COMP 304- Operating Systems: Assignment 1

Due: Sunday Mar 7, 11.59 pm

Notes: This is an individual assignment. All forms of cheating will be harshly punished! You may discuss the problems with your peers but the submitted work must be your own work. No late assignment will be accepted. Submit your answers through blackboard. This assignment is worth 4% of your total grade.
Corresponding TA: Fareed Qararyah, for homework related questions, please use discussion forum at Blackboard.

Problem 1

(5x3 points) In Unix/Linux, new processes are created using the **fork()** system call. Calling **fork()** creates a copy of the current process (in which **fork()** is called) and the new process starts executing immediately. After the **fork()** call, both parent and child process start executing the same code.s

- **Part (a)** Write a C program, which calls **fork()** 4 times consecutively. Then each of the forked processes prints its ID, its parents ID and its level in the process tree (The level of the main process is 0). Sample output for 3 forks is shown in Figure 1.

Base Process ID: 30511, level: 0
Process ID: 30512, Parent ID: 30511, level: 1
Process ID: 30513, Parent ID: 30511, level: 1
Process ID: 30514, Parent ID: 30511, level: 1
Process ID: 30515, Parent ID: 30512, level: 2
Process ID: 30517, Parent ID: 30512, level: 2
Process ID: 30516, Parent ID: 30513, level: 2
Process ID: 30518, Parent ID: 30515, level: 3
Figure 1

- **Part (b)** As mentioned previously, the child process is the copy of the parent process when created with **fork()** system call. However, it is possible to replace the contents of the child process with a new process. This is made possible through **exec()** family of system calls. Write a C program that forks a child process which executes **ps f** command. The **ps f** command will display a process tree. While the child process executes the command, the parent should wait for the child to finish. When the child finishes, the parent should print a message *Child finished execution*.
- **Part (c)** Write a C program that forks a child process that immediately becomes a zombie process. This zombie process must remain in the system for at least 5 seconds.

A zombie process is created when a process terminates but its parent does not invoke **wait()** immediately. By not invoking wait(), the child maintains its **PID** and an entry in the process entry table.

- Use the **sleep()** system call API for the time requirement.
- Run the program in the background (using the **&** parameter) and then run the command **ps -l** to determine whether the child is a zombie process.
- The process states are shown below the S column; processes with a state of Z are zombies in the ps command output.
- The process identifier (PID) of the child process is listed in the PID column, and that of the parent is listed in the PPID column.
- You dont want to create too many zombies. If you need, you can kill the parent process with **kill -9 PID**.
- Provide the source code (only .c) and the screenshots of the ps commands.

Problem 2

(10 points) Write a C program on Unix/Linux that creates two children processes. Your program will do the following

- The parent process (A) creates one child process (B).
- The parent process A sends the current time to this child process B by using ordinary pipes.
- The child process B prints the time after receiving it and then the child process B sleeps for 3 secs.
- Then the child process B creates another process C and sends the current time to this child process C using ordinary pipes.
- The child process C prints the time after it receives it.
- The child process C sleeps for 3 secs and then sends the current time to its parent process B using ordinary pipes.
- When the parent process B receives the message, prints the received time.
- The child process B sleeps for 3 secs and then sends the current time to its parent process A using ordinary pipes.
- When the parent process A receives the message, prints the received time.
- The parent B terminates the child process C.
- After that, the parent A terminates the child process B.

- Then the parent terminates.

You may use **sleep(), kill(), gettimeofday()** system call APIs in your assignment. You are required to submit your .c source code file and a snapshot of a sample run on your terminal. You may refer to the ordinary pipe example in the textbook.

Problem 3

(10+15 points) In this question, you will learn how to create a kernel module and load it into the Linux kernel. Note that you need to be a superuser on the computer for this assignment.

a) Read Linux Kernel Modules under the Programming Projects from the book in Chapter 2 (Page 96). Perform Part I and Part II of the assignment. You can arbitrarily set the day, month, year variables. Use the make file and template program we have provided as a starting point.

b) Design a separate kernel module that outputs the following characteristics of the processes:

- PID (process ID)
- its parent's ID
- executable name,
- its sibling list (their process ids and executable names),

You need to read through the task struct structure in $< linux/sched.h >$ to obtain necessary information about a process. The kernel module should take an PID as an argument. If no process ID is provided or the process ID is invalid, print an error message to kernel log.

Some 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:
https://github.com/torvalds/linux/blob/master/include/linux/sched.h
- You can use the **pstree** command to check if the sibling list is correct. Use -p to list the processes with their PIDs.
- Refer this website on how to pass arguments to a kernel module:
http://www.tldp.org/LDP/lkmpg/2.6/html/x323.html

Important Remarks

1. All the programming assignments should be implemented and tested in a Linux Operating System.
2. What to submit: Create a folder with your KUSIS ID and submit the folder as a single zip file. The folder should be organized as follows:

- a. Each problem should be in a separate subfolder
- b. Each subfolder should contain .c source files, screenshots of sample runs and a single README.txt file explaining how to compile and run our code. You may include a makefile if you have one.
- c. **Do NOT submit object files or executables** to Blackboard.
- d. Use the problem number to name your source files. For example, p2a refers to the source code solution for problem 2 part a.
- e. Improper naming or file organizations will result in 5 point penalty in the assignment grade.

GOOD LUCK