Assignment 2: Building your own shell

Goal of the Assignment:

• To make you familiar with system calls available for process management. •
To learn and understand how shell works and then build your own shell.

A few points to note:

There are a couple of things that need to be considered before starting this assignment. The following points will help students to build their concept of system calls and how they are used in a shell while creating a process. Students are encouraged to go through all the points before starting their assignment.

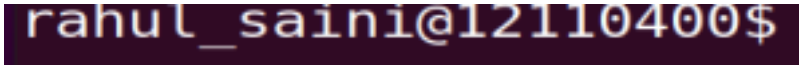
1. Learn about various system calls in Linux. Such as fork, exec, wait, exit. There is documentation available for all these system-level calls. Try looking into the 'man pages', which is quite a good source of learning. Just type "man exec" on the terminal in a Linux system. Online documentation is also readily available (eg. http://manpages.ubuntu.com/manpages/trusty/man2/fork.2.html)
2. There are several variants of exec and wait are available. For learning purposes, students are encouraged to go through them and learn different nuances of usage according to the variant.
3. Before starting this assignment, you must learn the system call usage programmatically. Just for a warmup, try writing a simple program that forks a child process and prints (process id) PID of the child process. Or create multiple child processes and use wait for child processes to wait for one another. This would help you understand the fundamental concept and build your programming aptitude. You can take help of demo programs shown in the classroom.
4. Knowledge of basic shell commands like echo, ls, mkdir, ps, grep, sleep, and so on is important to get familiarized with the Linux environment. You will be using "exec" for executing these commands. Let's be clear, your job is not to implement these commands but to execute basic shell commands. Students should not try to implement basic shell commands separately.
5. Once you understand the basic programming concepts of system calls, try to expand your knowledge using pipes "|". Pipes are basically declared using ("|"). It is used for running multiple shell commands one after another and the output of one command is used as input for the next command. You must have noticed it while using the grep command. For example, try running "ps -aux" and then run "ps -aux | grep bash", this will help you understand the concept of pipe in linux.

Part A: Build a Simple Shell [Total: 10 Points]

This part of assignment is about building a simple shell, just like we see in the Linux terminal. Basic usage of the shell is to take user input and fork child process(es) using fork() system call. After that, an exec() system call is made to execute user commands. You must build a shell like a Linux terminal, which executes all basic built-in commands of Linux e.g., ls, mkdir, echo, cat, etc. These commands are already available in Linux. Your job is to create a program that will use system calls like fork(), wait(), exec(), exit(), etc. to invoke existing executables. It is very important to note that students are not allowed to use any wrapper functions or library functions like "system" which creates a shell environment by simply invoking Linux shell. Doing this is not acceptable and no points will be awarded in this case.

We expect the following things from your shellcode to perform:

1. Your shell prompt should look like "roll_no:firstname_lastname" as shown in the figure below. Instead of the "$" prompt, we expect you to create a custom view of your shell.



2. Your shell should execute all basic commands provided in Linux terminal like *ps, ls, mkdir, netstat,* etc.
3. Make sure your shell support 'cd' command. Hint: chdir system call
4. Your shell must exit by typing "exit" or pressing Ctrl+C. Only this must be used to terminate your program.
5. Your shell should not print any unnecessary strings like "command: ls" or "current command: ls". We expect you to mimic the basic nature of Linux shell.
6. Your shell should take command and its arguments as input and tokenize the string using space as a delimiter. This will allow commands with multiple inputs to run properly.

Students should be able to create a simple Linux shell. Please try not to worry about the corner cases or over-complicated commands. We are not looking for any highly sophisticated program that runs everything. All we expect is that students should invoke exec on any command provided by the user as input. We expect you to learn and play with the code without using any pre designed helper libraries or helper functions.

Part B: Extending shell to support "|" and "&&" features [Total: 20 Points]

This part of the assignment is more about handling rather than creating processes. Here you are

expected to learn and write a program to run multiple processes. In this section you just have to extend your shellcode to support the following operations:

1. Students should parse multiple processes separated by the pipe symbol '|'. The pipe is used for further processing with another command. The main logic is to provide the output of preceding operations as an input to succeeding operations. **[10 Points]**
    - For example, "ps -aux | grep bash" or "ps -aux | grep sh | grep bash". This way you can learn the difference in the output of both commands. There can be multiple pipe symbols, students should not define any limit on pipe symbols

2. After completing the pipe symbol, students should now be able to work on the '&&' symbol. The purpose of this exercise is to get students familiar with the working of the "&&" symbol. Here the process will run sequentially one after the other. The point of focus here is that the first process will run successfully only then the second process will be executed. If in any case, the first process terminates with an error the second process will not be executed. **[10 Points]**
    - For example, "sudo apt-get update && sudo apt-get upgrade". It's quite a popular command for Linux users. Students can use this example or can come up with their own example to run and test their code.

In both the above cases, students should separate commands with '|' or '&&' as delimiters. After that, they can implement their logic for both cases. *It is important to note that students should focus on getting the basic commands running*.

Part C: Extending shell to support the redirection feature [Total: 8 Points]

In this part of the assignment, we want students to work on saving the output into a file. For example, "ls -al > list.txt", using this command, instead of printing everything on the terminal, the output will be saved in the 'list.txt' file. The symbol '>' is used for writing the output to a file. This will indicate that the output of the current command needs to be redirected into a file.

Deliverables in a tar ball on GC:

- Submission Guidelines: Upload the assignment report, code files, README, etc in GC as a tar ball with file name as <your roll no>_<your name>.tar The

folder structure should be in following order. Please follow this format for naming your files.

- 12110400_rahul_saini
  - |_____partA.c
  - |_____partB.c
  - |_____partC.c
  - |_____README
  - |_____REPORT

- Readable Report [2 points for report quality] enumerating steps followed with screenshots for each of the important steps. Put the screenshots in the report for better clarity.

Check Web sources for more information