Producer Consumer Problem

Mini project report submitted in partial fulfilment of the requirements of the degree of

**Second year of Engineering**

by

|  |  |
| --- | --- |
| **Name of student** | **Roll number** |
| Tausif Khan | 12 |
| Parth Bhatt | 06 |
| Naveed Merchant | 17 |

Under the guidance of

**Ms.Vidya Kawtikwar**



Department of Computer Engineering

St. John College of Engineering and Management

University of Mumbai

2019-2020

**CERTIFICATE**

This is to certify that the project entitled **“**Producer Consumer Problem **”** is a Mini Project report of

**“Tausif Khan” (12)**

**“Parth Bhatt” (06)**

**“Naveed Merchant” (17)**

submitted in partial fulfilment of **“Second year of Engineering”** in **“Computer Engineering”** as laid down by University of Mumbai during the academic year 2019-2020.

Ms.Vidya Kawtikwar

**Guide**

Dr. Rahul Khokale Dr. G.V. Mulgund

**Head of Department** **Principal**

**CONTENT**

|  |  |  |
| --- | --- | --- |
| **Sr No** | **Content** | **Page No** |
| 1 | Introduction | 1-4 |
| 2 | Implementation | 4-6 |
| 3 | Results | 6-8 |
| 4 | Conclusion | 8-9 |
| 5 | References | 9-10 |

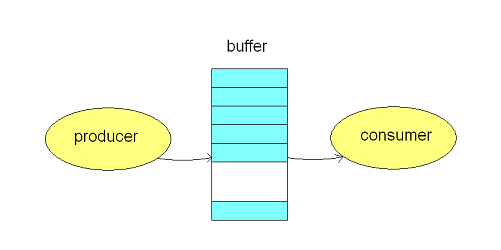
**Introduction:-**

**Producer Consumer Problem**

The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between more than one processes.

In the producer-consumer problem, there is one Producer that is producing something and there is one Consumer that is consuming the products produced by the Producer. The producers and consumers share the same memory buffer that is of fixed-size.

The job of the Producer is to generate the data, put it into the buffer, and again start generating data. While the job of the Consumer is to consume the data from the buffer.



#### **What's the problem here?**

The following are the problems that might occur in the Producer-Consumer:

* The producer should produce data only when the buffer is not full. If the buffer is full, then the producer shouldn't be allowed to put any data into the buffer.
* The consumer should consume data only when the buffer is not empty. If the buffer is empty, then the consumer shouldn't be allowed to take any data from the buffer.
* The producer and consumer should not access the buffer at the same time.

#### **What's the solution?**

The above three problems can be solved with the help of semaphores.

In the producer-consumer problem, we use three semaphore variables:

1. **Semaphore S:** This semaphore variable is used to achieve mutual exclusion between processes. By using this variable, either Producer or Consumer will be allowed to use or access the shared buffer at a particular time. This variable is set to 1 initially.
2. **Semaphore E:** This semaphore variable is used to define the empty space in the buffer. Initially, it is set to the whole space of the buffer i.e. "n" because the buffer is initially empty.
3. **Semaphore F:** This semaphore variable is used to define the space that is filled by the producer. Initially, it is set to "0" because there is no space filled by the producer initially.

## By using the above three semaphore variables and by using the *wait()* and *signal()* function, we can solve our problem(the *wait()* function decreases the semaphore variable by 1 and the *signal()* function increases the semaphore variable by 1). So. let's see how.

The following is the pseudo-code for the producer:

**void** **producer**() {

**while**(T) {

produce()

wait(E)

wait(S)

append()

signal(S)

signal(F)

}

}

**The above code can be summarized as:**

* *while()* is used to produce data, again and again, if it wishes to produce, again and again.
* *produce()* function is called to produce data by the producer.
* *wait(E)* will reduce the value of the semaphore variable "E" by one i.e. when the producer produces something then there is a decrease in the value of the empty space in the buffer. If the buffer is full i.e. the vale of the semaphore variable "E" is "0", then the program will stop its execution and no production will be done.
* *wait(S)* is used to set the semaphore variable "S" to "0" so that no other process can enter into the critical section.
* *append()* function is used to append the newly produced data in the buffer.
* *signal(s)* is used to set the semaphore variable "S" to "1" so that other processes can come into the critical section now because the production is done and the append operation is also done.
* *signal(F)* is used to increase the semaphore variable "F" by one because after adding the data into the buffer, one space is filled in the buffer and the variable "F" must be updated.

