Programming Assignment 1 (Simple Shell) [**30 pts + Bonus 10 pts**]

COMP 4270: Operating Systems

Spring 2019

Due: 02/28/2018

Objectives

By completing this programming assignment, you will be familiarized with Unix/POSIX system calls related to processes, file management, and inter-process communication (Bonus).

Task 1 (Teaming Up)

You can do this programming assignment alone or in a pair. If you decide to do this assignment in a pair, find a team member and send the names of your team to the TAs:

Mr. Quang Minh Tran (qmtran@memphis.edu)

Ms. Diem-Trang Pham (dpham2@memphis.edu)

You will do the rest of the programming assignments with your team member. Note that only one of you will hand in the assignment.

Task 2 (Warming Up)

- Install a POSIX-compliant operating system such as Ubuntu (recommended), Linux, MacOS. You can use a free virtualization software such as VMWare Player or VirtualBox to install a POSIX-compliant operating system on top of that. Note that your submitted program will be graded on a POSIX-compliant operating system. Thus, it is crucial for you to compile and test your program in this environment.

- In this programming assignment, you are allowed to use **C, C++, or Python**. If you decide to learn about programming in C/C++ in Linux/Unix environment, the following links may be helpful:
  https://linuxconfig.org/c-development-on-linux-introduction-i

  http://cseweb.ucsd.edu/classes/fa09/cse141/tutorial_gcc_gdb.html

  http://teaching.csse.uwa.edu.au/units/CITS2230/resources/gdb-intro.html

Task 3 (Programming)

Write a program that implements a simple shell. The shell takes a user command as input and executes the command. When the shell is started, it should take a user command as input, execute it and display the output. The following example shows an execution of the shell. It displays the command prompt 'uofmsh' and takes the user command 'ls' as input from STDIN. It then executes the command 'ls' and prints the output to STDOUT.

```
$./uofmsh

uofmsh> ls

*** output of ls ***

uofmsh> exit

$
```

Basically, your shell launches a program (*e.g.,* 'ls') and coordinates the input and output of the program. Note that you should implement your own shell and you are NOT allowed to create a wrapper shell using a library function **system()** (for C/C++). In this programming assignment, you should complete 4 subtasks.

Subtask 1 (Implementing Built-in Commands) [10 pts]

- Implement command parsing in your shell. You do not need to write a full parser in yacc/lex to parse commands. Using plain string functions such as strtok (e.g., for C/C++) should suffice. Once a command is parsed, the parsed command should be executed.
- Built-in command **exit** **[5 pts]:** Your shell should terminate when the exit command is parsed and executed in your shell. For example,

  ```
  $./uofmsh

  uofmsh> exit

  $
  ```

- Built-in command **stat** **[5 pts]**: Your shell should display the history of commands received from the user along with the system time when it was input by the user. For example,

```
$./uofmsh

uofmsh> stat

11:34 exit

11:32 pwd

11:33 cd

$
```

Subtask 2 (Executing Programs) [10 pts]

- Review course slides on the system calls for process creation and replacing the memory space of a process with a new program. Read the man pages of **fork()** and **exec()** for more details on the system calls.

  fork(): http://man7.org/linux/man-pages/man2/fork.2.html

  exec(): https://linux.die.net/man/3/exec

  Note that there are different versions of the **exec()** system call. You will have to decide which version of **exec()** to use in this programming assignment.

  If you decide to use Phyton, here are some useful resources for you.

  https://www.python-course.eu/forking.php

- Basically, your shell should execute a program specified by the user input. To execute a program, a new process should be forked and its memory space should be replaced by the user specified program. Test your shell with programs with different number of arguments. For example,

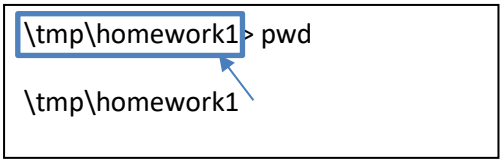
```
$./uofmsh

uofmsh> ls

*output of ls

uofmsh> touch f1

*output of touch f1

uofmsh> cp f1 f2

*output of cp f1 f2

uofmsh> rm -i f1 f2

*output of rm -i f1 f2
```

Subtask 3 (Changing Directory) [10 pts]

- Let's add support for changing the working directory. Implement the command **cd** in your shell using the **chdir(2)** system call. You can find more information about the system call here:

  http://man7.org/linux/man-pages/man2/chdir.2.html

- Make sure that '**cd -**' should change the directory to the last directory that the user was in, and '**cd**' with no arguments should change the directory to the user home directory. Also make sure to implement '**cd .**' that leaves you in the current directory and '**cd ..**' that moves up one directory.
- Verify that the **cd** command works correctly by using the '**pwd**' command which displays the current working directory.
- Display the current working directory in your shell prompt. For example,



Bonus: Subtask 4 (Implementing Redirection) [10 pts]

- Redirection is one of the most powerful features of Linux/UNIX systems. Special characters (i.e,. '<', '>' ,'|') are used to accomplish redirection. Read more information about the redirection here:
  https://www.digitalocean.com/community/tutorials/an-introduction-to-linux-i-o-redirection
- Review the following system calls related to file management: open(), close(), read(), write(), dup()/dup2().
  open(): http://man7.org/linux/man-pages/man2/open.2.html
  close(): http://man7.org/linux/man-pages/man2/close.2.html
  read(): http://man7.org/linux/man-pages/man2/read.2.html
  write(): http://man7.org/linux/man-pages/man2/write.2.html
  dup/dup2: http://man7.org/linux/man-pages/man2/dup.2.html
- To implement support for redirection, you should modify your parsing code such that these three special characters are identified and shell-level directives are individually parsed, rather than passing the entire command to **exec()**. For example, given a command 'ls > file', your shell should parse this command into 'ls', '>', and 'newfile'. The output of 'ls' will be saved in a file 'newfile'. If the file 'newfile' does not exist, your shell will create this file using a system call.
- Test your shell with the following commands:

```
\tmp\homework1> ls -l > file

*the output of 'ls-l' should be saved in a file 'newfile'.

\tmp\homework1> cat < file

*the contents of the file 'newfile' should be displayed.

\tmp\homework1>ls -l | wc -m

*the number of characters in the output of 'ls -l' should be displayed
```

What to turn in

- Submit all files that are needed to run your shell to the dropbox of eCourseware.
- Leave a short note on how to execute your program.

Evaluation criteria

- Your assignment will be evaluated based on the following:

  Documentation 10% - your code should be easy to read and well commented. For each function used in your program, the use of function, its parameters, and return values should be well described.

  Compilation 20% - your program should compile with no errors and/or warnings (base points)

  Correctness 70% - the output of your program should be correct

  - Check that your program produces correct output at least for the examples introduced in this document.

References

COMP 530: Operating Systems at UNC Chapel Hill (http://www.cs.unc.edu/~porter/courses/comp530)