Name: Vishal Shashikant Salvi.
UID: 2019230069
Class: SE Comps
Batch: C

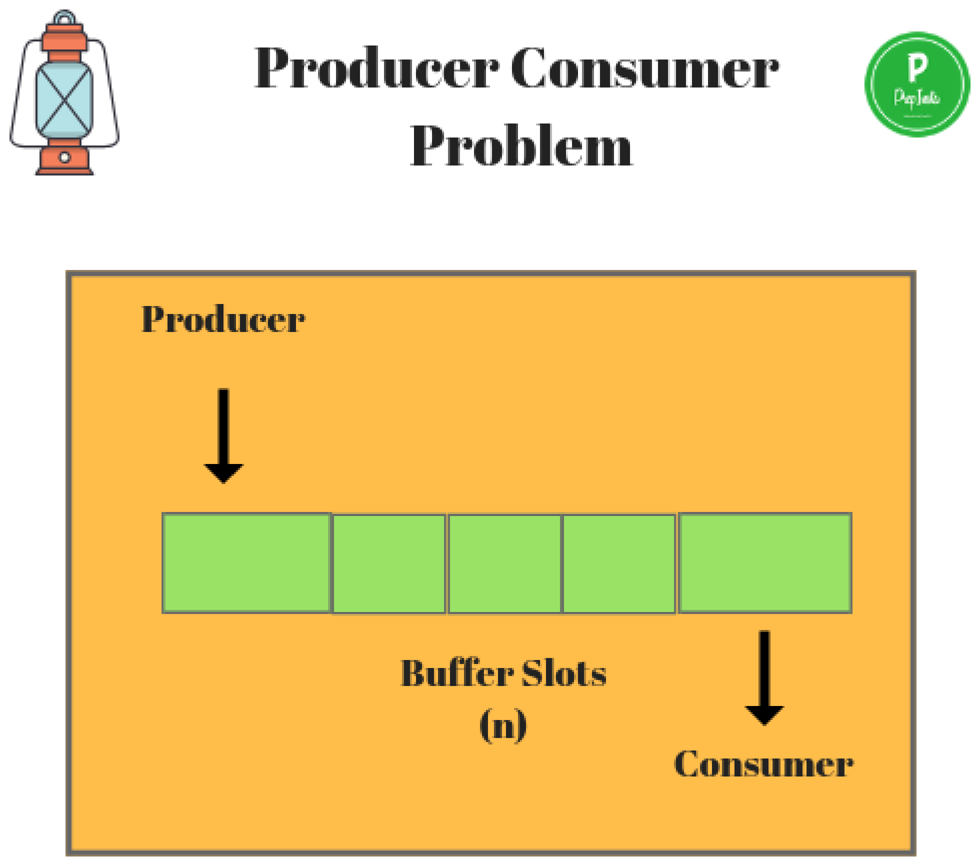
Experiment No 4

Aim: Producer Consumer Problem

Theory:

The producer consumer problem is a synchronization problem. There is a fixed size buffer and the producer produces items and enters them into the buffer. The consumer removes the items from the buffer and consumes them.

A producer should not produce items into the buffer when the consumer is consuming an item from the buffer and vice versa. So the buffer should only be accessed by the producer or consumer at a time.



The producer consumer problem can be resolved using semaphores. The codes for the producer and consumer process are given as follows:

Producer Process

The code that defines the producer process is given below:

```
do {
    .
    . PRODUCE ITEM
    .
    wait(empty);
    wait(mutex);
    .
    . PUT ITEM IN BUFFER
    .
    signal(mutex);
    signal(full);

} while(1);
```

In the above code, mutex, empty and full are semaphores. Here mutex is initialized to 1, empty is initialized to n (maximum size of the buffer) and full is initialized to 0.

The mutex semaphore ensures mutual exclusion. The empty and full semaphores count the number of empty and full spaces in the buffer.

After the item is produced, wait operation is carried out on empty. This indicates that the empty space in the buffer has decreased by 1. Then wait operation is carried out on mutex so that consumer process cannot interfere.

