CSE410: Operating System (Spring 2018)
Project #2: Unix Command Interpreter (shell) Implementation

Deadline

Thursday, Feb 22 2018 at 11:59pm (Handin)

Assignment Goals

Get familiar with operating system process handling and execution.

Assignment Overview

In this project, you are required to design and implement a C++ program that serves as a rudimentary command-line interpreter (shell).

Assignment Specifications

In this project you are required to implement the following tasks (total 100 points).

1. **./myShell –c <command> (15 points)**

   Let "myShell" be the name of your executable. After invoking the program, it will execute the <command> and then terminate after executing the command. For example:

   ```
   ./myShell -c ls -l
   ```

   The program (myShell) will execute the command "ls –l" and, then, terminate.

2. **The program should display a prompt repeatedly (5 points)**

   The program will repeatedly display a prompt containing the sequence number of the current command (starting at 1), the name of the machine, and the username of the user executing the program. This information will be enclosed in the characters '<' and '>'. For example:

   ```
   <1 cse410:userName>
   ```

   Where, the sequence number and username are separated by a single space; the machine name and username are separated by a colon. After displaying the prompt, the program will read one line of input from the user. An input line is defined as a sequence of zero or more tokens (character strings) separated by one or more delimiters (blanks and tabs) and ends with a newline character. The characters '&', '|', '<' and '>' are defined to be separate tokens. There will be no more than 128 characters in a line.

   We have two types of commands:

   1) If the first token is the name of a built-in command (shown below), then the program will take the appropriate action.
   2) Otherwise, the program will assume that the command is the name of a file containing an executable program.

3. **Built-in Commands (25 points)**

   The program will recognize the following built-in commands

Command Name	Description
hist	displays the ten most recent input lines
cd Dir	moves from the current directory to Dir
curr	displays the absolute path of the current directory
date	displays the current date and time (e.g. Fri Jan 30 00:17:54 2018)
printenv	prints out all the environmental variables associated with the process
quit	terminates the current process

Built-in commands will be completely processed by the program (the program will not create a child process to perform the processing). Also note that the command "hist" will display the ten most recent input lines entered by the user (along with the associated sequence number). An example of partial "hist" output:

```
97  hist
98  cd /usr/include
99  ls
100 lsa
101 cd /home/userID
```

The history list will hold up to 10 non-null input lines, including any lines with misspelled commands (such as command #100).

4. **External Commands (10 points)**
   For external commands, the program ***will create a child process***; the child process will use a member of the "exec" family of system calls to execute the external command. Be sure to check for possible errors with the "exec" command, don't assume it will execute.
5. **Execute external commands in the background (5 points)**
   If the last token is an ampersand ('&') and the command is an external command, the program will execute the command in the background. That is, the program will not wait until the completion of the given external command before continuing with its processing.
6. **Pipelined external commands (15 points)**
   The user may give commands (external command) in a pipeline, which is defined as a sequence of two and more commands separated by the token '|'. For example: **`ls -l | grep lib | wc -l`**
7. **Input/Output Redirection (15 points)**
   User may specify input output redirection using < and > respectively. For example:
   `ls > testfile`
   We will save the output of "ls" command in the file of "testfile".
8. **Proper Error Handling (10 points)**
   Your program is required to have a proper error handling, and appropriate output. The code should be thoroughly commented with each function clearly declared.
9. **Special Challenge (Extra Credits, 10 points)**
   Extra bonus (10 points) is considered if your shell interpreter could support the combination of I/O redirection and pipelining. For instances:
   `ls -l | grep lib > myfile`
   `cat < myfile | grep myTarget > myfile1`

Assignment Deliverables

A zip file named according to your NetID (e.g. john9999.zip where "john9999" is the NetID) containing the following files.

****.cpp*** – all the .cpp files we provided in "Project_Skeleton" folder together with those your created if any.

****.h*** – all the .h files we provided in "Project_Skeleton" folder together with those your created if any.

Makefile – make file for generating an executable "myShell"

Readme – if you have implemented the task for extra credits or anything you want us to know when grading the project, please mentioned about it in this file

The code should work on "cse410.cse.msu.edu". Any compilation error as well as segmentation fault problem would not be compromised. So please make sure your program is clear from any compilation and runtime errors before submission. Note that your project file should be submitted with specified file name via the "Handin" system.

Assignment Notes

1. You can find all the required source files by downloading the "Project2.zip" file. In this archive file, you can find the following items.
   - Makefile
   - *.cpp
   - *.h
   - myShell (this is for demonstration and you can play with it by executing it on "cse410.cse.msu.edu" for further understanding about the required tasks of this project) **NOTE: do NOT hand in this file please.**
   - lab03* files
   - lab04* files
   - lab06* files

2. You may refer to Lab03, 04, 06 for some introductory information.

3. The following manual pages may be of interest for this assignment
   - man –s 3 getenv
   - man –s 3 localtime
   - man –s 3 getcwd
   - man –s 3 exec
   - man –s 2 fork
   - man –s 2 wait
   - man –s 2 waitpid
   - man –s 2 pipe
   - man –s 2 close
   - man –s 2 dup
   - man –s 3 dup2

4. If you are not familiar with the "make" utility, you may wish to take a look at the following: *~cse410/General/intro.make*

5. All the functions you have to implement for this project are as follows.
   - MyShell::ExecuteCShellCommand()
   - MyShell::Prompt()
   - MyShell::PrintPromptInfo()
   - MyShell::PrintHistory()
   - MyShell::PrintEnv()
   - MyShell::UpdateHistory()
   - MyShell::GetUserName() - MyShell::GetHostName()
   - MyShell::GetTime()
   - MyShell::GetCurrentDirectory()
   - MyShell::ChangeCurrentDirectory()
   - MyShellProcess::ExecuteInBackground()
   - MyShellProcess::Execute()

6. Program flow:
   a) Command line input: “myShell -c <command>”. In this instance <command> is to be interpreted as an external command.
   b) Command line input ==> “myShell”. This kicks off the interactive shell. In the interactive shell, if the user enters a non-null command which is not one of the internal commands it will be treated as an external command. This necessitates use of the "fork()" call, with the possible use of I/O redirection, pipes, and dup calls as determined by the content of the command. The child process will handle the "exec()" call, again be sure to check for an error and terminate the child if necessary. Whether or not the parent waits for the termination of the child process depends on the presence of the ampersand ("&"). If the last token in the command line is a "&" the parent doesn't wait, otherwise it waits. When done the (parent) code moves back to the "wait for user input state".

      **It is imperative that the child process terminates, otherwise you'll have multiple copies of "myShell" executing (i.e. you'll have "fork bombed" yourself) and face the very real possibility of having your account run out of resources. Should this happen your current login activity will seize up and you won’t be able to log in.

   c) Any other command line input, e.g. “myShell <command>”, is to be treated as an error.