COP 4600 - Homework 3

Due Sept 21, 2020

## Total points: 6 + 1 bonus points

# Create a shell program (6 pt)

Use the Unix environment you installed in the previous homework to write a C or C++ program called mysh replaces the command shell in Unix. After started, it prints a prompt “#” and reads a command line terminated by newline. This line should be parsed out into a command and all its arguments. In other words, tokenize it.

* You may assume that the only supported delimiter is the whitespace character (ASCII character number 32).
* You do not need to handle "special" characters. Do not worry about handling quotation marks, backslashes, and tab characters. This means your shell will be unable support arguments with spaces in them. For example, your shell will not support file paths with spaces in them.
* You may set a reasonable maximum on the number of command line arguments, but your shell should handle input lines of any length.

# Implement built-in commands

Your shell should be able to interpret the following commands

# movetodir directory

It is used to change the current directory (which must be an internal variable) to the specified one. If the specified directory does not exist, it should keep the old directory and write an error message.

# whereami

Prints the current directory.

# history [-c]

Without the parameter, prints out the recently typed commands (with their arguments). If “-c” is passed, it clears the list of recently typed commands

# byebye

terminates the mysh shell

# start program [parameters]

The argument “program” is the program to execute. If the argument starts with a “/” (such as /usr/bin/xterm, the shell should interpret it as a full path. Otherwise, the program will be interpreted as a relative path starting from the current directory.

The program will be executed with the optional “parameters”. It uses fork() + exec() to start the program with the corresponding parameters, and waits until the program terminates (use the waitpid() call).

For instance

 run /usr/bin/xterm –bg green

would bring up a terminal with a green background. The prompt would not return until the terminal is closed.

Display an error message if the specified program cannot be found or cannot be executed.

# background program [parameters]

It is similar to the run command, but it immediately prints the PID of the program it started, and returns the prompt.

# exterminate PID

Immediately terminate the program with the specific PID (presumably started from this command line interpreter). Use the kill() function call to send a SIGKILL signal to the program. Display success or failure.

To help you, you might want to read some of the readings associated with this homework at the class webpage. I have also linked from the webpage a code segment which might help you in reading the commands.

## Extra credit (1 point)

Implement a repeat command as follows:

# repeat n command …

Interprets n as the number of times the command must be run, command as the full path to the program to execute, and the others as parameters. The command should start the specified number of program instances, print the PIDs of the created processes and then return to the prompt, without waiting for the processes to terminate. For instance:

 repeat 4 /usr/bin/xterm

would bring up 4 terminals and print out something like:

PIDs: 31012, 31013, 31014, 31015.

## More extra credit (1 point)

Implement the following command:

# exterminateall

Which immediately terminates all the programs previously started by the mysh shell which had not been previously terminated by it, or by exterminate. It should output something like this:

Murdering 3 processes: 16000 31012 31013

## What to submit:

* The code as a single .c or .cpp file.
* If you implemented the extra credit part: a text file describing the syntax of the implementation, and example of use.

# Notes:

Note 1: If you had not used a command line shell before, I would recommend that before you start the homework familiarize yourself with some shell(s). Bash – the default shell in most Linux installations and recent MacOS is a good choice. It is also available in Windows.

Note 2: Many of the individual components of this project are standard code snippets. Don’t be afraid to google for samples. For instance you might want to Google these and similar:

* + Tokenize string c++
	+ Fork process c++

Note 3: This is a relatively long, but easily modularizable program. This is an excellent opportunity to improve and show off your programming skills by organizing the program in a clever way. For instance, clearly, the individual commands should be implemented in their own function. Parameters should be passed to them. They should return an error / success code. The parsing should be done in one location, and the command functions called from there.

You should be able to debug these functions individually.

Note 4:

/usr/bin/xterm will work on Linux, or on Windows with VcXsrv or other X environment enabled. If you are on MacOS you will need to find a command line to bring up the Mac terminal.