CSE-381: Systems 2

Homework #4

Due: Wednesday September 16 2020 before 11:59 PM

Late-submission (75% points): Up to Thu Sept 17 before 11:59 PM

Email-based help Cutoff: 5:00 PM on Tue, Sept 15 2020

Maximum Points: 25 points

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 working with classes & problem-solving strategies.
- **Do not use global variables**

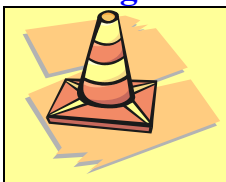
Submission Instructions

This part of the homework assignment must be turned-in electronically via Canvas using the CODE plugin. 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 with the naming convention `MUID_hw4.cpp`, where `MUID` is your Miami 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! **Programs 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 commands [**10 points**]
  - Additional feature: Process from URL (for `SERIAL` & `PARALLEL`) [**10 points**]
  - Formatting, Good organization, code reuse, documentation, etc. – **earned fully only if `SERIAL` & `PARALLEL` functionality operate correctly**: **5 points**.
- **-1 Points**: for each warning generated by the compiler.
- **-25 Points**: If global variables (without namespaces) are used.

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 the `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 jobs, internships, or 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. **Base functionality:** If the command is `exit` your shell must terminate
    2. **Extra functionality**: If the command is `SERIAL` then the 2nd word is URL to a text file that contains the actual commands to be executed one at a time. Your shell must wait for each command to finish.
    3. **Extra functionality**: If the command is `PARALLEL` then the 2nd word is URL to a text 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. **Base functionality**: 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 the 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. Once the command has finished, your shell must print the exit code. URLs will always start with "`http://`". 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 `ChildProcess` class from the lab exercise and use it.
- First write comments on make a plan. Include, code snippets in your comments. Then start implementing. Work one feature at a time and submit it. Refactor your program as needed.
- A lot of the code you need is already in your previous homework & lecture slides. So, ensure you review the material on string processing, `vector`, `fork`, and `exec`.

- Use the `ChildProcess.h` and `ChildProcess.cpp` from lab exercise to ease development. Ensure you use pass-by-reference for all object-parameters.
- Get base case working first. The string processing is trivial (adapt example on Slide 9 of `Part4_Cpp_Containers.pdf`) with `std::istringstream` and `std::quoted`. No, no – `split()` etc. won't work due to quotes processing.
- For reading command from the user, prefer the following style of coding –

```
void process(std::istream& is = std::cin,
             const std::string& prompt = "> ", bool parallel = false) {
    // Adapt the following loop as you see fit
    std::string line;
    while (std::cout << prompt, std::getline(std::cin, line)) {
        // Process the input line here.
    }
}
```

- If you code the base case method to use generic I/O streams (as suggested above) then processing commands from URLs becomes straightforward. Note this is just a suggestion. So, if you choose to use it is up to you to figure out the best way to use the above method.
- **Processing URLs was done in homework #1. Borrow approach and code from there.**
- 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
- 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.

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 to be printed by your program
- Other text are outputs from various commands `exec`'d by your program

Base case (10 points)

Ignore blank and comment lines and run commands typed-in by the user.

```
> # 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↵
```

Additional Feature: Process commands from a URL [7 points]

SERIAL test

```
> 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 http://ceclnx01.cec.miamioh.edu/~raodm/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↵
```

PARALLEL test

```
> echo "parallel test takes only about 1 second"↵
Running: echo parallel test takes only about 1 second
parallel test takes only about 1 second
Exit code: 0
> PARALLEL http://www.users.miamioh.edu/raodm/parallel.sh ↵
Running: sleep 1
Running: sleep 1s
Running: sleep 1.01
Running: sleep 0.99
Running: sleep 1s
Exit code: 0
Exit code: 0
Exit code: 0
Exit code: 0
Exit code: 0
> exit↵
```

Good documentation, Organization and structure [5 points]

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 URL (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. **Ensure you have Javadoc/doxygen style comments for your methods (see ChildProcess.h for example)**.

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 your program compiles without any warnings or style violations and operate correctly. Once you have tested your implementation, upload:

- Just the one C++ source file with the naming convention `MUID_hw4.cpp`, where `MUID` is your Miami 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.