Name: Meet Brijwani
Roll no: 14
Batch: S11

EXP 6:

Aim : Write a C program to implement solution of Producer consumer problem through Semaphore.

Theory :

In concurrent programming, concurrency represents a pivotal concept necessary to comprehend fully how such systems operate. Among the various challenges encountered by practitioners working with these systems stands out the producer-consumer problem - one of the most renowned synchronization issues. In this text, our objective consists of analyzing this topic and highlighting its significance for concurrent computing while also examining possible solutions rooted within C.

Introduction

In concurrent systems, multiple threads or processes may access shared resources simultaneously. The producer-consumer problem involves two entities: producers that generate data or tasks, and consumers that process or consume the generated data. The challenge lies in ensuring that producers and consumers synchronize their activities to avoid issues like race conditions or resource conflicts.

Understanding the Producer-Consumer Problem

Problem Statement

One possible definition of the producer-consumer problem involves two main groups: producers of data who store their work in a communal space called the buffer, and processors (consumers) of that content saved in said space. These people use their expertise around gathered items within this temporary holding scenario to analyze it comprehensively before delivering insightful results.

Synchronization Requirements

Achieving resolution of the producer-consumer conundrum will necessarily involve implementing techniques for synchronized collaboration among all stakeholders involved. The integration of optimal synchronization protocols is fundamental in avoiding scenarios where device buffers are either overloaded by producing units or depleted by consuming ones.

Implementing the Producer-Consumer Problem in C

Shared Buffer

In C, a shared buffer can be implemented using an array or a queue data structure. The buffer should have a fixed size and support operations like adding data (producer) and retrieving data (consumer).

Synchronization Techniques

Several synchronization techniques can be used to solve the producer-consumer problem in C, including −

- Mutex and Condition Variables − Mutexes provide mutual exclusion to protect critical sections of code, while condition variables allow threads to wait for specific conditions to be met before proceeding.
- Semaphores − Semaphores can be used to control access to the shared buffer by tracking the number of empty and full slots.
- Monitors − Monitors provide a higher-level abstraction for synchronization and encapsulate shared data and the operations that can be performed on it.

Solutions to the Producer-Consumer Problem in C

Bounded Buffer Solution

One common solution to the producer-consumer problem is the bounded buffer solution. It involves using a fixed-size buffer with synchronization mechanisms to ensure that producers and consumers cooperate correctly. The capacity for item production is limited by the buffer size making it crucial to factor in this specification so as not to exceed the available space in the buffer.

Producer and Consumer Threads

In C, the producer and consumer activities can be implemented as separate threads. Each producer thread generates data and adds it to the shared buffer, while each consumer thread retrieves data from the buffer and processes it. Synchronization mechanisms are used to coordinate the activities of the threads.

Handling Edge Cases

In real-world scenarios, additional considerations may be necessary. For example, if the producers generate data at a faster rate than the consumers can process, buffering mechanisms like blocking or dropping data may be required to prevent data loss or deadlock situations.

Code :

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


// Initialize a mutex to 1 and Number of full slots as 0 and Number of
empty slots as size of buffer

int mutex = 1;

int full = 0;

int empty = 10, x = 0;



// Function to produce an item and add it to the buffer
void producer()


