SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya School of Engineering

K. J. Somaiya School of Engineering, Mumbai-77
Somaiya Vidyavihar University
Department of Electronics and Computer Engineering

Somaiya TRUST

Course Name:	Operating Systems and Compilers	Semester:	VI
Date of Performance:	11 / 04 / 2025	Batch No:	B - 2
Faculty Name:	Prof. Nilesh Lakade	Roll No:	16014022050
Faculty Sign & Date:		Grade/Marks:	___ / 25

Experiment No.: 6

Title: Implementation of Process synchronization algorithms using semaphore - producer consumer

Aim and Objective of the Experiment:

To implement Process synchronization algorithms using semaphore - producer consumer problem, reader-writers problem.

COs to be achieved:

CO2: Describe the problems related to process concurrency and the different synchronization mechanisms available to solve them.

Theory:

Description of the chosen process synchronization algorithm:
This program simulates the **Producer-Consumer problem** using a shared buffer with a fixed size. The producer generates items and places them into the buffer, while the consumer takes items from the buffer.

- **Producer**: Can only produce if the buffer isn't full.
- **Consumer**: Can only consume if the buffer isn't empty.
- Both operations are synchronized using a **mutex** to ensure that only one process (producer or consumer) can access the buffer at a time.
- The buffer has a limited number of slots, and the program checks if it's full or empty before performing any action.

The main logic involves:

- The producer produces items and adds them to the buffer.
- The consumer consumes items from the buffer.
- The program ensures the producer doesn't exceed the buffer size and the consumer doesn't attempt to consume from an empty buffer.

Implementation details: (printout of code)

```c
#include <stdio.h>
#include <stdlib.h>

int mutex = 1;
int full = 0;
int empty = 10;
int x = 0;

// producer function
void producer() {
    --mutex;
    printf("Mutex is occupied.");
    ++full;
    --empty;
    x++;
    printf("\nProducer produces item %d", x);
    ++mutex;
    printf("\nMutex is freed.");
}

// consumer function
void consumer() {
    --mutex;
    --full;
    ++empty;
    printf("Consumer consumes item %d", x);
    x--;
    ++mutex;
}

int isBufferFull() {
    return empty == 0;
}

int isBufferEmpty() {
    return full == 0;
}
```

```c
int main() {
    int choice;

    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");

    while (1) {
        printf("\n\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            if (mutex == 1 && !isBufferFull()) {
                producer();
            } else {
                printf("\nBuffer is full! Producer cannot produce more items.\n");
            }
            break;

        case 2:
            if (mutex == 1 && !isBufferEmpty()) {
                consumer();
            } else {
                printf("\nBuffer is empty! Consumer cannot consume more items.\n");
            }
            break;

        case 3:
            printf("\nExiting program...\n");
            exit(0);
            break;

        default:
            printf("\nInvalid choice! Please enter 1, 2, or 3.");
        }
    }
    return 0;
}
```

```
PS C:\Users\admin\OneDrive\Desktop\sem 6\operating system and compilers> cd "c:\Users\admin\One
Drive\Desktop\sem 6\operating system and compilers\" ; if ($?) { gcc producer.c -o producer } ;
 if ($?) { .\producer }

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit

Enter your choice: 1
Mutex is occupied.
Producer produces item 1
Mutex is freed.

Enter your choice: 2
Consumer consumes item 1

Enter your choice: 2

Buffer is empty! Consumer cannot consume more items.


Enter your choice: 1
Mutex is occupied.
Producer produces item 1
Mutex is freed.

Enter your choice: 3

Exiting program...
PS C:\Users\admin\OneDrive\Desktop\sem 6\operating system and compilers>
```

Semaphores were not used in the code because the standard C library does not provide built-in support for semaphores, especially for process synchronization. While semaphores are commonly available in libraries like POSIX or specific OS implementations, the lack of such libraries in this environment prevented their use. Instead, a manual approach using integer flags like mutex, full, and empty was employed to simulate synchronization.

Post Lab Subjective/Objective type Questions:

1. **A semaphore is a shared integer variable**
   a. That can't drop below zero
   b. That can't be more than
   c. That can't drop below one

   Ans: That can't drop below zero.

2. **Mutual exclusion can be provided by the**
   a. Mute locks
   b. Binary semaphores
   c. Both a and b
   d. None of these

   Ans: Both a and b

3. **A monitor is a module that encapsulates**
   a. Shared data structures
   b. Procedures that operate on shared data structure
   c. Synchronization between concurrent procedure invocation
   d. All of the above

   Ans: All of the above

4. **To enable a process to wait within the monitor**
   a. A condition variable must be declared as condition
   b. Condition Variables must be used as Boolean objects
   c. Semaphore must be used
   d. All of the above

   Ans: A condition variable must be declared as condition

Conclusion:

This experiment demonstrates the implementation of process synchronization using semaphores in the Producer-Consumer problem. It ensures mutual exclusion and proper synchronization between the producer and consumer processes, effectively managing shared buffer access.

Signature of faculty in-charge with Date: