Programming Assignment 1 (Simple Shell) [**25 pts**]

COMP 4270: Operating Systems

Fall 2018

Due: 10/14/2018

Objectives

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

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 now. 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), UNIX, MacOS. If you use other OS, install a free virtual machine software such as VMWare Player or VirtualBox, and install Ubuntu on top of that. Refer to the following link for instructions about how to install Ubuntu:

  https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop#0

- You should use the C programming language for this assignment and subsequent programming assignments. Take some time to review C programming in a Linux/UNIX environment if you have not used it for some time. Also, since you are expected to write, compile, and debug your code in a Linux/UNIX environment, review some resources provided below to familiarize with programming in a Linux/UNIX environment.

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

```
$./uofmsh

uofmsh> ls

*** output of ls

uofmsh> exit

$
```

Basically, you 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()**. In this programming assignment, you are given 4 subtasks to solve.

Subtask 1 (Implementing Built-in Commands) [5 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 should suffice. Once a command is parsed, it should be processed. Let's start with a simple built-in command **exit**. Specifically, your shell should be able to parse the **exit** command, and when the **exit** command is input by the user, your shell should terminate. See an example of execution below:

  ```
  $./uofmsh

  uofmsh> exit

  $
  ```

Subtask 2 (Executing Programs) [5 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 flavors of the **exec()** system call. You will have to decide which version of **exec()** to use in this programming assignment.

- 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 program. Test your shell with programs with different number of arguments:

```
$./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) [5 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 by using the '**pwd**' command which displays the current working directory.
- Display the current working directory in your shell prompt. For example,

```
\tmp\homework1> pwd

\tmp\homework1
```

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
```

5. What to turn in

- Submit your uofmsh.c file to the dropbox of eCourseware.

6. 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

7. References

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