CSC 360: Operating Systems
Summer 2017

Assignment #1
Due: Thursday, June 1, 2017 at 11:55 pm via conneX submission

Programming Platform

For this assignment **your code must work on *linux.csc.uvic.ca***. You may already have access to your own Unix system (e.g., Ubuntu, Debian, macOS with MacPorts, etc.) yet we recommend you work as much as possible while connected to linux.csc rather than depending for assignment completion on your machine for later uploading to UVic. Bugs in systems programming tend to be platform-specific and something that works perfectly at home may end up crashing on a different hardware configuration. For example, the contents of stack memory on your computer may differ from what happens on *linux.csc*, and code that works perfectly on your machine may therefore fail dramatically at UVic. (We cannot give marks for submissions of which it is said "It worked on my machine!")

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).

Goals of this assignment

1. Write a C program implementing a very simple Unix shell.
2. Demonstrate your work to a member of the CSC 360 teaching team and explain your chosen data structures and algorithms that appear in the source code of your solution.

Assignment goal: Write *uvsh* (UVic Shell)

You are to design and implement a simple, interactive shell program named *uvsh*. Your solution must exist within one C-language source-code file (i.e., *uvsh.c*), and both compile and run on linux.csc.uvic.ca.

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 *uvsh*. 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 *.uvshrc* – note the period in the filename. This file is located in the same directory in which *uvsh* itself is executed. (Note: To see *.uvshrc* in a directory listing, you must use the "a" option, e.g., "ls -la".)

b. Execute simple commands with at most nine (9) arguments. The directories making up the path used by *uvsh* are to be contained in *.uvshrc*. The directories (which will be absolute paths) follow the first line of *.uvshrc* (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, *uvsh* will terminate.

c. If the command is preceded by *do-out* and a space, then the file to which command output is to be stored appear at the end of the command following the "::" symbol. A suitable error message must be printed if the *do-out* command is not properly formed (e.g., missing "::" symbol; missing filename after "::"). You may assume that 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 "*do-out ls -1 :: out.txt*" in *uvsh*.

d. If the command is preceded by *do-pipe* and space, then the command itself will consist of two commands separated by "::" where the left-most command's output is to be connected to the right-most commands input. (Pipes will *not* be nested within pipes in *uvsh*.) A suitable error message must be printed if the *do-pip*e 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 "*do-pipe ls -1 :: wc -l*" in *uvsh*.

To make the shell even simpler you need not worry about mixing together output redirection (*do-out*) with pipes (*do-pipe*). Make sure that all errors messages generated by *uvsh* 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.). In keeping with this, 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: 9.
- 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 *uvsh* and then passing along to that process a *bash*-compatible version of the *uvsh* command given by the user at the *uvsh* 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 *.uvshrc* file – can be found on *linux.csc.uvic.ca* in the directory */home/zastre/csc360/a1*.

What you must submit

- A single C source-code file named *uvsh.c* containing the solution to Assignment #1. Any departures from this single source-code solution structure must receive prior approval from the instructor, but unless there is a compelling reason for such a change I'm unlikely to approve it.

- Submit the file electronically to conneX.

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

Evaluation

Given that there are a variety of possible solutions to this assignment, the teaching team will not evaluate submissions using a marking script. Students will instead demonstrate their work to our course marker. Sign-up sheets for demos will be provided a few days before the due-date; each demo will require from 10 to 15 minutes.

Our grading scheme is relatively simple.

- "A" grade: An exceptional submission demonstrating creativity and initiative. The *uvsh* program runs without any problems. Any functionality in addition to what is required for the assignment must be in a version of the program named *uvshplus.c*.

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

- "C" grade: A submission completing most of the requirements of the assignment. The *uvsh* 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.