Name: Meet Raut
Batch: S21
Roll number: 2201084

Assignment 6

AIM- Write a C program to implement solution of Producer consumer problem through Semaphore

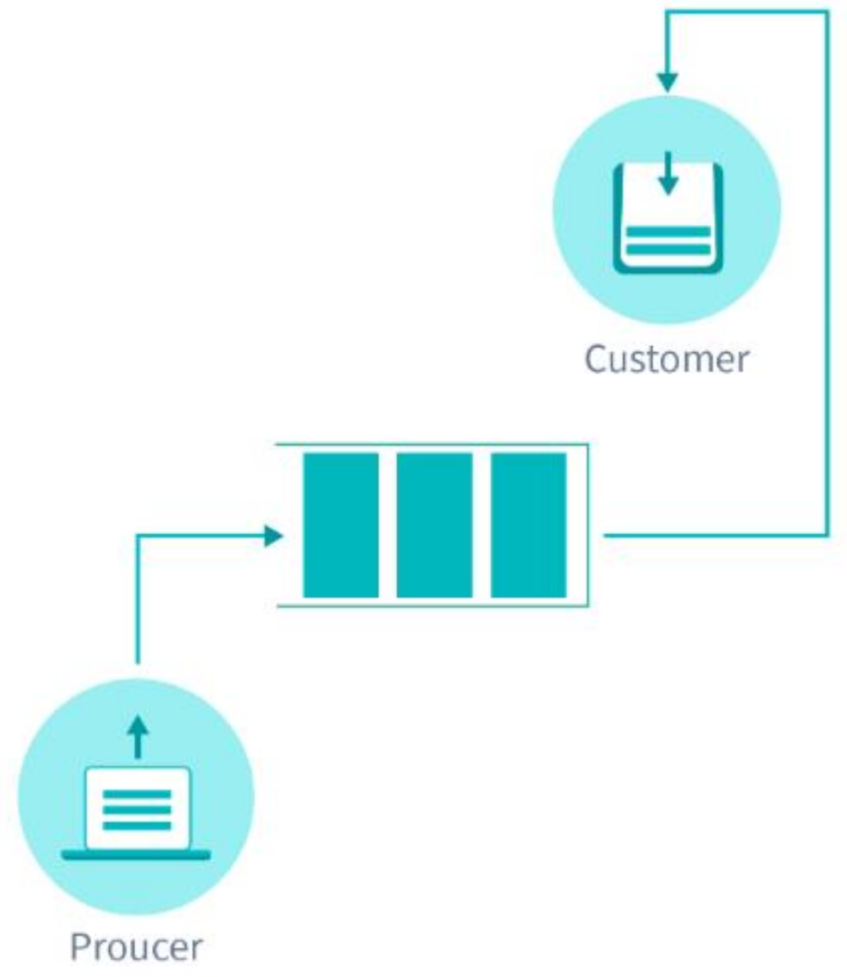
THEORY:-

Producer-Consumer problem is a classical synchronisation problem in the operating system. With the presence of more than one process and limited resources in the system the synchronisation problem arises. If one resource is shared between more than one process at the same time then it can lead to data inconsistency. In the producer-consumer problem, the producer produces an item and the consumer consumes the item produced by the producer.

What is the Producer Consumer Problem?

Before knowing what is Producer-Consumer Problem we have to know what are Producer and Consumer.

- In operating System Producer is a process which is able to produce data/item.
- Consumer is a Process that is able to consume the data/item produced by the Producer.
- Both Producer and Consumer share a common memory buffer. This buffer is a space of a certain size in the memory of the system which is used for storage. The producer produces the data into the buffer and the consumer consumes the data from the buffer.



So, what are the Producer-Consumer Problems?

1. Producer Process should not produce any data when the shared buffer is full.
2. Consumer Process should not consume any data when the shared buffer is empty.
3. The access to the shared buffer should be mutually exclusive i.e at a time only one process should be able to access the shared buffer and make changes to it.

For consistent data synchronisation between Producer and Consumer, the above problem should be resolved.

Solution For Producer Consumer Problem

To solve the Producer-Consumer problem three semaphores variable are used :

Semaphores are variables used to indicate the number of resources available in the system at a particular time. semaphore variables are used to achieve `Process Synchronisation.

Full

The full variable is used to track the space filled in the buffer by the Producer process. It is initialised to 0 initially as initially no space is filled by the Producer process.

Empty

The Empty variable is used to track the empty space in the buffer. The Empty variable is initially initialised to the BUFFER-SIZE as initially, the whole buffer is empty.

Mutex

Mutex is used to achieve mutual exclusion. mutex ensures that at any particular time only the producer or the consumer is accessing the buffer.

Mutex - mutex is a binary semaphore variable that has a value of 0 or 1.

We will use the Signal() and wait() operation in the above-mentioned semaphores to arrive at a solution to the Producer-Consumer problem.

