CSC 209 Assignment 3, Summer 2022: Directories and processes

Due by the end of Monday August 1, 2022; no late assignments without written explanation.

Part 1 (smaller): scandir()

A slightly higher-level function built upon opendir()/readdir()/closedir() is *scandir*(), which reads through an entire directory and returns an array of all of the "struct dirent"s. Scandir() needs to malloc space for each of the copies of the struct dirents, as well as space for the array of pointers-to-struct dirent. Its return value indicates the numbers of entries in the directory, or −1 for error.

The real scandir() also takes function-pointer arguments to allow selecting a subset of the directory list, and to allow sorting the directory list in arbitrary ways. You won't emulate these aspects of scandir(); your "myscandir()" will take just two arguments. The first argument is a string indicating the directory to scan; this is simply passed to opendir().

The second argument is a pointer to a variable which will be set to the array information. The array will be an array of pointers-to-struct-dirent (since you will call malloc() for each struct dirent). Therefore myscandir() needs to give the caller a pointer to the zeroth item of this list; such a pointer would be a pointer-to-pointer-to-struct-dirent. Thus the second argument to myscandir() is actually a pointer-to-pointer-to-pointer-to-struct-dirent, which myscandir() dereferences to assign the pointer-to-pointer-to-struct-dirent value to the appropriate variable as chosen by the caller.

myscandir(), like scandir(), returns type int. It returns the number of items in the resulting array, or −1 for error.

The resulting data structure needs a small loop to free() properly. You will also write "myfreescandir()", which takes a pointer-to-pointer-to-struct-dirent argument and an integer argument (the latter being the original return value from myscandir()). (It is not valid to call myfreescandir() if myscandir() returned −1.)

There is a myscandir.h in /u/csc209h/summer/pub/a3/myscandir.h which you must copy without editing, and #include in your myscandir.c. It simply declares myscandir() and myfreescandir(). Also in that directory is an example test driver with a main() which you may want to use in testing. (You submit only myscandir.c, without a main().)

Part 2 (bigger): a binary search tree without malloc(), via fork()

In this part of the assignment you will write a binary search tree, except that each connection between nodes will be represented by pipes, and each node is a separate process. The processes are arranged in a binary search tree, connected with pipes. Each connection needs bidirectional communication, in which a search request can be sent from parent to child and the result can be sent from child to parent; thus each link will correspond to *two* pipes (since a pipe is unidirectional).

The search keys are integers, and the values are arbitrary character strings of size up to 9 bytes (i.e. stored in an array of size 10). The initial process is the root node. The key and value are specified on the

command-line (the key is argv[1] and the value is argv[2]).

The initial process reads lines from stdin in a loop. The line can be simply an integer; this means to search for that key. Otherwise, the line can be an integer, a space, and the value; this means to set that key to that value. In either case the (possibly new) key and value are output.

Each process loops until EOF, getting a key or a key+value either from stdin (in the case of the root node process) or from the pipe from its parent (in the case of all other processes). Then, if the key matches its own key, it can return the key+value information up the pipe to its parent (or, if it's the root node, output it to stdout). Otherwise, it passes the request down to either its left child or right child as appropriate. When a process is first created, it doesn't have a left child or right child; when it goes to pass a request down to a child, it calls fork() if necessary (setting up the new child, which then does its own loop).

Do not attempt to height-balance your binary search tree.

Upon EOF from the pipe from the parent, close the pipes to any child node processes, wait for all child node processes, then exit. Your child nodes, if any, will thus get EOF from their pipes to their parents (you), and do the same; and so on all the way down the tree. Similarly, when the root node process gets EOF from stdin, it will also do this process (close pipes to children, wait, exit). This will cause an orderly shutdown of the entire tree when you signal EOF on stdin (e.g. by pressing control-D). (Note that this scheme means that the root node will not exit until all other nodes have exited, which is good because the shell you ran the program from is waiting for the initial process, which is the root node.)

There is "starter code" in /u/csc209h/summer/pub/a3/tree.c.starter which contains a suitable main() function dealing with argv, plus a function to parse the key+value data (from stdin in the main process) into a struct containing the integer key and the string value. You can then simply read() and write() this struct between parents and children.

You do not necessarily need to modify the main() in tree.c.starter, but you can do so if you wish; similarly for the utility functions in tree.c.starter.

Other notes

Your programs must be in standard C. They must compile on the teach.cs computers with "gcc –Wall" with no errors or warning messages, and may not use linux-specific or GNU-specific features.

You should call your assignment files myscandir.c and tree.c. Auxiliary files are not permitted. Submit with commands such as

```
submit -c csc209h -a a3 myscandir.c tree.c
```

and the other 'submit' commands are as before.

Please see the assignment Q&A web page at https://www.teach.cs.toronto.edu/~ajr/209/a3/qna.html for other reminders, and answers to common questions.

Remember:

This assignment is due at the end of Monday, August 1, by midnight. Late assignments are not ordinarily accepted, and *always* require a written explanation. If you are not finished your assignment by the submission deadline, you should just submit what you have, for partial marks.

Despite the above, I'd like to be clear that if there *is* a legitimate reason for lateness, please do submit your assignment late and send me that written explanation.

And above all: Do not commit an academic offence. Your work must be your own. Do not look at other students' assignments, and do not show your assignment (complete or partial) to other students. Collaborate with other students only on material which is not being submitted for course credit.