After the item is put in the buffer, signal operation is carried out on mutex and full. The former indicates that consumer process can now act and the latter shows that the buffer is full by 1.

Consumer Process

The code that defines the consumer process is given below:

```
do {


      wait(full);

      wait(mutex);

      .      .

      .  REMOVE ITEM FROM BUFFER

      .

      signal(mutex);

      signal(empty);

      .

      . CONSUME ITEM

      .

} while(1);
```

The wait operation is carried out on full. This indicates that items in the buffer have decreased by 1. Then wait operation is carried out on mutex so that producer process cannot interfere.

Then the item is removed from buffer. After that, signal operation is carried out on mutex and empty. The former indicates that consumer process can now act and the latter shows that the empty space in the buffer has increased by 1.

Producer consumer problem is a classical synchronization problem. We can solve this problem by using semaphores.

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.

Semaphores are of two types:

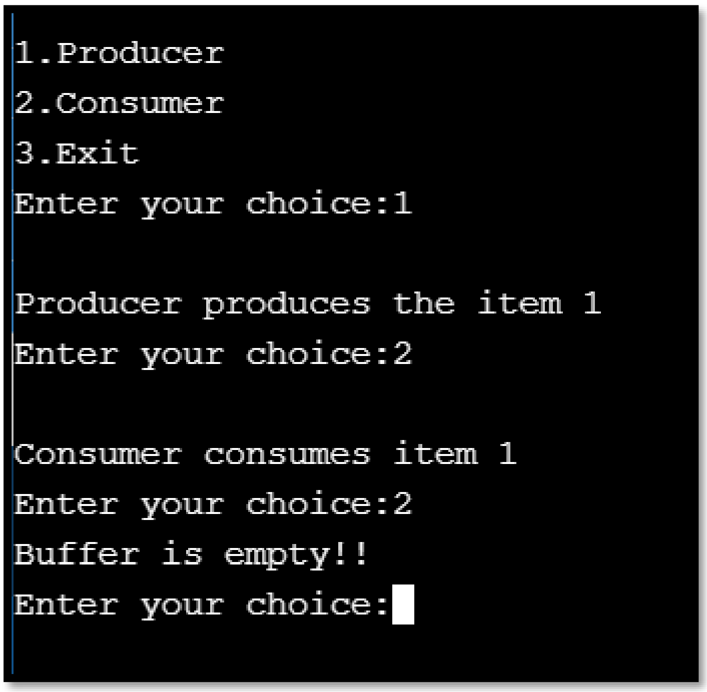
1. **Binary Semaphore –** This is similar to mutex lock but not the same thing. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problem with multiple processes.
2. **Counting Semaphore –** Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances

Program:

