CSC 360: Operating Systems
Spring 2021

Assignment #1
Due: Friday, February 12 at 11:55 pm via push to your Gitlab.csc repository

Programming Platform

For this assignment **your code must work on the Virtualbox/Vagrant configuration you provisioned for yourself in assignment #0.** You may already have access to your own Unix system (e.g., Ubuntu, Debian, macOS with MacPorts, etc.) yet I recommend you work as much as possible while with your CSC360 virtual machine. Bugs in systems programming tend to be platform-specific and something that works perfectly at home may end up crashing on a different computer-language library configuration. (We cannot give marks for submissions of which it is said “It worked on Visual Studio!”)

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussions are encouraged. However, sharing of code is strictly forbidden. If you are still unsure about what is permitted or have other questions regarding academic integrity, please direct them as soon as possible to the instructor. (Code-similarity tools will be run on submitted programs.) Any fragments of code found on the web and used in your solution **must be properly cited** where it is used (i.e., citation in the form of a comment given source of code).

Use of gitlab.csc.uvic.ca

Each student enrolled in the course has been assigned a Git repository at gitlab.csc.uvic.ca. For example, the student having Netlink ID `bonniehenry` would have their CSC 360 repository at this location:

`https://gitlab.csc.uvic.ca/courses/2021011/CSC360/assignments/`**`bonniehenry`**`/csc360-coursework`

Please form the address of your repository appropriately and perform a git clone in environment you have provisioned as part of assignment #0. You are also able to access this repository by going to `https://gitlab.csc.uvic.ca` (and use your Netlink username and password to log in at that page).

Goals of this assignment

1. Write a C program implementing a very simple Unix shell.
2. Describe your completed work in a `README.md` file submitted along with your solution for this assignment.

Assignment goal: Write `sh360` (UVic Shell)

You are to design and implement a simple, interactive shell program named `sh360`. Your solution must exist within one C-language source-code file (i.e., `sh360.c`), and both compile and run in your Virtualbox/Vagrant installation.

Even a partial implementation of the functionality in shells such as `bash` or `csh` is beyond the scope of what can be accomplished for the time you have available to complete something like `sh360`. Features such as *globbing*, *job control*, and mixing *background processing* with *redirection* are what make shells such powerful tools for working programmers. However, the purpose of this assignment is to help you understand how to spawn processes and work with the Unix filesystem, and therefore you will implement a much smaller set of shell functionalities. Your shell must provide the following four features:

a. Repeatedly prompt the user for commands, and execute those commands in a child process. The characters to be used in the prompt will make up the first line of a file named `.sh360rc` and note the period in the filename. This file is located in the same directory in which `sh360` itself is executed. (Note: To see `.sh360rc` in a directory listing, you must use the "a" option, for example, "`ls -la`".) A sample version of such a file can be found on `linux.csc.uvic.ca` in the directory `/home/zastre/csc360/assign1` (you can use `scp` to copy these files over into your own Virtualbox/Vagrant system)

b. Execute simple commands with at most seven (7) arguments. The directories making up the path used by `sh360` are to be contained in `.sh360rc`. The directories (which will be absolute paths) follow the first line of `.sh360rc` (i.e., the line with the prompt) where there is one directory per line. A suitable error message must be printed if the location of the binary for a command cannot be found. When "exit" is entered at the prompt as the sole command, `sh360` will terminate.

c. If the command is preceded by `OR` and a space, then the file to which command output is to be stored appears at the end of the command following the "`->`" symbol. A suitable error message must be printed if the `OR` command is not properly formed (e.g., missing "`->`" symbol; missing filename after "->"). You may assume the output file overwrites any existing file with the same filename as that given in the command. For example:

```
   ls -1 > out.txt
```

in `bash` is equivalent to

```
  OR ls -1 -> out.txt
```

in `sh360`.

d. If the command is preceded by `PP` and space, then the command itself will consist of at most three commands separated by "->" where the left command's output is to be connected to the right command's input. A suitable error message must be printed if the `PP` command is not properly formed (e.g., missing "->" symbol; missing commands before or after the "->" symbol). For example,

```
  ls -1 | wc -l
```

in *bash* is equivalent to

```
 PP ls -1 -> wc -l
```

in `sh360`.

As another example,

```
 ps aux | grep 'root' | wc –l
```

in `bash` is equivalent to

```
 PP ps aux -> grep 'root' -> wc –l
```

in `sh360`.

To make the shell even simpler you need not worry about mixing together output redirection (`OR`) with pipes (`PP`). Make sure that all errors messages generated by `sh360` itself are output to `stderr`.

In order to help you with programming, you are not permitted to use memory (i.e., must not use `malloc`, `calloc`, etc.). Please believe me, this is doing you a favour. In keeping with this restriction, you may assume the following limits and are permitted to declare variables having suitable dimensions (e.g., char arrays):

- Maximum number of arguments in any command: 7.
- Maximum length of input line provided by user: 80 characters.
- Maximum length of prompt: 10 characters.
- Maximum number of directories in the path: 10 directories.

There are four further restrictions. ***Violating any one of these may result in a zero grade for the assignment:***

1. You must not use `pthreads` (POSIX threads) in this assignment.

2. After creating a child process, you must use `execve()` when loading the binary for the command, i.e., you are not permitted to use `execvp()` or any other member of the `execv()` family of functions besides `execve()`. (The environment provided to `execve()` will be an array of `char *` with the single value of null, i.e., no `PATH` will be given.)

3. When establishing a pipe between two child processes you must use the `pipe()` system call, i.e., you are not permitted to user `popen()` or any other related system call.

4. Solutions must not cheat by spawning a `bash` subshell process from within `sh360` and then passing along to that process a `bash`-compatible version of the `sh360` command given by the user at the `sh360` prompt.

Completing this assignment will require you to combine string processing with process creation with the use of some system calls (and combination of calls) that are most likely new to you. In addition to this assignment description are several "appendix" programs that I have written as sample code. Each of these focus on a very specific task. You are not required to use this code or my approach, but you should be familiar with the "appendix" code (hint hint demo questions that may be asked hint hint). The "appendix" programs – plus a sample `.sh360rc` file – can be found on `linux.csc.uvic.ca` in the directory `/home/zastre/csc360/assign1` (you can use `scp` to copy these files over into your own Virtualbox/Vagrant system).

What you must submit

- A single C source-code file named `sh360.c` containing the solution to Assignment #1. Any departures from this single source-code solution structure must receive prior written approval from the instructor, but unless there is a compelling reason for such a change I'm unlikely to approve the request. You may also submit your `.sh360rc` file if you wish.

- A single Markdown file named `README.md` describing what you have completed for each of the four features described on a previous page (letters a. through d.).

- These files must be contained within the `a1/` directory of your CSC 360 repository. That is, ensure the all files are added, committed, and pushed. If you wish to verify your push has succeeded, go to:
  `https://gitlab.csc.uvic.ca`
  and browse the contents of your repository.

- Any code submitted which has been taken from the web or from textbooks must be properly cited – where used – in a code comment.

Evaluation

Our grading scheme is relatively simple.

- "A" grade: An exceptional submission demonstrating creativity and initiative, with a solution that is very clearly structured. The `sh360` program runs without any problems.

- "B" grade: A submission completing the requirements of the assignment. The `sh360` program runs without any problems.

- "C" grade: A submission completing most of the requirements of the assignment. The `sh360` program runs with some problems.

- "D" grade: A serious attempt at completing requirements for the assignment. The application runs with quite a few problems.

- "F" grade: Either no submission given, or submission represents very little work.

Please note that software-similarity tools will be used this semester to detect plagiarism and inappropriately-shared code.