COMP 304- Operating Systems: Assignment 1

Due: 17th of February 2020 midnight

Notes: This is an individual assignment, partners are not allowed and cheating is a punishable offense. 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 to Blackboard. This assignment is worth 4% of your total grade.

Corresponding TA: Mandana Bagheri (mmarzijarani20@ku.edu.tr) Office: ENG 110
Office Hours: 9:30-11:30 on Monday and Wednesday, 4-6 pm on Thursday

Problem 1

(10+10 points) Write a C program in Linux that creates one child process, your program should do the following:

Part I:
-The parent process creates a child process and the child process will output the time of the day and its own PID every second.
-The parent process waits for 5 seconds.
-The parent process will kill the child process after 5 seconds and prints “Child <PID> killed.”. Then the parent terminates.

Part II:
-The parent process creates 4 child processes and each child process will output the time of the day and its own PID every second.
-The parent process waits for 5 seconds.
-The parent process will kill the child processes after 5 seconds and prints “Child <PID> killed.”. Then the parent terminates.

You may need to use sleep(), gettimeofday(), kill() and fork() system calls in your assignment. You are required to submit your C source code file along with a snapshot of the execution in terminal for each part.

Problem 2

(20+10 points) Write a C program on Linux that creates children processes and does the following:

Part I:
- The parent process creates 1 child.
- The child process itself creates 1 child. This will be the grandchild of the parent.
- The parent process reads one number (integer) from the standard input. The parent process sends this number to the child using standard pipes.
- The child multiplies the number by 2, prints it as "Child: Input X, Output X*2" and sends the result to the grandchild.
- The grandchild multiplies the input number by 2, prints it as "Grandchild: Input X, Output X*2" and sends it back to the child.
- The child will send the result back to the parent.
- The parent prints the final result.
- The parent kills all children processes, and terminates.

Part II:
Add a sleep(5) parameter to each process operation, so that you can track the behavior as it executes.

Open two terminals. In the first terminal, run the tool ***htop***. Once in htop, press ***t*** to sort all processes as a process tree. Resize the window so that you can see the processes properly.
In the second terminal, execute your program. Find the program in htop, and take a screenshot of its hierarchy (parent/child/grandchild).

Kill the child process amidst execution by pressing *k* in htop, and sending it SIGKILL to immediately terminate it. Take a screenshot of the program execution output.

For Problem 2, you may need to use sleep(), kill() and fork() system calls. Refer to the ordinary pipe example in the textbook (also in Blackboard). You are required to submit your C source code file as well as a snapshot of the execution in terminal for each part.

Problem 3

(15+15 points)
In this program you will use shared memory objects as well as pipes for interprocess communication.

Write a C program on Linux that reads input on standard input, writes the same input on standard output, and also writes the hexadecimal equivalent of the input on standard error.

The program should do this by creating two children, parent sending the input via a pipe to the children, and reading the result of the child from shared memory.

To be more specific, the parent creates two children, child A and child B. The parent sends an identical file to both children through pipe, both children create a separate shared memory object (shared with the parent). Child A's shared memory object size is equal to the size of the input and Child B's should be twice the size of input (for hexadecimal representation).

Child A is responsible for simply writing what it receives through pipe to its shared memory, and child B is responsible for converting anything it receives to hexadecimal format and then writing it to its shared memory.

Once they are done writing the result to the shared memory objects, they send a message 'done' to the parent. Once the parent receives 'done', it reads the content of the shared memory, and for child A outputs to standard output, and for child B, outputs to standard error.

Part I:

Test your program manually. Once you are confident the program works fine, test it using a small input file with the size less than 64KB.

Hint: the file size should be smaller than the pipe size.

Part II:

Repeat the part I but this time instead of using a small input file, test it using a large input file via the command below, which provides the Linux dictionary as the input to the program, and writes the outputs to two separate files, which you can then verify manually:

```
$ ./myprogram </usr/share/dict/words >output-normal.txt
2>output-hex.txt
```

Hint: You will have to read the pipes from parent to children multiple times.

You may need to use sleep(), kill() and fork() system calls. You may want to refer to the ordinary pipe example and shared memory object example in the textbook (also in Blackboard under Lecture notes). You are required to submit your C source code file as well as a snapshot of the execution in terminal for each part.

Problem 4

(10+10 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 120 also provided in Blackboard). Perform Part I and Part II of the assignment. You can arbitrarily set the day, month, year variables. Use the makefile and template program provided as a starting point.

b) Design a separate kernel module that outputs the following characteristics of the processes: - PID (process ID)

- parent
- executable name
- status (running/zombie/stopped/etc.)
- 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 a 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://elixir.bootlin.com/linux/latest/source/include/linux/sched.h

- You can use the **pstree -p** or **htop** command to check if the sibling list is correct.

- Refer to the following website on how to pass arguments to a kernel module: http://www.tldp.org/LDP/lkmpg/2.6/html/x323.html

Submission

Create a folder named KUSIS-ID-A1 then the folder should contain a separate folder for each question. In each question folder, name the source code files with a file name with the question number (e.g., problem1-partII.c).

Zip your KUSIS-ID-A1 as a single file and upload to Blackboard.

******** You are required to submit your .c source code files and a snapshot of a sample run on your terminal for all programming assignments. We will deduct points for those who submit executables or object files. *********