{

    // Decrease mutex value by 1 and Increase the number of full slots
```

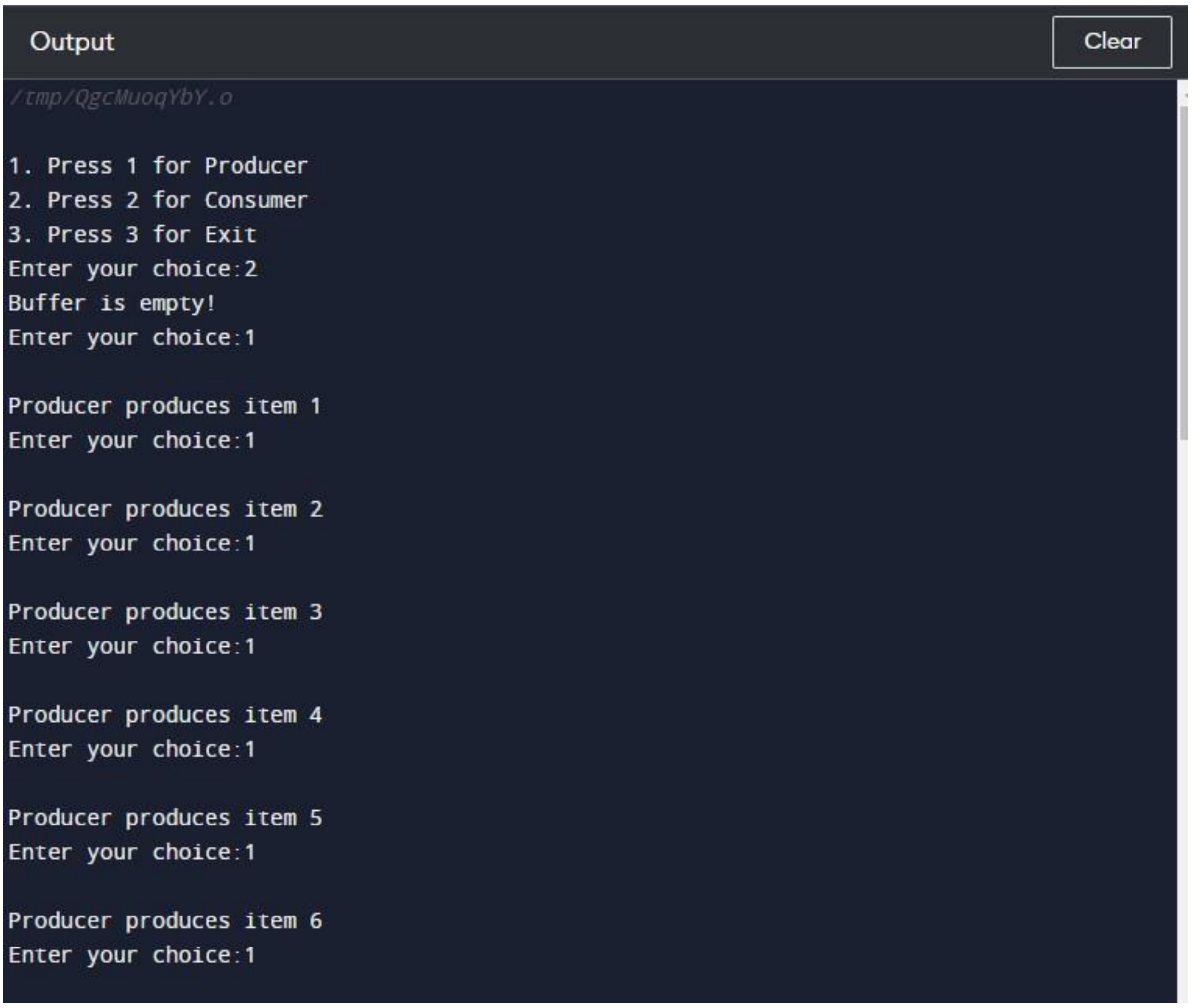
```
by 1 and Decrease the number of empty slots by 1
    --mutex;
    ++full;
    --empty;
```

```
// Item produced
    x++;
    printf("\nProducer produces item %d",x);
// Increase mutex value by 1
    ++mutex;
}

// Function to consume an item and remove it from buffer
void consumer()
{
     // Decrease mutex value by 1 and Decrease the number of full slots
by 1
    --mutex;
    --full;
 // Increase the number of empty slots by 1
    ++empty;
    printf("\nConsumer consumes item %d",x);
    x--;
// Increase mutex value by 1
    ++mutex;
}

