Lab sheet 5- Race condition

Part 1: Race condition

Consider a system with one parent process and two child processes A and B. There is a shared signed integer X initialized to 0. Process A increments X by one 10 times in a for loop. Process B decrements X by one 10 times in a for loop. After both A and B finish, the parent process prints out the final value of X. Declare a shared memory variable to hold X (see the calls shmget(), shmat(), shmdt(), and shmctl() in Linux). Write the programs for processes A and B. Do not put any synchronization code in the code for A and B . You should write the code in such a way so that you can simulate race condition in your program by slowing down A or B appropriately by using sleep() calls at appropriate points. Note that if there is no race condition, the value of X finally should be 0. Simulating race conditions means that if you run the program a few times, sometimes the final value of X printed by your program should be non-zero.

Parent process

- Initialize the shared memory region
- Set the value of x in the shared memory to zero
- Fork processes A & B
- Read the final value of x from the shared memory and print it

Process A

- Read the value of x from the shared memory
- Increment it 10 times
- Write the final value of x to the shared memory

Process B

- Read the value of x from the shared memory
- Decrement it 10 times
- Write the final value of x to the shared memory

Part 2: Peterson's Algorithm

How to avoid the race condition issue in the part 1 using Peterson's algorithm. You can create a shared memory region to hold turn and flag variable in Peterson's solution.