WEEKS 6 & 7: Producer Consumer Problem Date: 01/10/2020

OBJECTIVE:

PROGRAM 1: Write a C Program to simulate race condition in Producer-Consumer Problem

PROGRAM 2: Write a C program to implement Producer Consumer problem using Mutex (prevent race condition)

- CONCEPTS ARE ALREADY COVERED IN THEORY
- STUDENTS ARE ADVISED TO REFER TO THE TEXT BOOK, LECTURE MATERIAL AND PROGRAMS DEMONSTRATED IN THE CLASS TO IMPLEMENT THE GIVEN PROGRAMS.
- STUDENTS ARE REQUIRED TO PROVIDE PROOF OF CONDUCTION (AS PER SUBMISSION PROCEDURE BELOW).

SUBMISSION:

1. The source code files for both PROGRAM 1 and PROGRAM 2 should be uploaded to EDMODO in WORD or ZIP FORMAT.
2. All the screenshots clearly showing the directory name as YOURSRN_NAME_WEEK6-7 for both the programs should be uploaded to EDMODO in a SINGLE FILE (Word or PDF format only, Do NOT zip this file).

Students should keep these TWO deliverables (i.e. 1 & 2 above) separate and NOT zip all the files together in order to facilitate quick, timely and effective evaluation.

Contact your respective Lab faculty for any questions or clarifications needed.

DUE DATE FOR SUBMISSION: IN VIEW OF ISA 1, SUBMIT BOTH PROGRAM 1 and PROGRAM 2 TOGETHER ON OR BEFORE 22/10/2020 11:59 PM

PROGRAMS FOR EXECUTION AND SUBMISSION:

PROGRAM 1: Write a C Program to simulate race condition in Producer-Consumer Problem

Implement a main program that creates two threads: producer and consumer threads which execute producer and consumer functions respectively. The producer should produce an item and update the buffer. The consumer should consume an item and update the buffer. You can use bounded buffer and both the producer and consumer threads can be infinite loops. Show how race condition occurs between producer and consumer without mutual exclusion.

PROGRAM 2: Write a C program to implement Producer Consumer problem using Mutex. (Do NOT use POSIX Semaphores in this program)

Extend Program 1 by using mutex to enclose the critical sections in both producer and consumer so that only one of them can update the buffer at a time and prevent race condition as shown in sample outputs below. Also, consumer should **wait** if buffer is empty and producer should **signal** when the buffer has at least one item. You can use unbounded buffer so that producer does not have to wait for buffer availability. Both the producer and consumer threads can be infinite loops and each can randomly sleep to let the other proceed. The output should be the number of items in buffer along with the consumer/producer that is updating the buffer.

Sample Outputs:

Your output can vary depending upon your implementation and the manner in which the threads are scheduled on your system.

With race condition	Without race condition
Job 1 started	Job 1 started
Job 2 started	Job 1 finished
Job 3 started	Job 2 started
Job 2 finished	Job 2 finished
Job 4 started	Job 3 started
Job 3 finished	Job 3 finished
Job 1 finished	Job 4 started
Job 4 finished	Job 4 finished
….	…

With race condition

Producer: Produced item 1
Producer: Produced item 2
Consumer: Consumed item 1
Producer: Produced item 3
Producer: Produced item 4
Producer: Produced item 5
Producer: Produced item 6
Consumer: Consumed item 4
Consumer: Consumed item 3
Producer: Produced item 7
....

Without race condition

Producer: Produced item 1
Consumer: Consumed item 1
Producer: Produced item 2
Consumer: Consumed item 2
Producer: Produced item 3
Consumer: Consumed item 3
Producer: Produced item 4
Consumer: Consumed item 4
Producer: Produced item 5
Consumer: Consumed item 5
...