CS 385 –Operating Systems – Fall 2013
Homework Assignment 4
Exploring the FileSystem Structure with FindIt

Due: **Friday December 6. Electronic copy due at 2:30, optional paper copy may be delivered to the TA.**

Overall Assignment

For this assignment you are to write a program named "findit" that operates very similarly to the UNIX system command "find", with some slightly different functionality.

Background: The find command

The find command is one of the most powerful in all of UNIX, because it can search the entire system for files meeting certain criteria, and once it finds them, it can take a variety of different actions, including printing out the file or executing any other UNIX command with the found file(s) as input arguments.

The basic syntax of find is:

find [path...] [expression]

- The first argument that begins with a dash, e.g. –print, marks the beginning of the list of expressions. All arguments before that argument are paths.
- Paths, if present, indicate the starting location(s) where find should start its search. (Generally one or more directories, but other files can also make sense if wild cards are used.) If no paths are given, then find starts its search from the current working directory.
- Find examines all of the accessible files in the subdirectory tree(s) starting from the initial location(s), and for each such file it evaluates the string of given expressions, each of which will evaluate to either true or false, following the rules of "short-circuiting" when evaluating complex Boolean expressions in C or C++. I.e. as soon as the overall truthhood of the complex expression can be determined it stops evaluating the string of expressions. The expressions are considered to be ANDed together, unless the OR operation is expressly specified. Find also supports parentheses, though we will probably not for this assignment.
- The find documentation categorizes expressions into one of four categories:
    1. Options – Control how find works, such as the "–depth" option which specifies that find perform a depth-first search instead of a breadth-first search. Always true.
    2. Tests – Determine whether any given file meets certain criteria, such as the "-uid n" option that evaluates to true if the user ID of the owner of the file is equal to n.
    3. Actions – Always evaluate to true, and also carry out some action, such as –print that prints out the full pathname of the file (relative to the search starting location.)
    4. Operators – Modify the expressions in some way, such as the "–not" to invert the truthhood of an expressions result or the "-or" to combine two expressions in an OR sense instead of an AND sense.

Findit Expressions

Your findit program should implement the following expressions that are not implemented by find:

- -summarize

  Prints out a table listing the different file types (ordinary files, directories, character device files, etc., and for each file type reports how many of those type of files were found, and the total disk space consumed by all of the files found of that type. (If appropriate.) See lstat doc for file types.
- -access type

  Evaluates to true if the findIt program has access of the type given, where the type is one of (read, write, execute, any). Note that this requires identifying the **effective** user id and group id of the user running the findit program, so the appropriate set of access control bits can be examined for each file found. (The -not expression can be used to find files for which findit does ***not*** have access.)
- -largest type
- -smallest type

  Finds the largest or smallest item of the given type, where type is one of (file, dir, tree). For the dir type, findit considers only the sum of the sizes of the files directly contained within a given directory (and the directory itself), and for the tree type it considers all of the files and directories in the subtrees below each directory found. Obviously the largest tree will be a direct subdirectory of the initial search location(s), and the smallest will be a directory that has no subdirectories.
- -treedir

  Prints out a directory tree listing all of the directories, subdirectories, sub-subdirectories, etc starting at the initial search location, and indicating how many directories and files are contained in each one, and their total sizes (including the sizes of subdirectories, but not the files contained therein.) For example:

```
usr( 2 dirs, 5 files, 23456 bytes )
+- john( 3 dirs, 20 files, 23423423 bytes )
|  +- courses( 3 dirs, 0 files, 0 bytes )
|  |  +- cs 107( 0 dirs, 17 files, 45645634 bytes )
|  |  +- cs 385( 0 dirs, 27 files, 288774234 bytes )
|  |  +- cs 440( 0 dirs, 23 files, 15464234 bytes )
|  +- papers( 0 dirs, 10 files, 56456 bytes );
|  +- UASC( 0 dirs, 14 files, 9038745 bytes );
+- mary( 0 dirs, 15 files, 98347593 bytes )
```

- -sparse

  Evaluates to true if the number of blocks allocated is less than that indicated by the file size.

Find Expressions

Your program should also implement the following find expressions. See the find documentation for details:

- -help
- -depth
- -maxdepth
- -amin, -atime
- -cmin, -ctime
- -mmin, -mtime
- -links
- -uid, -gid
- -size
- -type
- -print
- -fprint
- -exec
- -ok

System Commands

The following system commands will likely be useful for this assignment:

- **find** [path...] [expression] // For reference only. Execing find is ***not*** acceptable!
- **int stat(const char ****file_name***, struct stat ****buf***);**
- **int lstat(const char ****file_name***, struct stat ****buf***);**
- **`DIR *opendir(const char *`***`name`***`);`**
- **`struct dirent *readdir(DIR *`***`dir`***`); // From manual section 3, system calls`**
- **`int closedir(DIR *`***`dir`***`);`**
- **`uid_t geteuid(void);`**
- **`gid_t getegid(void);`**

Required Output

- All programs should print your name and ACCC account ID as a minimum when they first start.
- Beyond that see above.

Other Details:

- The TA must be able to compile your program (on bert and ernie) by typing "make orderSearcher". Provide a makefile if necessary. As always, you are free to develop wherever you wish, but the TA will be grading the programs based on their performance on the CS department machines.

What to Hand In:

1. Your code, **including a makefile if needed, and a readme file,** should be handed in electronically using Blackboard.
2. The purpose of the readme file is to make it as easy as possible for the grader to understand your program. If the readme file is too terse, then (s)he can't understand your code; If it is overly verbose, then it is extra work to read the readme file. It is up to you to provide the most effective level of documentation.
3. The readme file must specifically provide direction on how to run the program, i.e. what command line arguments are required and whether or not input needs to be redirected. It must also specifically document any optional information or command line arguments your program takes, as well as any differences between the printed and electronic version of your program.
4. If there are problems that you know your program cannot handle, it is best to document them in the readme file, rather than have the TA wonder what is wrong with your program.
5. **For this assignment you must also document the results of your experimentation, as well as reporting the best results found by your program.**
6. A printed copy of your program, along with any supporting documents you wish to provide, (such as hand-drawn sketches or diagrams) may be delivered to the TA, and must match exactly the electronic submissions.
7. Make sure that your **name and your ACCC account name** appear at the beginning of each of your files. Your program should also print this information when it runs.

Optional Enhancements:

- Implement other expressions, either supported by find or those that you envision yourself.