Institute of Technology
Philippos.Mordohai@stevens.edu

Collaboration Policy. Homeworks will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems. Use of the Internet is allowed, but should not include searching for previous solutions or answers to the specific questions of the assignment.

Late Policy. No late submissions will be allowed without consent from the instructor. If urgent or unusual circumstances prevent you from submitting a homework assignment in time, please e-mail me explaining the situation.

Objective

In this assignment, we learn how to parse directories and obtain information about files and the filesystem.

Preparing the Filesystem

Create a new directory in your Linux environment using `mkdir` and a name of your choice. Within that directory, make at least two subdirectories. In each of the new directories, including the one you made first, create a few files of different sizes using commands like `man chown > mchown.txt`, but any other option is allowed.

Change the permissions of some of these files so that some are writable by the user and some are not. This can be accomplished with `chmod`.

Functionality 1. Find largest files in flat directory (25 points)

Create a function that can find the largest writable and non-writable file by the user in a directory without subdirectories (flat directory). This function will be part of a program named `maxfile` that can be invoked like this:

```
./maxfile <directory>
```

You will be reading the contents of the directory passed as the only argument to your program using the function `readdir()`. Begin by opening the directory using `opendir()`, and obtain information for every file using `stat()`. `stat()` allows you to check whether a file is a regular file or a directory. (In this assignment, hard and symbolic links, special files, etc. will be ignored.) At the end, close the directory using `closedir()`.

For Functionality 1, the input will be assumed to be a flat directory and subdirectories should not be traversed. Regular files must be separated into those for which the *user* has permission to write and those for which the user does not have permission to write. For each of the two types return to the caller:

(1) the size of the largest file. `stat()` can be used to find the size of a file.

(2) the filename of the largest file.

Functionality 2. Find largest files recursively (25 points)

Extend your previous function to operate on an arbitrary directory with subdirectories. The directory system can have arbitrary depth. Think about whether the order the directories are visited matters and how to reuse the code from Functionality 1.

When combining results from multiple directories, you do not need to build up the path to the largest files in the output strings. Just the *basenames* are sufficient.

Ideally, you should submit a program that can work recursively on the directory tree; this will also provide Functionality 1.

Functionality 3. Compute disk usage in flat directory (25 points)

Create a function that can compute the total disk usage of the contents of a flat directory. This functionality will be part of the same program as above, and in fact can be implemented by the same function as Functionality 1. The output should match that of the following Unix command:

```
du -b <directory>
```

Your code will still be invoked as above:

```
./maxfile <directory>
```

You will need the same commands, but the goal is now different. Keep in mind that empty directories take up disk space.

Functionality 4. Compute disk usage recursively (25 points)

Extend your previous function to operate on an arbitrary directory with subdirectories. The directory system can have arbitrary depth.

Ideally, you should submit a program that can work recursively on the directory tree; this will also provide Functionality 3.

Integration

It is acceptable to submit one function that provides Functionalities 2 and 4. If you are uncertain about either one or the recursive implmentations, however, it is safer to separate the code into multiple functions and specify what should be expected of your code in the report.

At the end your program should print five outputs:

(1) The size of the largest writable file.

(2) The filename of the largest writable file.

(3) The size of the largest non-writable file.

(4) The filename of the largest non-writable file.

(5) The total disk usage of the directory that was provided as argument.

Suggestions

(1) Look carefully at all examples in the notes on filesystems.

(2) If you do not have sufficient permissions to access a file or directory, print an appropriate message and continue processing.

(3) You can assume that symbolic or hard links and special files do not exist in the directories you are working on. You do not need to check for them. Only regular files and directories will be included.

(4) Do not process '.' and '..' for obvious reasons. Hidden files that begin with '.' should be processed, however.

(5) You can break ties in size among file arbitrarily, i.e. by always keeping the current maximum or always replacing it with the new file found.

Requirements

(1) Your program should compile and work correctly. Compiling and performing part of the requirements is better than not compiling.

(2) It is your responsibility to test all cases, such as files and directories for which you have insufficient permissions.

(3) Use comments where necessary to explain what you are doing. No comments or over-commenting are both bad.

(4) Do not leak memory. Free all allocated memory and close all opened files.

Deliverables

In a zip file, submit the following:

(1) A shell script for making a directory, at least 2 subdirectories included in it, with at least 4 files in each subdirectory and at least 2 files in the top-level new directory. All paths must be relative.

(2) A C file named `maxfile.c` implementing the above functionalities.

(3) A pdf file with brief explanations of your approach and a statement whether the recursive implementation works or not.