CMPT 300: Assignment #4 Tutorial

The UNIX File System

Reading a directory

There are several key functions that that can be used to read the contents of directories and to determine various file characteristics. These functions are: ***stat()***, ***lstat()***,***opendir()***, ***readdir()***, and ***closedir()***. Use the Unix man pages to learn more about these functions.

To read a directory, you need to

1. Open a directory using ***opendir()***.
2. Then ***readdir()***.
3. Next get the name of the directory entry with **dp->d_name**. **dp** must be a pointer to a struct **dirent** (a directory entry structure).
4. Declare a **stat** structure (e.g., **buf**), and then use this **buf** and **dp->d_name** to call ***stat()*** or ***lstat()*** as appropriate.
5. ***closedir()***, once you have finished reading a directory it needs to be closed using ***closedir().***

Implementing *ls*

1. Read the man page for ls to get a detailed description of ls's behaviour.
2. **-i** in ***ls*** means print the **inode** (column 0: **st_ino**) number of each file on the leftmost column of output.
3. **-l** in ***ls*** means long-listing. For each file, you need to display (formatted from left to right):
   - the read/write/execute permission of the file (column 1: from **st_mode**);
   - the number of hard-links pointing to the file (column 2: **st_nlink**);
   - the user name of the file owner (column 3: **st_uid**);
   - the group name the owner belongs to (column 4: **st_gid**);
   - the size of the file in bytes (column 5: **st_size**);
   - the date and time the when the file was last modified (column 6: **st_mtime**);
   - the filename (column 7)

4. The output from **-l** needs to be formatted and aligned, just like ***ls***.
   The printf function from <stdio.h> can do most kinds of simple formatting very nicely. You may exert fine control over the formatting of your output by using extra arguments placed between the percent sign (%) and the character that indicates the type of data that you are printing. The most simple extra argument is a number, which defined the "field width" within which the data will be formatted. Normally printf uses only as much space as necessary to format the data, but you can direct it to use a larger field width if you wish. If the data needs more space than the field width that you specify, then printf will use the extra space (it will not discard information to make it fit within the provided width, but will rather use more space).
   For example, to print an integer x in a field width of 11 character positions, use:

   ```
   printf("%11d", x);
   ```

   printf will normally right-justify the data in the provided space, that is, put the extra spaces on the left. To get it to right justify the data instead, just use a negative field width, as in:

   ```
   printf("%-11d", x);
   ```

CMPT 300: Assignment #4 Tutorial

If you wish to print a number with leading 0s as necessary to make up the specified field width (printf normally uses spaces), then just add a 0 before the width specification. This is very helpful for times of the day, as we normally expect to see 12:00, rather than 12: 0 so try:

```
printf("%02d:%02d", hours, minutes)
```

5. **-R** in ***ls***:
   The option -R means recursively list the contents of the directory. It means the program must traverse every subdirectory recursively and list the entries of all these directories. If the –*R* option is invoked, your **myls** should also output the name of **each directory** before showing the contents of that directory, just like the ls command does. Note that when ls is doing a recursive traversal of a directory structure it prints the names of all of the files and directories in each directory, and then for each directory prints its names and recursively displays the contents of the directory. A very simple way to accomplish this in your program is to read the directory twice, the first time printing the names (and associated details if the -l flag or -i flag or both are given), and then on the second time through just dealing with those entries that are directories. For this purpose, the ***rewinddir*** function can be very helpful.

<u>Handling symbolic links in *ls*</u>

When you encounter a symbolic link, (e.g., def is a symbolic link to the directory **old**), ***ls -l*** should output something like the following:

```
lrwxrwxrwx 1 ownerName groupName 5 Mar 24 10:16 def -> old/
```

To read the contents of a link, use the ***readlink()*** function. (*man* ***readlink*** for details).
The contents of the symbolic link described in *man* ***readlink*** means the **filename** the link points to.
To make a symbolic link use *ln -s.* To create the symbolic link described above (the link is named def and it points to a directory named old), use *ln -s old def* . Be sure to make symbolic links to both directories and regular files for your testing.

When in doubt about what your program should do simply do what ***ls*** does since it is the reference implementation.