CSE-381: Systems 2

Homework #5

Due: Thu Sept 2303 2021 before 11:59 PM (Midnight)

Late-submission (80% points): Up to Fri Oct 1 before 11:59 PM

Email-based help Cutoff: 5:00 PM on Sun, Feb 25 2018

Maximum Points for This Part: 27

Objective

The objective of this part of the homework is to develop 1 C++ program to:

- Appreciate how the `bash` shell runs commands
- Gain familiarity with fork and exec system calls
- Continue to build strength with I/O streams & string processing
- Review problem solving strategies.

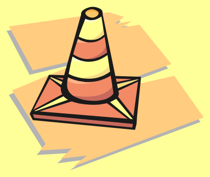
Submission Instructions

This homework assignment must be turned-in electronically via the CODE Canvas plug-in. Ensure your program compiles without any warnings or style violations. Ensure you have tested operations of your program as indicated. Once you have tested your implementation, upload the following onto Canvas:

- Just the one C++ source file, named with the convention `MUID_hw5.cpp`, where `MUID` is your Miami University unique ID.

General Note: Upload each file associated with homework (or lab exercises) individually to Canvas. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

Grading Rubric:



The programs submitted for this homework **must pass necessary base case test(s) in order to qualify for earning any score at all**. Programs that do not meet base case requirements will be assigned zero score!

Program that do not compile, have a method longer than 25 lines or badly formatted code as in } } }, etc., or just some skeleton code will be assigned zero score.

- Point distribution for various commands:
  - Base case: Ignore comments & execute specified command **8 points**.
  - Additional features: `SERIAL` : **7 points**, `PARALLEL` : **8 points**
  - Formatting, Good organization, code reuse, documentation, etc. – **earned fully only if `SERIAL` & `PARALLEL` functionality operate correctly**: **4 points**.
- **-1 Points**: for each warning generated by the compiler when using Miami University C++ project (warnings are most likely sources of errors in C++ programs)

- **-1 Points**: for each style violation in the programs reported by CSE department's C++ style checker. Ensure you use correct `Miami University C++ Project` setting in `NetBeans`.

Develop a custom Linux shell

Background

Operating Systems essentially provides `fork` and `exec` system calls to run programs, but an OS does not start running programs by itself. This task is delegated to another program, called a "shell". The shell (in our case we are using `bash` shell) accepts inputs from the user and based on the user-input executes different commands. Note that shells can be graphical and permit users to double-click on an icon to indicate the program to run.

Homework requirements

This homework requires developing a textual shell program analogous to `bash` used in Linux. Note that the shell you will be developing will be rather simple when compared to `bash`, but will help you obtain a good understanding of how a shell works. In addition, this is a great project to showcase your skills for interviews. Your shell must operate in the following manner:

Repeatedly prompt (via a simple `"> "`) and obtain a line of input (via `std::getline`) from the user and process each line of input in the following order:

- If the line is empty or the line begins with a pound (#) sign, ignore those lines. They serve as comments similar to how `bash` shell works.
- The 1st word in the line is assumed to be a command (case sensitive) and must be processed as:
    1. If the command is `exit` or input has reached end-of-file your shell must terminate
    2. If the command is `SERIAL` then the 2nd word is name of a file that contains the actual commands to be executed one at a time. The shell waits for each command to finish.
    3. If the command is `PARALLEL` then the 2nd word is name of a file that contains the actual commands to be executed. In this case, all the commands are first `exec`'d. Then wait for all the processes to finish **in the same order they were listed**.
    4. If the 1st word is not one of the above 3, then it is assumed to be the name of the program to be executed and rest of the words on the line are interpreted as a command-line arguments for the program.

Note: For every command run (including `SERIAL` and `PARALLEL` runs), your shell must print the command and command line arguments (after they have been suitably split up using `std::quoted` as discussed further below). Once the command has finished, your shell must print the exit code. See sample outputs for wording and spacing.