This is how we solve the produce part of the producer-consumer problem. Now, let's see the consumer solution. The following is the code for the consumer:

**void** **consumer**() {

**while**(T) {

wait(F)

wait(S)

take()

signal(S)

signal(E)

use()

}

}

**The above code can be summarized as:**

* *wait(F)* is used to decrease the semaphore variable "F" by one because if some data is consumed by the consumer then the variable "F" must be decreased by one.
* *wait(S)* is used to set the semaphore variable "S" to "0" so that no other process can enter into the critical section.
* *take()* function is used to take the data from the buffer by the consumer.
* *signal(S)* is used to set the semaphore variable "S" to "1" so that other processes can come into the critical section now because the consumption is done and the take operation is also done.
* *signal(E)* is used to increase the semaphore variable "E" by one because after taking the data from the buffer, one space is freed from the buffer and the variable "E" must be increased.
* use() is a function that is used to use the data taken from the buffer by the process to do some operation.

So, this is how we can solve the producer-consumer problem.

## **Implementation:-**

## **Producer Consumer Problem in C++**

Below is the program to implement this problem.

#include<iostream>

using namespace std;

void wait(int &x)

{

if(x>0)

x--;

}

void signal(int &x)

{

x++;

}

int main()

{

int a[5];

int empty = 5;

int full=0;

int mutex=1;

while(1)

{

cout<<"....................................................."<<endl;

cout<<" Empty "<<empty<<" Full "<<full<<" Mutex "<<mutex<<endl;

cout<<"Enter your choice 1. produce 2. consume"<<endl;

int x;

cin>>x;

if(x==1)

{

if(empty!=0&&mutex==1)

{

cout<<"Enter the data"<<endl;

wait(empty);

wait(mutex);

cin>>a[full];

signal(mutex);

signal(full);

}

else cout<<"Buffer is full"<<endl;

}

else if(x==2)

{

if(full>0&&mutex==1)

{

wait(full);

wait(mutex);

cout<<"Data is "<<a[full]<<endl;

signal(mutex);

signal(empty);

}

else cout<<"Buffer is empty"<<endl;

}

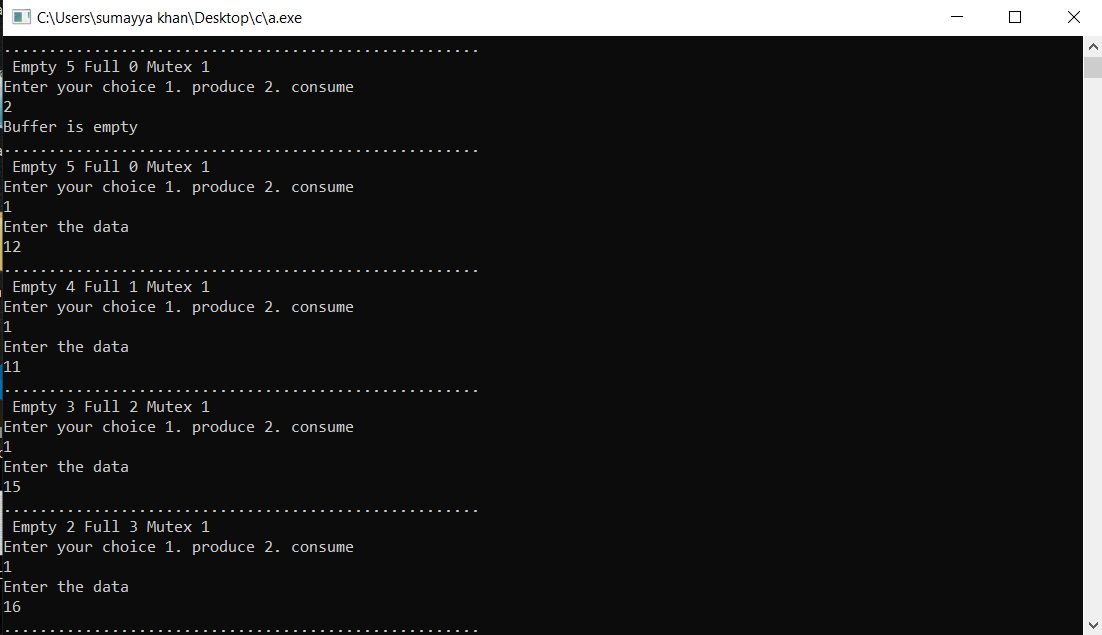
else break;

