Lab 5: Directories and Files
CMSC 242
Due: Friday, Apr. 8 by 11:59:59pm

Unix systems provide a standard library which contains functions for common tasks. One of the most important features the standard library provides is the ability to navigate directory hierarchies and inspect both the contents and the attributes of a file. In class, we've explored several of these functions and you should have a pretty good idea of how they work. If you need to review, you can pull up a manpage for any of them by typing, for instance,

```
man 3 readdir
```

to get an explanation of how the readdir function works. Almost all of the I/O functions we've talked about are in page 3 of the manual, but if you can't find something in page 3, try page 2.

If you can't remember the name of a function, you can look it up by typing "apropos <keyword>" where <keyword> is a term that describes what the function does. For instance:

```
apropos directory
```

will list all the available man pages on directories.

The `ls` command allows a user to list the files in a directory. When typed with no arguments, it lists the files in the current directory. When typed with the "-l" flag, `ls` provides a "long" listing that gives information (such as permissions, owner, group, and time last modified) for each file. When typed with the "-R" flag, `ls` operates recursively, descending into the subdirectories of the current folder and listing them as well.

In this lab, you will implement your own version of ls. It will be a little different from the standard ls command, but it will give you lots of practice using the directory manipulation functions, the 'stat' function and the stream I/O functions we've covered in class.

We are going to build up the program a step at a time. In each step, we are going to copy the code to a new file and add a new feature. When we are done, you should have several different "myls" programs that do the listing operation a little differently.

Step 1. Getting Started

Create a folder named Lab5. In that folder, create a file named "myls1.c" that will hold the code for your first `ls` program. At the top of your file place the usual comments:

```
//myls1.c
//YOUR NAME
//Spring 2014
//(c) YOUR NAME
```

Step 2. Listing the current directory

Now, in "myls1.c", create a main function for your ls command. In the main function, add code for printing out the names of the files in the current directory. You will probably need to use these functions: opendir, closedir, readdir. Print each name on its own line and print only the name.

You will need to add some #include statements to the top of your code and you will probably also need to declare DIR* and struct dirent* variables.

Don't forget that the filename "." is always a name for the current directory.

As usual, compile and test your code to be sure it works correctly before proceeding.

Step 3. File information

Listing the files is interesting, but not nearly as useful as "ls -l", which gives us lots of information about the file. Copy "myls1.c" to "myls2.c" and modify your code so that in addition to the filename it provides the following information:

uid	(the id number of the owner of the file)
gid	(the id number of the group that owns the file)
mode	(the permissions for the file)
mtime	(the time the file was last modified)
size	(the size of the file in bytes)

Your directory should list files one to a line in exactly the following format:

The name, left justified, in a field 40 characters wide.
The uid, as a decimal integer, in a field 4 digits wide, padded with 0's.
The gid, as a decimal integer, in a field 4 digits wide, padded with 0's.
The mode, as an octal integer, in a field 6 digits wide, padded with 0's.
The size, as a decimal integer, in a field 10 digits wide, padded with spaces.
The mtime, in a field 20 characters wide in the following format, padded with spaces:

```
MM-DD-YYYY hh:mm:ss
```

Where MM is the two digit month (with leading 0 if necessary), DD is the two digit day, YYYY is the four digit year, hh is the two digit hour in coordinated universal time (UTC), mm is the two digit minute, and ss is the two digit second.

You should separate each of these fields with a single tab character.

You will need to use at least the following functions:

```
stat
printf
strftime
```

Compile and test your code.

Step 4. File Types

In Unix, a file can either be a regular file, a directory, a FIFO pipe, a character device, a block device, a symbolic link, or a socket. There is a set of macros we discussed in class that can tell you, from the file mode, whether a file is each of these types. They are listed in the manpage for the "stat" function.

Using these macros, determine if the file is a directory. If so, print a "`/`" character at the end of the filename. It should occur before the tab separating the filename from the uid, but should not be part of the field for the filename. If the file is a symbolic link, print an "@" character after the filename.

Step 5. File Contents

Make a copy of “myls2.c” and name it “myls3.c”.

So far, we've stuck pretty close to the standard “ls” command. Here is where I want you to extend the tool a little.

For regular files, we are going to provide a little more information. Instead of just listing files one on each line, we're going to print THREE lines. On the first line for each entry, print a file listing just as in Step 4. On the next line, print the first line of the file. On the third line, print the last twenty bytes of the file – as hexadecimal digits. If the file isn't twenty bytes long, print as many bytes as possible.

Directories, symbolic links, devices, FIFOs, and sockets should be listed just as they were before (on one line).

You will probably need to use:

fopen
fclose
fgets
fseek

As usual, test your code.

Step 6. Recursion

Make a copy of “myls3.c” and name it “myls4.c”. Create a new function named “list_directory” that takes two parameters: a DIR pointer named “directory” and a C-string named “path”.

The “directory” variable is a pointer to an already opened directory object.   The “path” variable represents the path to that directory.

Modify your code so that the main function is responsible for opening and closing the current directory, but calls “list_directory” to do the rest of the work.

Then, add code that checks whether each entry is a directory. If so, it should open the directory and call list_directory recursively to list the contents of that directory UNLESS the directory is either the “.” or “..” directory (we don't want an infinite recursion to happen).

Compile and test your code.

Step 7. Prettier Recursion

Right now, all the directories are printed at the same level. That makes it hard to tell which files belong to a subdirectory and which are part of the parent directory. Add a new integer parameter to the “list_directory” function named “level”. The level should indicate how deep the recursion is. Then adjust your code so that in front of each entry it prints one tab per level. You do not need to indent the content lines for regular files – just the line that contains the filename.

Compile and test the code again.

Glitter

Make a new file named “myls.c” that contains your glitter. It can be a copy of any of the previous “myls” files. Extend it in a creative and interesting way. One simple idea and easy idea (probably worth only a few glitter points, but a good place to start) is to use your color library from Lab1 to color code files based on their attributes.

To get full glitter, you should either demonstrate a deeper understanding of the function calls that we used to implement this lab by making use of features we didn't already use, or make use of other functions from the standard library that we didn't cover.

Submitting

As usual, create a tarball of your entire Lab5 folder:

```
cd ..
tar czvf Lab5.tar.gz Lab5/
```

Then turn your work in through the course web site:

http://marmorstein.org/~robert/submit/