COP 4600 - Homework 3

Due Feb 28, 2022

Total points: 5 + 2 bonus points

Create a shell program (5 pt)

Use the Unix environment you installed in the previous homework to write a C or C++ program called mysh that 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, you need to 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.

Built-in commands

Your shell should implement the following commands.

```
# history [-c]
```

Without the argument, it prints out the recently typed commands (with their arguments), in reverse order, with numbers. For instance, it will print:

```
0: history
1: start chromium-browser
2: start /usr/bin/xterm –bg green
```

If the argument "-c" is passed, it clears the list of recently typed commands. You will need to implement an internal history data structure to keep the list of commands. The data structure should allow you to grow the list to an arbitrary length.

You will need to save the history data structure to an external file called mysh.history in the current directory (when exiting), and load it (when starting the shell). If the file does not exist, the program will need to start with an empty shell.

```
# byebye
```

Terminates the mysh shell (and it should save the history file).

```
# replay number
```

Re-executes the command labeled with the given number in the history

```
# 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". mysh should use fork() + exec() to start the program with the corresponding parameters, and wait until the program terminates (use the waitpid() call).
For instance

```
start /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.

```
# terminate PID
```

Immediately terminate the program with the specific PID (presumably started using background). Use the kill() function call to send a SIGKILL signal to the program. Display success or failure.

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:

```
# terminateall
```

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

```
Terminating 3 processes: 16000 31012 31013
```

Or

```
No processes to terminate.
```

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, before you start the homework familiarize yourself with the shell used by the Unix environment you installed in Homework 2 - very likely bash in Linux and zsh on Mac.

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.

Note 5:
The program should not use any external library, except the ones that are included by default by the compiler. You program should be compilable using

```
gcc mysh.c
```

or

```
g++ mysh.cpp
```

Note 6:
The program should not rely on the existence of any file on the file system – such as assuming that a directory, or a file already exists. For instance, the program should create its own history file, if it is not existing.
You might want to test your program by moving the executable into an empty directory.