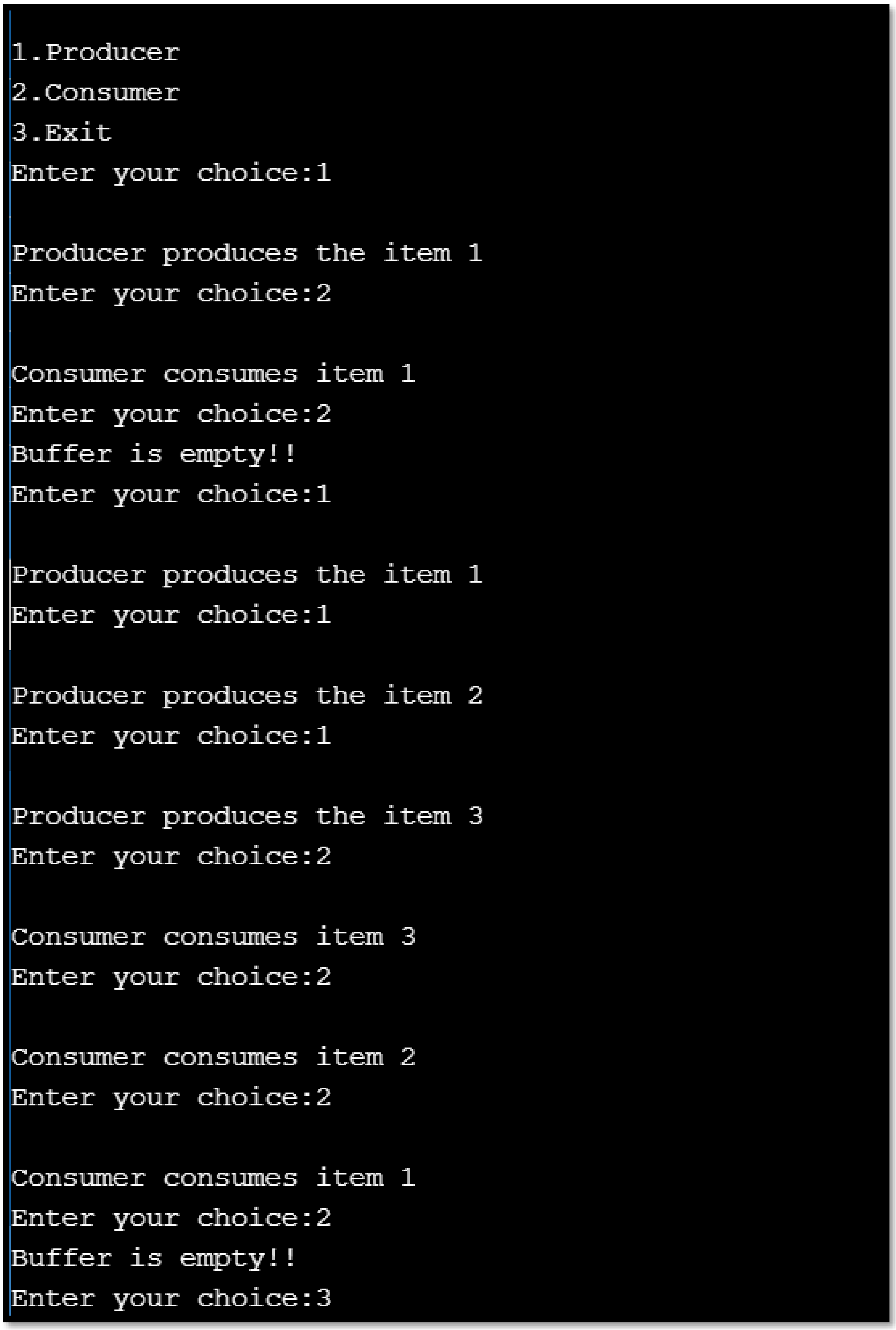
```
#include<stdio.h>
#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()
{
        int n;
        void producer();
        void consumer();
        int wait(int);
        int signal(int);
        printf("\n1.Producer\n2.Consumer\n3.Exit");
        while(1)
        {
                printf("\nEnter your choice:");
                scanf("%d",&n);
                switch(n)
                {
                        case 1:if((mutex==1)&&(empty!=0))
                                                producer();
                                        else
                                                printf("Buffer is full!!");
}
                                        break;
                        case 2:if((mutex==1)&&(full!=0))
                                                consumer();
                                        else
                                          printf("Buffer is empty!!");
                                        break;
                        case 3:
                                        exit(0);
                                        break;
                }
        }

        return 0;
}

int wait(int s)
{
        return (--s);
}

int signal(int s)
{
        return(++s);
}
```

```
void producer()
{
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
        printf("\nProducer produces the item %d",x);
        mutex=signal(mutex);
}

void consumer()
{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\nConsumer consumes item %d",x);
        x--;
        mutex=signal(mutex);
}
```

Output:



```
1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:2

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


```
1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:2

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

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
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 learnt producer consumer problem as well as how it can be implemented. Producers are involved in a never-ending process of introducing new products and services and then observing economic behaviour. By having several products, producers can experiment and watch economic behaviour as consumers will focus on the features and products that are most desirable.