OS EXPERIMENT 6

Name: Mohammed Arhaan Ali
Div: S11
Roll No: 02

Aim: Write a C program to implement a solution of Producer Consumer problem through semaphore.

Theory:
A semaphore S is an integer variable that can be accessed only through two standard operations : wait() and signal().
The wait() operation reduces the value of semaphore by 1 and the signal() operation increases its value by 1.

```
wait(S){
while(S<=0);   // busy waiting
S--;
}

signal(S){
S++;
}
```

Semaphores are of two types:

1. Binary Semaphore – This is similar to mutex lock but not the same thing. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problem with multiple processes.

2. Counting Semaphore – Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

Problem Statement – We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items

and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section.
To solve this problem, we need two counting semaphores – Full and Empty. “Full” keeps track of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied slots.
Initialization of semaphores –

```
mutex = 1
Full = 0 // Initially, all slots are empty. Thus full slots are 0
Empty = n // All slots are empty initially
```

Solution for Producer –

```
do{

//produce an item

wait(empty);
wait(mutex);

//place in buffer

signal(mutex);
signal(full);

}while(true)
```

When a producer produces an item then the value of “empty” is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumers from accessing the buffer. Now, the producer has placed the item and thus the value of “full” is increased by 1. The value of mutex is also increased by 1 because the task of producer has been completed and consumers can access the buffer.
Solution for Consumer –

```
do{

wait(full);
wait(mutex);

// consume item from buffer
signal(mutex);
signal(empty);


}while(true)
```

As the consumer is removing an item from the buffer, therefore the value of “full” is reduced by 1 and the value of mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the consumer has consumed the item, thus increasing the value of “empty” by 1. The value of mutex is also increased so that producers can access the buffer now.

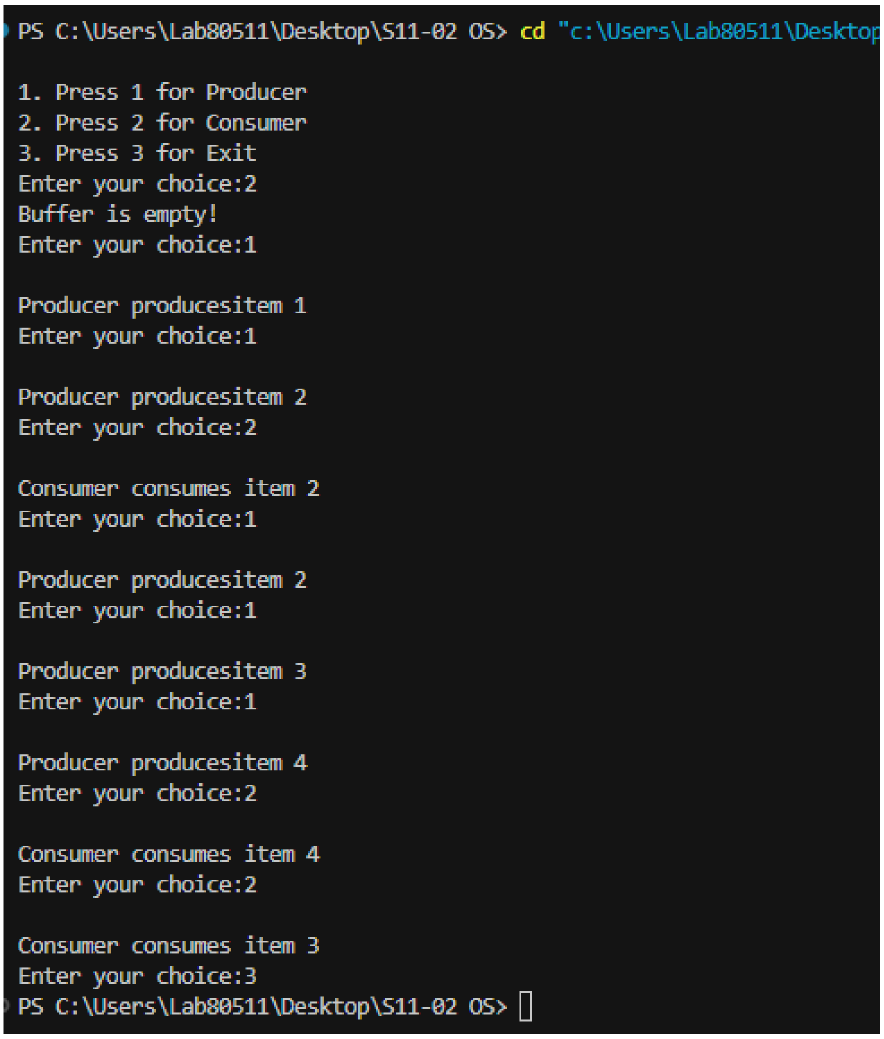
Program:

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


int mutex = 1;

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

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

    --mutex;
    ++full;
    --empty;
    x++;
    printf("\nProducer produces"
           "item %d",
           x);
    ++mutex;
}


void consumer()
{

    --mutex;
    --full;

    ++empty;
    printf("\nConsumer consumes "
           "item %d",
           x);
    x--;

    // Increase mutex value by 1
    ++mutex;
}

// Driver Code
int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");


#pragma omp critical

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

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

        // Switch Cases
```

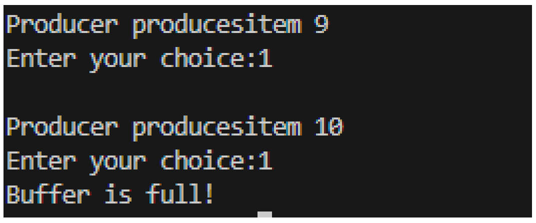
```
switch (n) {
case 1:

    // If mutex is 1 and empty
    // is non-zero, then it is
    // possible to produce
    if ((mutex == 1)
        && (empty != 0)) {
        producer();
    }

    // Otherwise, print buffer
    // is full
    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();
    }

    // Otherwise, print Buffer
    // is empty
    else {
        printf("Buffer is empty!");
    }
    break;
// Exit Condition
case 3:
    exit(0);
    break;
```

```
        }
    }
}
```

Output:



```
PS C:\Users\Lab80511\Desktop\S11-02 OS> cd "c:\Users\Lab80511\Desktop

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 producesitem 1
Enter your choice:1

Producer producesitem 2
Enter your choice:2

Consumer consumes item 2
Enter your choice:1

Producer producesitem 2
Enter your choice:1

Producer producesitem 3
Enter your choice:1

Producer producesitem 4
Enter your choice:2

Consumer consumes item 4
Enter your choice:2

Consumer consumes item 3
Enter your choice:3
PS C:\Users\Lab80511\Desktop\S11-02 OS>
```



```
Producer producesitem 9
Enter your choice:1

Producer producesitem 10
Enter your choice:1
Buffer is full!
```

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