Assignment #4

Decomposition Outline Due: February 21, 2017 11:59pm
Due: February 24, 2017 11:59pm

Overview

This assignment requires interaction with the filesystem (via system calls), traversing a portion of the directory hierarchy, and managing scarce resources. For this assignment you will write a program that recursively prints directory contents in a tree-like format.

Decomposition Outline Document

You must submit a program decomposition outline by the posted due date. Be certain to include enough detail to make it clear that you have given careful thought to the construction of this program.

Be certain to address how you intend to manage file descriptors so that your program will not run out.

File Tree

Your program, named `tree`, must take as command-line arguments optional switches and any number of file/directory names. For each given file name, your program must only print that name. For each directory name, your program must print the contents of that directory (and, recursively, each subdirectory) in a tree-like format. If no file names or directory names are given, then list the contents of the current directory.

As an example, your program output should look as follows:

```
% ./tree tmp
tmp
|-- alpha
|-- bob
|-- tmp
|   |-- alpha
|   |-- bob
|   |-- burrito
|   |-- tmp
|   |   |-- bob
|   |-- zebra
|-- zebra
```

The format of your program's output **must** match that shown above. This tree shows the contents of the `tmp` directory and each subdirectory below it. The contents of each directory must be **sorted** alphabetically.

Note: When processing a directory's contents, do not recursively follow the two special files "." and ".." (consider why not). Also, do not follow symbolic links (but do print the link name itself).

Command-line Arguments

Your program must support the two following command-line switches. If given, these switches will immediately follow the program name (and precede any specified files). Your program must allow the switches to be provided in any order and must allow them to be combined into a single switch (e.g., `tree -la tmp`).

- Hidden Files [`-a`]: By default, your program should not include "hidden" files (those preceded by a ".") in its listing. The `-a` switch causes "hidden" files to be processed (except for the two special files "." and "..").
- Long Format [`-l`]: The `-l` switch causes the access permissions for each file to be printed. The format of the access permissions should be the same as the `ls` command. For example,

```
[drwx------] tmp
|-- [-rw-------] alpha
|-- [-rw-------] bob
|-- [drwx------] tmp
|   |-- [-rw-------] alpha
```

```
|   |-- [-rw-------] bob
|   |-- [drwx------] burrito
|   |-- [drwx------] tmp
|   |   |-- [-rw-------] bob
|   |-- [-rw-------] zebra
|-- [-rw-------] zebra
```

Resource Management

All programs must properly manage the resources they use. In this respect, systems programs differ only in that the resources they must manage are often more scarce than those used by general applications. To address this issue, one will often use more readily available resources (e.g., memory) to avoid exhausting other resources.

If you are not careful in your implementation, then your `tree` program will likely run out of file descriptors and be unable to continue. Do not allow this to happen. Your program will be tested with a very deep, very contrived directory structure (i.e., it will be much deeper than the number of files that a process may open) or with the limit for the number of open files set artificially low (you should do this for your testing).

Useful Functions

You might find the following functions to be of use (this is not to say that you will need them all or that you will not use others)

chdir, fchdir, opendir, readdir, stat, lstat

Submit

- All source code.
- A Makefile that will build your program.

Grading

Feature	Percentage
Decomposition Outline	10
Basic Functionality	30
Format	20
Hidden Files	10
Long Format	10
Resource Management (memory, file descriptors)	10
Error Handling	10