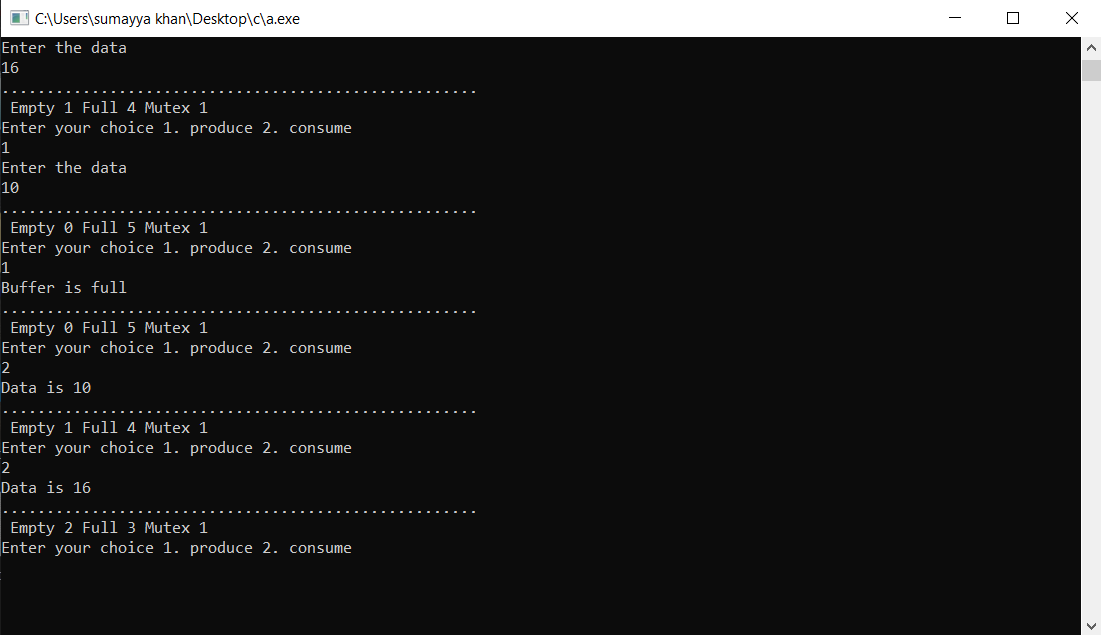
}

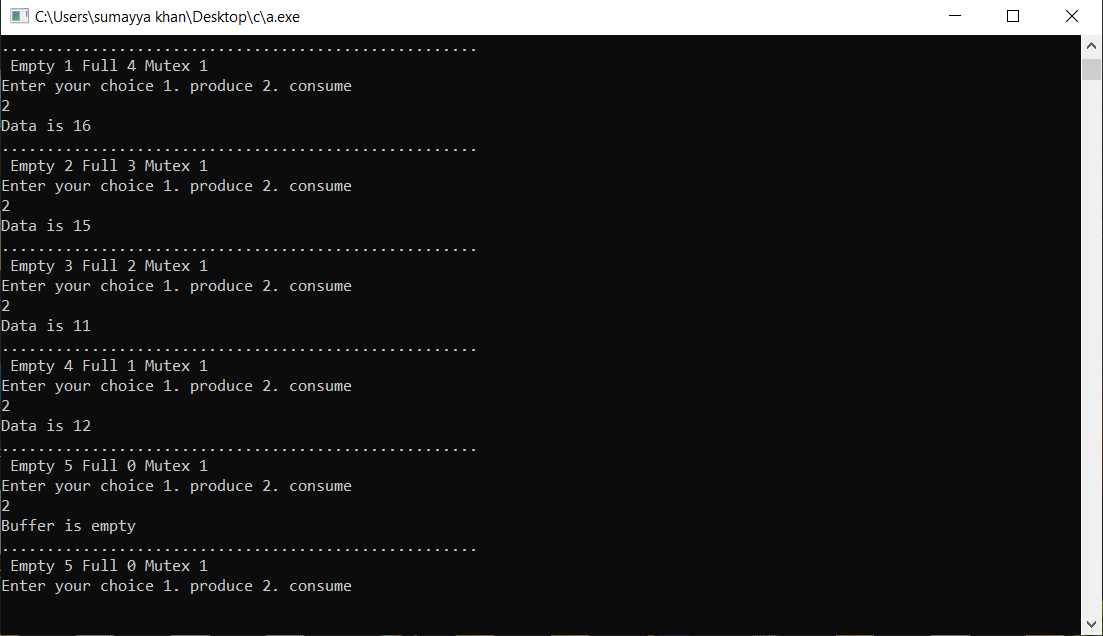
return 0;

}

**Results:-**







**Conclusion:-**

Thus in this project we have studied and implemented Producer Consumer problem. Producer Consumer problem uses in multi-processing system. This has been deemed appropriate since most of the current applications are distributed in nature and run in multi-processing environment. This problem satisfy necessary requirements to provide solution to the critical section problem.

**References:-**

1. https://afteracademy.com/blog/the-producer-consumer-problem-in-operating-system

2. https://www.thecrazyprogrammer.com/2016/09/producer-consumer-problem-c.html