int main()
```

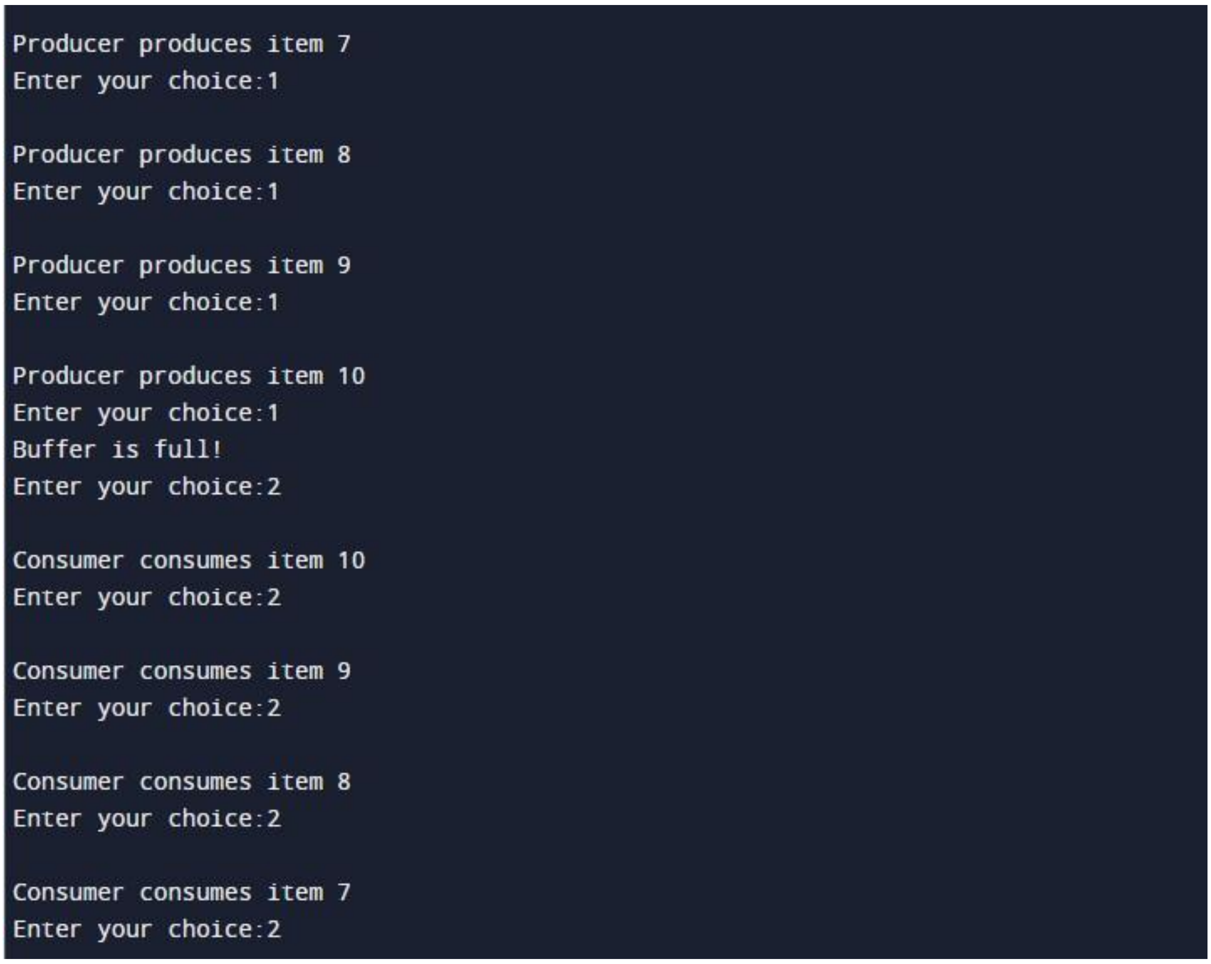
```
{

    int n, i; printf("\n1. Press 1 for Producer \n2. Press 2 for
    Consumer \n3.

Press 3 for Exit");




    for (i = 1; i > 0; i++) {
```

```
        printf("\nEnter your choice:");

        scanf("%d", &n);

        switch (n) {

        case 1:


// If mutex is 1 and empty is non-zero, then it is possible to produce

            if ((mutex == 1)

                && (empty != 0)) {

                producer();

            }

            else { printf("Buffer is
                full!");


            }

            break;


        case 2:

 // If mutex is 1 and full is non-zero, then it is possible to consume

            if ((mutex == 1) && (full != 0)) {

                consumer();

            }
            else {

                printf("Buffer is empty!");

            }

            break;

        case 3:

            exit(0);

            break;

