Project 4: File system utilities

Due date: Monday, April 26th 11:59 pm on Canvas.

Note: This is an individual project. Each student is expected to work independently on code.

Description: In this project, you will write four different programs based on various UNIX utilities.

You must submit a tar file containing the source code for each utility on separate files, and a makefile for easy compilation. Because we will be using automated scripts to test your code, it is important that you follow project structure conventions and that the output exactly matches the requirements. Before you submit, you should test that your project behaves as expected when the following commands are typed:

(This is only an example - replace 'smith' with your last name)

```
$ tar xvf p4-smith.tar
$ ls
makefile myls-smith.c mysearch-smith.c mystat-smith.c
mytail-smith.c
$ make
gcc -o myls myls-smith.c
gcc -o mysearch mysearch-smith.c
gcc -o mystat mystat-smith.c
gcc -o mytail mytail-smith.c
$ ls
makefile myls myls-smith.c mysearch mysearch-smith.c mystat
mystat-smith.c mytail mytail-smith.c
```

1. **Stat**: Write your own version of the command line program `stat`, which simply calls the `stat()` system call on a given file or directory. Print out file size, number of blocks allocated, reference (link) count, file permissions, and file inode.

Useful interfaces: stat()

2. **List Files**: Write a program that lists files in the given directory. When called without any arguments, the program should just print the file names. When invoked with the `-l` flag, the program should print out information about each file, such as the owner, group, permissions, and other information obtained from the `stat()` system call. The program should take one additional argument, which is the directory to read, e.g., `myls -l directory`. If no directory is given, the program should just use the current working directory.

Useful interfaces: stat(), opendir(), readdir(), getcwd().

3. **Tail**: Write a program that prints out the last few lines of a file. The program should be efficient, in that it seeks to near the end of the file, reads in a block of data, and then goes backwards until it finds the requested number of lines; at this point, it should print out those lines from beginning to the end of the file. To invoke the program, one should type: `mytail -n file`, where `n` is the number of lines at the end of the file to print.

Useful interfaces: stat(), lseek(), open(), read(), close().

4. **Recursive Search**: Write a program that prints out the names of each file and directory in the file system tree, starting at a given point in the tree. For example, when run without arguments, the program should start with the current working directory and print its contents, as well as the contents of any sub-directories, etc., until the entire tree, root at the CWD, is printed. If given a single argument (of a directory name), use that as the root of the tree instead.

Useful interfaces: you figure it out.

IMPORTANT NOTICE: You are expected to write your own code. Copying from the Internet is not allowed. All programs will be checked for plagiarism. Submitting code that is not your own will result in a FF grade on the course.

Grading Rubric:

Each utility will be tested using automated scripts and by manual inspection. Therefore, it is important that ***all*** *file names match structure described above (2%), and that you provide a makefile for easy compilation (2%)*. Each program is worth 24% of the grade, and will be graded according to the following metrics:

[10%] I/O works as expected (e.g., `stat` prints file size, number of blocks allocated, etc.)

[10%] Correct implementation (e.g., use of APIs such as `stat`, `opendir`, `lseek`, etc)

[4%] Programming style (code is organized and well-commented)