Tips & suggestions

- This homework uses concepts from lab exercise on working with fork and exec. Ensure you review the lab exercise on how to run programs.
- A lot of the code you need is already in lecture slides. So, ensure you review the material on string processing, `vector`, `fork`, and `exec` before starting on the program.

- Prefer to use the `myExec` program from lecture slides to minimize sources of errors, such as forgetting to add the `nullptr`.
- Get base case working first. The string processing is trivial (almost all the code in slides) with `std::quoted`. Do not overcomplicate the string processing part. For reading command from the user, prefer the following style of coding –

```
// Adapt the following loop as you see fit
std::string line;
while (std::cout << "> ", std::getline(std::cin, line)) {
   // Process the input line here.
}
```

- Processing commands from files becomes straightforward if you code the base case method to use generic I/O streams as in -- `process(std::istream& is = std::cin, std::ostream& os = std::cout)`.
- When printing messages from your program, adopt the following two important tips to avoid garbled or seemingly inconsistent outputs (where output from parent and child process get mixed together):
  - Prefer to print from parent process, before the fork system call
  - Use `std::endl` (rather than "`\n`") to flush the output to ensure it is displayed
- Use a `std::vector` to hold PIDs of child processes so that you can call `waitpid` on each one when running them in parallel. Exit codes are obtained from `waitpid`. **Simply printing zero for exit code is totally wrong and you will lose points**.
- Do not use `exit(0)` to terminate your program. That is not the correct approach. Instead appropriately use the `break` statement.

Sample input and outputs

In the sample outputs below, the following convention is used:

- Text in **bold** are inputs (logically) typed-in by the user (↵ is for pressing ENTER key)
- Text in blue are additional outputs printed by your program
- Other text are outputs from various commands `exec`'d by your program

Base case (8 points)

```
> # Lines starting with pound signs are to be ignored.↵
> echo "hello, world!" ↵
Running: echo hello, world!
hello, world!
Exit code: 0
> ↵
> head -2 /proc/cpuinfo↵
Running: head -2 /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
Exit code: 0
> ↵
> # A regular sleep test. ↵
> sleep 1↵
Running: sleep 1
Exit code: 0
> ↵
> # Finally exit out↵
> exit↵
```

SERIAL test (7 points)

```
> echo "serial test take about 5 seconds"↵
Running: echo serial test take about 5 seconds
serial test take about 5 seconds
Exit code: 0
> SERIAL simple.sh↵
Running: sleep 1
Exit code: 0
Running: sleep 1s
Exit code: 0
Running: sleep 1.01
Exit code: 0
Running: sleep 0.99
Exit code: 0
Running: sleep 1s
Exit code: 0
Running: echo -e "done" running\t the simple.sh script\x2e
"done" running       the simple.sh script.
Exit code: 0
> exit↵
```

PARALLEL test (8 points)

```
> # echo "parallel test take about 1 second"↵
> PARALLEL parallel.sh↵
Running: sleep 1
Running: sleep 1.01
Running: sleep 1s
Running: sleep 0.99
Running: sleep 1s
Exit code: 0
Exit code: 0
Exit code: 0
Exit code: 0
Exit code: 0
> exit↵
```

Organization and structure verification

In order to earn the full 6 points in this category, the program should strive to reuse code as much as possible. Specifically, processing lines from files (in `SERIAL` or `PARALLEL` command) should reuse code from the method used for processing individual commands. This kind of code reuse will be an important expectation in your future jobs. **Note that in order to get all of the points in this category, `SERIAL` & `PARALLEL` functionality must operate correctly**

Submit to Canvas

This homework assignment must be turned-in electronically via CODE Canvas plug-in. Ensure all of your program(s) compile without any warnings or style violations and operate correctly. Once you have tested your implementation, upload the following onto Canvas:

- The 1 C++ source file for this part of the homework, with the naming convention `MUID_hw5.cpp`, where `MUID` is your Miami University unique ID.

Upload each file associated with homework (or lab exercises) individually to Canvas. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.