Programming Project: pseudofind

Introduction

For this assignment you will write a program that works a little bit like the Unix **find** command. In doing so, you will have a chance to work with the Unix directory structure, recursion, and, optionally, the stat() system call.

Explanation of find

Imagine you have set up a complex, multi-level directory structure to organize your files. Directories contain subdirectories which in turn contain subdirectories of their own. Everything is in perfect order, but suddenly you realize that you mislaid a file called wtmplib.c .

With the find command, you could type:

```
% cd ; find . -name wtmplib.c -print
```

The Unix **find** command searches directories and all their subdirectories for files that fit certain specifications. In the example just mentioned, find is used to search by name. Find also allows you to search for files by modification time, by size, by owner, by number of links, *etc*, as well as combinations of attributes. Read the manual page on find for more info.

Explanation of pfind

The actual find command is not too tricky to write once you get the general structure in place. Rather than focus on the details of the different sorts of search criteria, you will implement a version that searches by name only. The syntax of your version of find, (pfind for pseudo-find), is:

```
% pfind starting_dir filename
```

for example, the command

```
% pfind /home/s/smith core
```

would search all the subdirectories of /home/s/smith and print the full pathnames of all files called core. pfind, like the real find, does not stop until it has searched all subdirectories.

Getting Started

1. Read the manual page on the find command. Try it out in your own directory or in the course directory. For example, you might try:

```
% find ~COURSE -name Makefile -print
```

The real version of find has a more complicated syntax. Your version is much simpler: it takes two arguments only.

2. cd to ~COURSE/hw/pfind and explore the directory trees under pft.d and pft2.d . Search for things named *cookie* under pft.d, and search for things called *d8* under pft2.d . Use the find command to perform these two searches. Your program will need to perform the same search and produce the same results.

3. Once you see how real find works and have worked through the procedure manually, think about what your find has to do. It has to look at each entry in the starting directory. If the entry matches the name you

are looking for, print the directory name with the file name appended. If the entry is a directory your program will have to do the same operations on that subdirectory. Therefore the search and print function will be recursive.

4. Decide how you will determine if a name refers to a directory or a file. Read the text, online documentation, or header files. This step is an essential part of your program.

5. Write a function called searchdir(char *dirname, char *findme). This function will open the directory, read it entry by entry, print out any names of files that have name 'findme'. In addition, if any entry in the directory is a directory, then searchdir() will have to create a new string to hold the new directory name and pass that string to itself. Use malloc() to create the string; a fixed size buffer is liable to run out when you least expect it.

Building Your Program

Write your own file or files of code. Be sure to write a Makefile and an explanatory document.

Testing Your Program

You can make up your own test system data, but you must test it sometime with ~COURSE/hw/pfind/test.pfind . This shell script will exercise your program on a preset directory tree. To run it, type:

```
% ~COURSE/hw/pfind/test.pfind
```

The Open File Limit

There is no limit on the depth of a directory tree; any directory can contain subdirectories. The shell script loop in the text shows how to create extremely deep structures.

Each time you open a directory in the recursive calls, your progam has another open file. Unix usually sets a limit on the number of open files a program can have. Once that limit was 20, then 60, now it could be much larger. Nonetheless, in a very deep directory tree, you might hit that limit. In that case, the call to `opendir()`,

Describe a solution to this limit. Devise a method that will allow your program to process directory trees that are deeper than the maximum number of open files.

What to Turn In

Hand in (1) your source code, (2) a typescript of its output from the test.pfind test script and any other samples you wish to include, (3) a Makefile, (4), a short answer to the *Open File Limit* question, (5) a design outline.