        }

    }

}
```

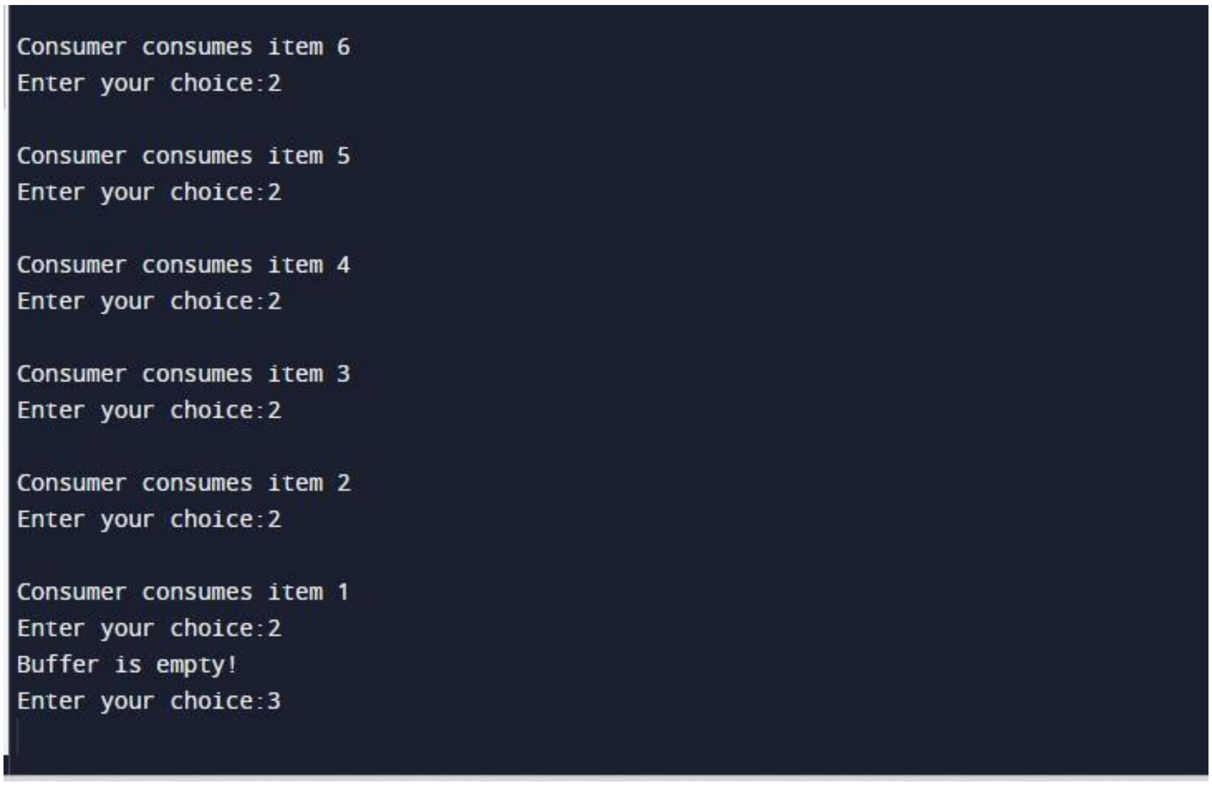
Output :



```
/tmp/QgcMuoqYbY.o

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:2
Buffer is empty!
Enter your choice:1

Producer produces item 1
Enter your choice:1

Producer produces item 2
Enter your choice:1

Producer produces item 3
Enter your choice:1

Producer produces item 4
Enter your choice:1

Producer produces item 5
Enter your choice:1

Producer produces item 6
Enter your choice:1
```



```
Producer produces item 7
Enter your choice:1

Producer produces item 8
Enter your choice:1

Producer produces item 9
Enter your choice:1

Producer produces item 10
Enter your choice:1
Buffer is full!
Enter your choice:2

Consumer consumes item 10
Enter your choice:2

Consumer consumes item 9
Enter your choice:2

Consumer consumes item 8
Enter your choice:2

Consumer consumes item 7
Enter your choice:2
```



```
Consumer consumes item 6
Enter your choice:2

Consumer consumes item 5
Enter your choice:2

Consumer consumes item 4
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!
Enter your choice:3
```

Conclusion : Thus we have successfully implemented Producer Consumer problem using Semaphore.