CSCI 3323 (Principles of Operating Systems), Fall 2021

Homework 1b

Credit: 20 points.

1 Reading

Be sure you have read, or at least skimmed, Chapter 5 of the textbook.

2 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) one of two ways:

- using my `mail-files` script, linked from the course Web site under "Links".
- by putting them in your course "TurnIn" folder on Google Drive. (Note that I want plain-text files, ideally with an extension appropriate for the language — e.g., `.c` for C — but if Google Drive balks at that, rename to have an extension of `.txt`. I want something I can compile as is, except for possibly a change of filename. So no screenshots!)

You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (20 points) For this problem, your mission is to produce a simple command shell for a Linux system, using the approach described in Chapter 5 of the textbook, i.e., by executing each command as a separate process. Your starting point is incomplete code that takes care of the parts I think are tedious to write in C; it repeatedly prompts the user for a command and command-line arguments and parses the result, breaking it apart at white space. (Yes, this is a much-simplified version of what real shells do — there's no wildcard expansion, recognition of environment variables, etc., etc.). What you need to do is add code that for each command entered:

    - Executes the command in a separate process, or prints an error message if it can't. You can and should assume that the command name is a pathname (this is tedious for the end user, but hardcoding a search path would be limiting in that only commands in the search path could be executed).
    - Waits for the command to complete.

    The program runs until it hits end of file on standard input, so if you're running it interactively you can stop it by pressing control-D.

    Here is a sample execution:

    ```
    $ ./simple-shell
    next command?
    /bin/ls
    ```

```
Makefile     simple-shell.c  test-input.txt
simple-shell  simple-shell.c     typescript

next command?
/bin/echo ab cd ef gh
ab cd ef gh

next command?
junk
cannot execute command: No such file or directory

next command?
/bin/ls junk
/bin/ls: cannot access junk: No such file or directory

next command?
```

Starter code: `simple-shell.c`.

Comments with the word `FIXME` indicate where you should make changes.

If you compile with the default version of `gcc`, you may need the `-std=c99` flag.

Note that you should not need to write a lot of code; the textbook chapter has several examples of calling `fork()` and `execvp()` to do similar things, and the starter code builds an array of C-style strings containing the command name and parameters.

For extra credit (up to 5 points), you can add more functionality (searching a path for the command, doing more sophisticated parsing of inputs, exiting when the user types "exit", etc.). If you do, *add something to the comments in the code describing your added functionality*. (If you just put it in your code, I could easily miss it.) Whatever changes you make, however, be sure your program will still work with input that is valid for the starter code. (E.g., you could implement some sort of search path, but if you do, be sure the program still accepts a full path for the command.)

C tip: Get in the habit of compiling with the `-Wall` flag and paying attention to warning messages. Sometimes warning messages really are just warnings you can ignore, but often they are signs of problems you should fix. Code that produces warnings with compiled with `-std=c99 -Wall -pedantic -O` will lose points. Recall (or note) that `man` pages for functions tell you what if any `#include` directives you need to include in your code.

3 Essay and pledge

Include with your assignment the following information.

For programming assignments, please put it a separate file. (I strongly prefer plain text, but if you insist you can put it in a PDF — just no word-processor documents or Google Drive links please.) For written assignments, please put it in your main document.

3.1 Pledge

This should include the Honor Code pledge, or just the word "pledged", *plus* at least one of the following about collaboration and help (as many as apply). Text *in italics* is explanatory or something for you to fill in; you don't need to repeat it!

- I did not get outside help *aside from course materials, including starter code, readings, sample programs, the instructor.*
- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, etc. (Here, "help" means significant help, beyond a little assistance with tools or compiler errors.)*
- I got help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. (Here too, you only need to mention significant help — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided help to *names of students* on this assignment. *(And here too, you only need to tell me about significant help.)*

3.2 Essay

This should be a brief essay (a sentence or two is fine, though you can write as much as you like) telling me what if anything you think you learned from the assignment, and what if anything you found interesting, difficult, or otherwise noteworthy.