CS 502/CS 3013 Operating Systems — WPI, Summer 2024
Craig E. Wills — Project 4 (25 pts)
Assigned: Tuesday, June 25, 2024 — Due: Tuesday, July 16, 2024 (11:59pm)

Introduction

Hierarchical file systems maintain a number of attributes about a file. Typically file systems support attributes such as access permissions, size and last modification time. On a Unix/Linux system, information about the standard set of attributes for a file can be obtained using the *stat()* system call. In this project you will write a program to visualize attributes of files.

Basic File and Directory Visualization

Your program, which should compile into an executable called *proj4*, will take a list of files on the command line. For each file on the command line your program will print the name of the file as well as additional information about the file as described in this project description. If no files are specified then it should display files in the current directory. When your program encounters a file that is a directory then it should open the contents of the directory and recursively display its contents. Your program should use the Unix library calls *opendir()*, *readdir()* and *closedir()* (see man pages) to read the contents of a directory. Your program should ignore the entries "." and ".." within a directory. Files within a directory should be ordered as returned by *readdir()*.

In addition to printing each file name, your program should append symbols to each name as appropriate as done with the "-F" option of the *ls* command. Specifically append '/' if the file is a directory, '*' if the file is executable and '@' if the file is a symbolic link (you will need to use *lstat()* to get additional information). Using these visualization directives, a sample invocation of the program is as follows.

```
% ./proj4 cs502 README
cs502/
    copy.cpp@
    mytestdir/
        moredir/
        testout
    yourtestdir/
        foo
    proj4*
    proj4.cpp
    proj4.o
README
```

This example illustrates additional aspects of your project. Each directory encountered should cause names of files within the directory to be indented when printed. The number of spaces used

for indentation is controlled by the "-i" flag with a valid range of 1–8 spaces and a default of 4 spaces. Similarly, your program should work with a maximum depth of recursion. The recursion depth is controlled by the "-d" flag with a valid range of 0–8 and a default of 2. All options must begin with a '-', be specified on the command line before any file names, and include an '=' sign if a value is specified. For example, the following shows the use of these two options with the above example.

```
% ./proj4 -d=1 -i=2 cs502 README
cs502/
  copy.cpp@
  mytestdir/
  yourtestdir/
  proj4*
  proj4.cpp
  proj4.o
README
```

Visualization of File Size and Age

The next feature of your program is to add visualization of each file's size. You will use the symbol '#' to represent file size using a logarithmic approach. If the file is less than 100 bytes then the size is represented as a single '#'. If the file size is at least 100 bytes, but less than 1,000 bytes then it is shown as "##". Similarly if the file size is at least 1,000 bytes, but less than 10,000 bytes then it is shown as "###". You should continue in this manner with a maximum of 7 symbols for files of size of at least 10,000,000 bytes.

You should use a similar "logarithmic" approach to encode the age (based on file modification time) using up to 7 instances of the symbol '.'. Use a single '.' to indicate if the file is less than one minute old. Use ".." for files that are less than one hour old. Use "..." for files less than one day old, "...." for files less than one week old, "....." for files les than 30 days old, "....." for files less than one year old and "......." for files at least one year old.

Size and age visualization should be printed after the file name on each output line. You should also provide an option "-a" to determine the age based on the last access time rather the last modified time. Adding these visualizations to the preceding example we have:

```
% ./proj4 -d=1 -i=2 cs502 README
cs502/ ## ......
  copy.cpp@ # .
  mytestdir/ ## ...
  yourtestdir/ ## ..
  proj4* ### ..
  proj4.cpp ## ..
  proj4.o ## ..
README # .
```

HTML Visualization

Completion of basic file and directory, as well as file size and age, visualization is worth 19 of the 25 points of the project. For 2 additional points of the project, you need to add a "-h" option, which generates the output in HTML format. Using this format you will *not* display any '#' or '.' symbols after a file name. Rather you will encode the file size using font size (larger files in bigger font) and file age using black and white shading (newer files in darker shades). Using this option you should display the command line as the title for your generated page. The output should be redirected to a file for display via a browser, such as:

```
% ./proj4 -h -d=1 -i=2 cs502 README > output.html
```

where the title of the generated page is "`proj4 -h -d=1 -i=2 cs502 README`".

A simple means to control both font size and shading is to use the HMTL `<font>` tag, which is no longer available in HTML5. More information is available at `http://www.w3schools.com/tags/tag_font.asp`, which includes information and pointers on setting size and shading with CSS.

Additional Work

For the remaining 4 points of the projects, you need to add two additional options. The first option is "-s" and indicates that the files within each directory should be sorted in alphabetical order rather than in the default order provided by *readdir()*. The second option is "-t" and indicates that more information about the file type should be shown for files that are not directories, executables or symbolic links. This information should be obtained using the *file* command, where a sample command-line invocation is:

```
% file proj4.cpp
proj4.cpp: ASCII C++ program text
```

You can use the library call *popen()* to invoke a user-level command and have the output of the command piped to your process for reading. In the default text mode, you can print type information as text after the size and age. With the HTML option, you need to map different types of files to different colors. You must include a key for which file types are mapped to which color. You should still use shades of color to represent age of the file.

Summary of Command Syntax

Just to summarize, if you implement all options in the project then the command syntax for your program is as follows. Note that options are indeed optional and they may be specified in any order, but must come before any specified files. If no file list is specified then files in the current directory should be displayed with no initial indentation.

```
proj4 [-i=m] [-d=n] [-a] [-h] [-s] [-t] [file(s)]
```

where the options are

-a use last access time to determine file age rather than last modification time.

-d=n use a maximum recursion depth of *n* when showing sub-directory contents. Value range is 0–8. Default is 2.

-h generate the output in HTML using font size and shading to represent file size and age.

-i=m indent *m* spaces for each subdirectory. Value range is 1–8. Default is 4.

-s sort directory entries alphabetically rather than in the order they are read.

-t show the file type based on the output of *file* command. Should be done with colors (including a key) when using HTML.

Additional Information

The visualization of information in general and hierarchies in particular is historically an interesting research problem. One relevant approach for the visualization of file hierarchies is the use of Treemaps. More information about this approach as well as a pointer to a free tool is available at `http://www.cs.umd.edu/hcil/treemap-history/`.

Submission of Assignment

Submit your project via InstructAssist with the name "proj4". At a minimum you should submit your program code and a file showing execution of your program on test cases for your program. The *script* command is useful for doing so.