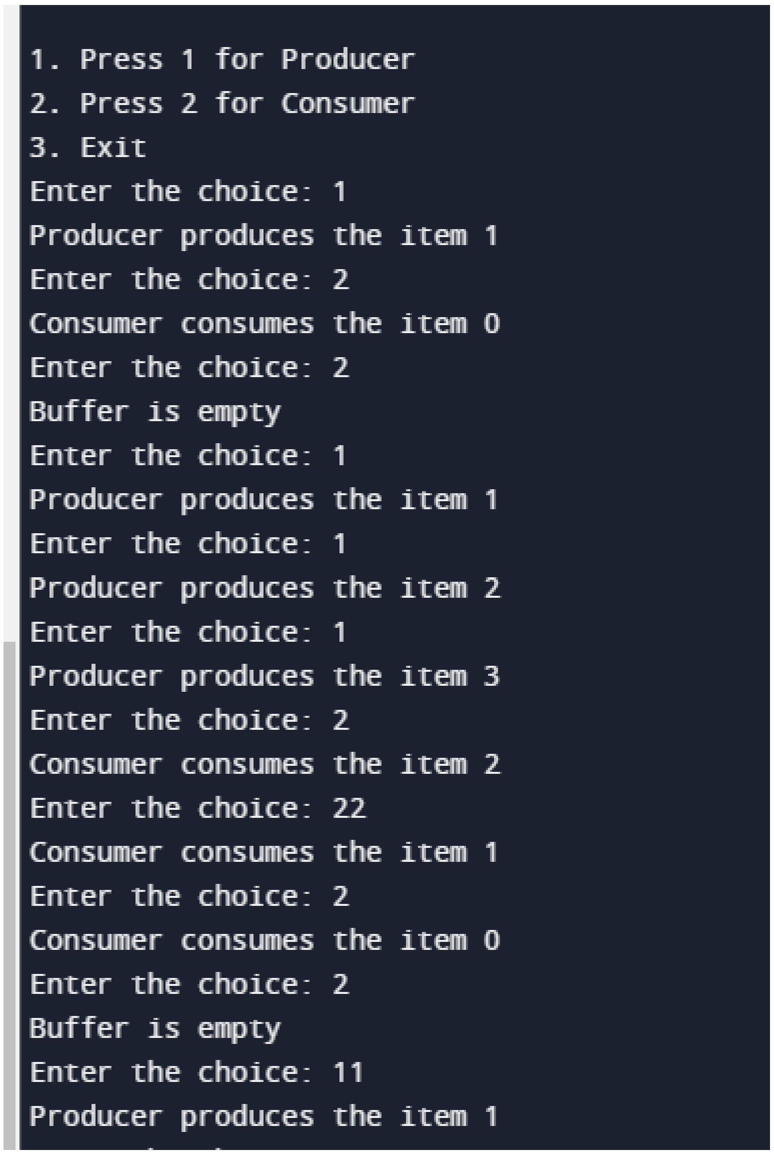
Signal() - The signal function increases the semaphore value by 1. Wait() - The wait operation decreases the semaphore value by 1.

CODE:

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

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

void Producer()
{
    --mutex;
    ++full;
    --empty;
    x++;
    printf("Producer produces the item %d", x);
    ++mutex;
}

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

int main()
{

    int n, i;

    printf("\n1. Press 1 for Producer \n2. Press 2 for Consumer \n3. Exit");
    #pragma omp critical

    for (i = 1; i > 0; i++)
    {
        printf("\nEnter the choice: ");
        scanf("%d", &n);
        switch (n)
        {
            case 1:
                if ((mutex == 1) && (empty != 0))
                    Producer(full, empty, x, mutex);
                else
                    printf("Buffer is full");
                break;

            case 2:
```

```
            if ((mutex == 1) && (full != 0))
                Consumer(full, empty, x, mutex);
            else
                printf("Buffer is empty");
            break;
        case 3:
            printf("Exiting......");
            exit(0);
            break;
        default:
            printf("Invalid choice!!!!");
            break;
    }

  }

}
```

OUTPUT:



```
1. Press 1 for Producer
2. Press 2 for Consumer
3. Exit
Enter the choice: 1
Producer produces the item 1
Enter the choice: 2
Consumer consumes the item 0
Enter the choice: 2
Buffer is empty
Enter the choice: 1
Producer produces the item 1
Enter the choice: 1
Producer produces the item 2
Enter the choice: 1
Producer produces the item 3
Enter the choice: 2
Consumer consumes the item 2
Enter the choice: 22
Consumer consumes the item 1
Enter the choice: 2
Consumer consumes the item 0
Enter the choice: 2
Buffer is empty
Enter the choice: 11
Producer produces the item 1
```

CONCLUSION:

In conclusion, employing semaphores in C to tackle the Producer-Consumer problem offers an elegant solution for managing shared resources and coordinating concurrent execution between producers and consumers. By using semaphores to enforce synchronisation and mutual exclusion, this approach ensures that producers and consumers operate safely and efficiently, preventing issues such as race conditions or data corruption. Through careful design and implementation, the Producer-Consumer problem can be effectively resolved in C, facilitating the development of robust and scalable multi-threaded applications.