COMP 304 Kush Shell: Project 1

Due: Thursday March 10th, 11.00 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

Description

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

Part I

(30 points)
The shell must support the following internal commands:

- 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.

  In addition, **kush** should support the followings:

- *cd* command. *cd* should change the working directory. This command should also change the PWD environment variable.
- *clear* command that clears the screen.
- *env* command that list all the environment variables.
- *echo* < *comment* > command that displays the comment on the display followed by a newline.
- 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 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 **kush**. Feel free to modify the command line parser as you wish.
- 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. You are *not* asked to implement the history feature in this project.

Part II

(20 points) In this part of the project, you must support pipes and I/O redirection for stdout. That is, the command line will execute the program with arg1 and arg2, the stdout file stream replaced by outputfile.

```
kush> program arg1 arg2 > outputfile
kush> ls | less
```

With output redirection, if the redirection character is > then the output file is created if it doesn't not exist and truncated if it does. If the redirection token is >> then outputfile is created if it does not exist and appended to if it does.

Part III

(30 points) In this part of the project, you will implement two new **kush** commands:

- The first command is *trash*. This command will take a time and a path as arguments and periodically cleans the files under the path every day at given time. In order to implement *trash*, you may want to use the crontab command. *trash -l* should list the current trash list with their time and paths. *trash -r path* should remove the crontab entry for that path.

```
kush> trash 7.15 /home/users/xxx/Downloads
```

  Before implementing the new command inside **kush**, you should get familiar with crontab (if you decide to use crontab).

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

- The second command is any new **kush** command of your choice. Come up with a new command that is not too trivial and implement it inside of **kush**. Be creative. Selected commands will be shared with your peers in the class.

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 **kush**.

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

  - PID (process ID)
  - its parent
  - executable name,
  - its sibling list (their process ids and executable names),
  - user ID,
  - its scheduling related attributes: its dynamic (priority), static priority, time slice, scheduling policy

  You need to read through the *task_struct* structure in $< linux/sched.h >$ to obtain necessary information about a process. When possible, use more descriptive outputs (e.g. policy=0 (SCHED_NORMAL)

- Test your kernel module first outside of **kush**
- The kernel module should take a PID, new scheduling policy and new priority as arguments. The process' scheduling policy and the dynamic priority will be changed based on these arguments.
- Then define a new command called *schedInfo*. The kernel module should be triggered via this new command that you will define in the **kush** shell.
- The new command should accept the arguments (all integers) for the kernel module such as

```
kush> schedInfo PID policy prio
```

  which should internally call:

```
kush> sudo insmod schedInfo processID=PID processSPolicy=policy processPrio=priority
```

- When the command is called for the first time, **kush** will prompt your sudo password to load the module.
- Successive calls to the command will notify the user that the module is already loaded.
- If successive calls use a different process ID, then **kush** 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.
- **Kush** will remove the module when the **kush** shell is exited.

- See the screen shot of a schedInfo module in Blackboard.

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 sibling 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 **kush** 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 **kush** 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 Linux distribution.

GOOD LUCK.