SFWRENG 3SH3: Operating Systems
Lab 5: File System

Dr. Borzoo Bonarkdarpour

1 Background

In this lab assignment you will study how the UNIX file system is structured and which data structures are used to represent different objects and which system calls work with these data structures.

It is assumed that you are familiar with the following UNIX commands:

- `df`
- `ls`
- `ln`

2 Inodes

In a Unix-based file system, the `inode` is a data structure used to represent a filesystem object, which can be one of various things including a file or a directory. Each `inode` stores the attributes and disk block location(s) of the filesystem object's data. Filesystem object attributes may include manipulation metadata (e.g. change, access, modify time), as well as owner and permission data (e.g. group-id, user-id, permissions).

In this assignment, you are to write a program that completes two tasks: (1) reporting file attributes, and (2) traversing directories. That is, if the input argument to the program is a file, the program will print the file's attributes. If the input argument to the program is a directory, the program will traverse the directory recursively.

2.1 File Attributes

In this part of the assignment you will learn about the `stat(2)` system call and the `stat` structure that it returns. Observe that there are several `stat` functions, use the right one in the right place, especially for symbolic links. The `stat` structure contains a number of attributes with information from a file's inode.

You will write a program that, given the name of a file or directory on the command line, reports the following information from the corresponding inode: (1) mode (permissions), (2) number of links, (3) owner's name (string, not numerical), (4) group name (string, not numerical), (5) size in bytes, (6) size in blocks, and (7) last modification time, and (8) name.

It is not necessary to print the mode symbolically like ls does (e.g. `-rwxr-xr-x`). But, if you choose to print it numerically, use octal (e.g., `0755` for the aforementioned symbolic mode). If the file is a symbolic link you do not have to read the link.

2.2 Directory Traversal

Now, extend your program and write a function that traverses the file system from a starting point provided on the command line, similar to `ls -lRa`. The major functions you will need are `opendir(3C)`, `readdir(3C)`, `closedir(3C)`, `chdir(2)`, `getcwd(3C)` and `rewinddir(3C)`.

A natural structure for the program is a function that traverses the list of files in a single directory. When it reaches a directory in the list, it can call itself recursively. To start, just invoke the function with the name of the starting directory.

Always check return values. Most system calls return a value to indicate if they were successful, and if not, the reason for failure. They do fail sometimes! In these cases you should use `perror(3C)` to print a message describing the failure.

You will need to deal with the possibility that you may not have the proper permissions to search or enter certain directories. There are also some other error-like situations that need to be handled properly. In none of these cases should the program need to exit, although it may need to take some special action.

3 Sample Solution Executables

Sample solution Linux and OSX executables can be found here:
`http://www.cas.mcmaster.ca/~borzoo/teaching/17/3SH3/labs/Lab5`

Note that the sample solution does not handle of all errors (e.g., permission denied).

4 Guidelines

You will

- work on this assignment individually
- implement this assignment using C or C++ (you are free to separate your program into multiple source files as you see fit)
- present your implementation and output to your lab TA.

You may present your program in a different section in the same week **only once** throughout the term.