Assignment 3

Goal: To understand and learn about Process Synchronization and to simulate **Race Condition**

This assignment has 3 parts.

Part 1 :

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.

Part 2 :
Add synchronization code based on semaphores toprocess A and B above so that there is no possibility of race conditions. Use the calls semget(), semop(), semctl() in Linux to create and manage sempahores.
Name the C file <your_roll_no>_sem.c (for example, CSCxxxx _sem.c).

Part 3 :
In this program, you'll write a program to solve the m-producer n-consumer problem, m, n>= 1. You have a shared circular buffer that can hold 20 integers. Each of the producer processes stores the numbers 1 to 50 in the buffer one by one (in a for loop with 50 iterations) and then exits. Each of the consumer processes reads the numbers from the buffer and adds them to a shared variable SUM (initialized to 0). Any consumer process can read any of the numbers in the buffer. The only constrain is that every number written by some producer should be read exactly once by exactly one of the consumers.

Of course, a producer should not write when the buffer is full and a consumer should not read when the buffer is empty.

Write a program that first creates the shared circular buffer and the shared variable SUM using the shm*() calls in Linux. You can create any other shared variable that you think you may need. The program then reads in the value of m and n from the user, and forks m producers and n consumers. The producer and consumer codes can be written as functions that are called by the child processes. After all the producers and consumers have finished (the consumers exit after all the data produced by all the producers have been read. How does a consumer know this?), the parent process prints the value of SUM. Note that the value of SUM should be m*25*51 if your program is correct.

Test your program with at least
(a) m=1, n=1,
(b) m=1, n=2,
(c) m=2, n=1, and
(d) m=2, n=2.

Name the C file <your_roll_no>_prod.c(for example, CSCxxxx_prod.c).

Submission: You have to demonstrate your program and its working in Lab hours.

Note: You can directly try this assignment using Thread Implementation as well. I prepared the statement for process based implementation.

Hints:

1. UNIX System Calls and Subroutines using C (Online Tutorial)

http://www.cs.cf.ac.uk/Dave/C/

From this online book read the semaphore, shared memory based programing system calls and fundamentals.

2. Tutorial on Linux IPC mechanism

http://tldp.org/LDP/lpg/node7.html

3. Unix network programming, by R. Stevens, Vol-2

https://github.com/shihyu/Linux_Programming/blob/master/books/W.%20Richard%20Stevens%20-%20Unix%20Network%20Programming%20Vol2.pdf

4. Beej's guide for Unix IPC

http://beej.us/guide/bgipc/html/single/bgipc.html