COMP 304 myshell: Project 1

Due: Sunday March 19th, 6.00 pm

Notes: The project can be done **individually or teams of 2**. You may discuss the problems with other teams and post questions to the OS discussion forum but the submitted work must be your own work. This assignment is worth 10% of your total grade. START EARLY.
Any material you use from web should be properly cited in your report. Any sort of cheating will be harshly PUNISHED.

Corresponding TA for the project : Nufail Farooqi and Najeeb Ahmad

Description

The main part of the project requires development of an interactive Unix-style operating system shell, called **myshell** in C/C++. After executing **myshell**, **myshell** will read both system and user-defined commands from the user. The project has four main parts:

Part I

(20 points)
The shell must support the following:

- Use the skeleton program provided as a starting point for your implementation. The skeleton program reads the next command line, parses and separates it into distinct arguments using blanks as delimiters. You will implement the action that needs to be taken based on the command and its arguments entered to **myshell**. Feel free to modify the command line parser as you wish.
- Commandline inputs should be interpreted as program invocation, which should be done by the shell **fork**ing and **exec**ing the programs as its own child processes. Refer to Part I-Creating a child process from Book, page 155.
- The shell must support background execution of programs. An ampersand (&) at the end of the command line indicates that the shell should return the command line prompt immediately after launching that program.
- Use execv() system call (instead of execvp()) to execute UNIX commands (e.g. ls, mkdir, cp, mv, date, gcc) and user programs by the child process.

The descriptions in the book might be useful. You can read Project 1- Unix Shell Part-I in Chapter 3 starting from Page 154.

Part II

(20 points)
In this part your are required to implement a history feature in **myshell**.

- Read Part II- Creating History Feature of Project-1 in Chapter 3 and implement the history feature as described in the book.
- In addition, you are also required to support scrolling through history using up and down arrow keys. First up arrow key press should display the most recent command at the prompt while first down arrow key should display the oldest command in the history. Subsequent up or down arrow key press should scroll through the history likewise.

Part III

(40 points) In this part of the project, you will implement four new **myshell** commands:

- The first command is the **bookmark** command. This feature will enable users to bookmark frequently used commands. See the following example to add a command to the bookmarks and execute it.

```
1       myshell> bookmark "cd /home/users/xxx/yyy/zzz"
2       myshell> bookmark "ssh dunat@lufer.hpc.ku.edu.tr"
3       myshell> bookmark -l
4            0 "cd /home/users/xxx/yyy/zzz"
5            1 "ssh dunat@lufer.hpc.ku.edu.tr"
6       myshell> bookmark -i 0
7       /home/users/xxx/yyy/zzz>
8       myshell> bookmark -d 0
9       myshell> bookmark -l
10           0 "ssh dunat@lufer.hpc.ku.edu.tr"
```

  In line 1, a command to change directory is bookmarked at index 0 and the next command is bookmarked at index 1. Using *-l* lists all the bookmarks added to **myshell**. *bookmark -i idx* executes the command bookmarked at index *idx*. Using *-d idx* deletes the command at index *idx* and shifts the successive commands up in the list.

- The second command is called *muzik*. This command will take a time and a music file as arguments and play the song every day at given time. In order to implement the *muzik* command, you may want to use the *crontab* command provided by Linux. *muzik -l* should list the current play list with their times. *muzik -r time* should remove the crontab entry for that time.

```
1 myshell> muzik 7.15 seninle_bir_dakika.wav
```

  Before implementing the new command inside **myshell**, you should get familiar with crontab (if you decide to use crontab). Feel free to pick your favourite song to play.

  More info about crontab:
  http://www.computerhope.com/unix/ucrontab.htm

- The third command is called **codesearch**. This command is very useful when you search a keyword or phrase in source codes. The command takes a string that is going to be searched and searches this string in all the files under the current directory and prints their line numbers, filenames and the line that the text appears. If -r option is used, the command will recursively search all the subdirectories as well. The file formats searched by the command are limited to .c, .C, .h, .H, .cpp, .c++. Here is an example:

```
myshell> codesearch "foo"
45: ./foo.c -> void foo(int a, int b);
92: ./foo.c -> foo(a,b);

myshell> codesearch -r "foo"
45: ./foo.c -> void foo(int a, int b);
92: ./foo.c -> foo(a,b);
112: ./include/util.h -> void foo(int a, int b, int c);
254: ./lib/x86/lib64.cpp -> for (int i=0; i < N; i++) // foo is called inside
```

- The fourth command is any new **myshell** command of your choice. Come up with a new command that is not too trivial and implement it inside of **myshell**. Be creative. Selected commands will be shared with your peers in the class. Note that many commands you come up with may have been already implemented in Unix. That should not stop you from implementing your own.

Part IV

(20 points) This part of the project requires you to successfully complete problem 4 from Assignment I. You will again write a kernel module this time to display certain characteristics of a process by using **myshell**.

- First design a kernel module that outputs the following characteristics of the processes:

  - PID (process ID)
  - its parent's ID
  - executable name,
  - its children list (their process ids and executable names),
  - its thread group list (their ids and executable names)
  - user ID,
  - process voluntary and involuntary context switch count
  - Its nice (priority)
  - Its vruntime

  You need to read through the *task_struct* structure in $< linux/sched.h >$ to obtain necessary information about a process.

- Test your kernel module first outside of **myshell**

- The kernel module should take a PID and new priority as arguments. The process' dynamic priority will be changed based on these arguments.
- Then define a new command called *processInfo*. The kernel module should be triggered via this new command that you will define in the **myshell** shell.
- The new command should accept the arguments (all integers) for the kernel module such as

```
myshell> processInfo PID prio
```

  which should internally call:

```
myshell> sudo insmod processInfo processID=PID processPrio=priority
```

- When the command is called for the first time, **myshell** will prompt your sudo password to load the module. Once the module is loaded, print the process info from the module to the screen.
- Successive calls to the command will notify the user that the module is already loaded.
- If successive calls use a different process ID, then **myshell** will unload the previously loaded kernel module and load it again with the new arguments.
- If no process ID is provided or the process ID is invalid, print an error message to kernel log.
- **myshell** will remove the module when the **myshell** shell is exited.

Useful References:
- Info about task link list (scroll down to Process Family Tree):
http://www.informit.com/articles/article.aspx?p=368650
- Linux Cross Reference:
http://lxr.free-electrons.com/source/include/linux/sched.h
- Info about scheduler in Linux
Search for sched_setscheduler()
- You can use **pstree** to check if the children list is correct. Use -p to list the processes with their PIDs.
- Even though we are not doing the same exercise as the book, Project 2 - Linux Kernel Module for Listing Tasks discussion from the book might be helpful for implementing this part.

Deliverables

You are required to submit the followings packed in a zip file (named your-username(s).zip) to blackboard :

- .c source code file that implements the **myshell** shell. Please comment your implementation.
- .c course code file that implements the kernel module you developed in Part IV.
- any supplementary files for your implementations (e.g. Makefile)
- a short REPORT briefly describing your implementation, particularly the new command you invented in Part III. You may include your snapshots in your report.
- Do not submit any executable files (a.out) or object files (.o) to blackboard.
- Finally your team will perform a demo of your **myshell** implementation to TAs after the project submission deadline.

Final Notes: The book says the project can be completed on any Unix-based platform. We require the project to be done on a Unix-based virtual machine or on a Linux distribution.

GOOD LUCK.