COMP 304- Kabuk - A Personalized Shell: Project 1

Due: Wednesday March 18th, 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.

Corresponding TA for the project : Yahya Hassanzadeh

Description

In this project, you will get more familiar with the Unix system call interface and the shell by implementing several features in a small shell, called *kabuk*. The main part of the project requires development of an interactive Unix-style operating system shell. After executing kabuk, kabuk will read both system and user-defined commands from the user. The project has three main parts:

Part I

(40 points) Read Project 1- Unix Shell and History Feature from the book in Chapter 3 starting from Page 154.

- Perform Part I-Creating a child process (refer to Book)
- 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 kabuk. 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. Using execv() means that you will have to read the $PATH$ environment variable, then search each directory in the $PATH$ for the command file name that appears on the command line.
- Support also the *cd* command. *cd* should change the working directory.
- Perform Part II- Creating History Feature (refer to Book).

Part II

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

```
kabuk> goodMorning 7.15 /home/musics/muse.mp3
```

- The first command is *goodMorning*. This command will take a time and a music file as arguments and set an alarm to wake you up by playing the music using rhythmbox. In order to implement *goodMorning*, you may want to use the crontab command.

  Before implementing the new command inside kabuk, you should get familiar with crontab (if you decide to use crontab) and rhythmbox on a regular shell.

  More info about rhythmbox:
  http://manpages.ubuntu.com/manpages/hardy/man1/rhythmbox-client.1.html
  More info about crontab:
  http://www.computerhope.com/unix/ucrontab.htm

- The second command is any new kabuk command of your choice. Come up with a new command and implement it inside of kabuk. Be creative. Selected commands will be shared with your peers in the class.

Part III

(30 points) This part of the project requires you to successfully complete problem 3 from Assignment I. You will again write a kernel module this time to display certain characteristics of a process by using kabuk.

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

  - PID (process ID)
  - its parent and real parent's ID
  - executable name,
  - its state,
  - its sibling list (their process ids and executable names),
  - its scheduling related attributes: its nice (priority), static priority, policy, time slice

  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. state = 0 (runnable), policy=SCHED_NORMAL)

- Test your kernel module first outside of kabuk

- Then define a new command called *processInfo*. The kernel module should be triggered via this new command that you will define in the kabuk shell.

- The command will **optionally** take a PID as an argument such as processInfo PID

- When the processInfo command is called for the first time, kabuk will prompt your sudo password and 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 kabuk will unload the previously loaded kernel module and load it again with the new process ID.
- Kabuk will remove the module when the kabuk 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
- You can use **pstree** to check if the sibling list is correct.
- Even though we are not asking the same tasks as the book, Project 2 - Linux Kernel Module for Listing Tasks discussion from the book might be helpful for implementing Part III.

Part IV

(5 point extra credit) If you would like to challenge yourself, add the following supports to kabuk.
- Implement I/O redirection (e.g env > env-var-list.txt)
- Implement pipes (e.g. ls | less)

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 kabuk shell. Please comment your implementation.
- .c course code file that implements the kernel module you developed in Part III
- snapshots of sample runs on your terminal showing sample commands and all the user-defined commands
- any supplementary files for your implementations (e.g. Makefile)
- a REPORT file briefly describing your implementation, particularly the new command you invented in Part II. You may include your snapshots in your report.
- Finally your team will perform a demo of your kabuk 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.