CSC465/665: Operating Systems; Spring 2019

Assignment 1

Due Date: February 20, 2019 -- Due Time: 23:59

Problem 1: Creating Processes using fork() (C/Unix programming + Written Part) [35 points]

a) Write a C program that creates exactly 15 new processes on 4 levels. Your tree should have one process at level 0 (your main program), 4 processes at Level 1, 5 processes at Level 2, and 2 processes at Level 3. Each process should indicate:

- Its process id
- What level it is at in the process tree
- "Who" its parent and grandparent are

b) Visualize what your program did by drawing the process tree that it created, indicating the process id (printed by your program of each process in the tree. Make sure that your tree matches your program. (i.e., each process has the correct id, ancestors and is located at the correct level).

Problem 2: Inter Process Communication using Shared Memory (C Programming) [30 points]

In Linux, a parent process passes information to its child since the child receives a copy of its address space. However, in its raw form, the parent does not have any way access to the child's address space. In this part of the assignment, you will create a program in which a driving school instructor instructs its student on what to do in various situations through Shared Memory.

The student may encounter the following 5 situations stored in an array of strings:

Situations:

a) The light is turning green
b) There is a pedestrian in front of me
c) The car in front of me just stopped
d) The car in front of me is moving
e) The road is turning to the right

The teacher's recommendation for each situation is, respectively:

Recommandations:

a) Press the accelerator
b) Press the break

c) Press the break
d) Press the accelerator
e) Steer right

The student can then do the correct thing or not:

Action:

- when instructed to press the accelerator, s/he may do the correct thing and respond: “Done” or do the wrong thing and respond: “Oops, I pressed the break instead”;
- when instructed to press the break, s/he may do the correct thing and respond: “Done” or do the wrong thing and respond: “Oops, I pressed the accelerator instead”;
- when instructed to steer right, s/he may do the correct thing and respond: “Done” or do the wrong thing and respond: “Oops, I steered left”.

Your program should work as follows:

The parent process (the teacher) creates three arrays of strings called Situation, Recommendation and Action, respectively. I should also create a shared memory region that will contain 1) the index of the situation, recommendation or action that will be shared between the teacher and the student; 2) (a) boolean flag(s) to coordinate the order in which the processes should proceed. It should then create a child (the student) process that maps this shared memory region into its address space. The teacher then waits until the student signals that it is ready. After mapping the shared memory region into its address space, the student selects, at random, the situation it will face. It writes the index of that situation in the shared memory region and prints it to the terminal. It then updates the flag(s) to signal to the teacher that it is ready for a recommendation. When its turn comes, the teacher reads the index of the situation and makes the proper recommendation. It writes the index of the proper recommendation in the shared memory region and prints it to the terminal. It then updates the flag(s) to signal to the child that it is ready. From then on, the teacher waits for the student to take action and terminate. Whether the student takes the correct or incorrect action is determined at random. Once chosen, the student writes the index of its action in the shared memory region and prints that action. It then terminates. Upon the student’s termination, the teacher prints “What a good teacher I am!” or “What a lousy teacher I am!” depending on the action chosen by the student.

The output should look something like:

Student: The car in front of me just stopped
Teacher: Press the break
Student: Oops, I pressed the accelerator instead
Teacher: What a lousy teacher I am!

Problem 3: Inter Process Communication using Pipes (C/Unix Programming) [30 points]

Implement the same program as in Problem 2, but this time, there should be no shared memory and the communication between the parent (teacher) and child (student) should be done entirely through pipes (though signals can also be used if necessary). You will need to create two pipes: one that allows the teacher to send information to the student and one that allows the student to send information to the teacher. The functionality of your program and its kind of output should be entirely the same as in Problem 2.

Problem 5: Formalizing the IPC Strategy of your two programs (Written) [5 Points]

In this part of your assignment, you are asked to reflect on the programs you wrote for Problem 3 and 4 and discuss them in terms of Inter Process Communication Strategies. In particular, make sure that you address the following concepts in your discussion of both problems:

- Direct versus Indirect Communication
- Synchronous/Blocking versus Asynchronous/Non-Blocking Communication
- Automatic versus Explicit Buffering

Also discuss whether you used any other tools to help you solve the problem.

Please discuss which approach you found more appropriate/easier to design for this particular problem. Could you provide examples of tasks for which one approach would be better than the other and vice-versa.