**PRODUCER-CONSUMER USING SEMAPHORES**

**EXPT NO: 4 DATE:28/10/22**

**AIM**

To implement bounded buffer producer-consumer problem

**THEORY**

The bounded-buffer problems (aka the producer-consumer problem) is a classic example of concurrent access to a shared resource. This issue is a Classic Synchronisation Problem.

This problem focuses primarily on two different tasks: Producer and Consumer**.** Both of them share a fixed size and a common buffer.

* The producer creates data and puts it into the buffer and restarts it.
* The consumer consumes the data. In other words, the consumer removes the data from the buffer that the producer has created.

The producer and consumer problem are to ensure that the producer should not create data into the buffer memory once it gets full and simultaneously, the consumer should not remove data from a buffer memory that is empty.

The Producer-Consumer problem can be resolved by placing a semaphore in the buffer.

A [semaphore](https://www.geeksforgeeks.org/semaphores-in-process-synchronization/) 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. 

In producer-consumer problem 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

Empty=n

do{

//Consume an item

Wait(full)

Wait(mutex)

//Remove item from buffer

Signal(mutex)

Signal(empty)

}while(true);

do{

//Produce an item

Wait(empty)

Wait(mutex)

//Place in buffer

Signal(mutex)

Signal(full)

}while(true);

**SOURCE CODE**

#include<iostream>

#include<queue>

using namespace std;

#define MAX\_SIZE 10

#define WAIT s--

#define SIGNAL s++

#define FULL EMPTY\_SLOTS==0

#define SLOT\_OCCUPIED EMPTY\_SLOTS--

int s=0;

int EMPTY\_SLOTS=MAX\_SIZE;

queue<int> BUFFER;

void DISPLAY()

{

queue<int> DISPLAY\_BUFF=BUFFER;

cout<<"CURRENT BUFFER CONTENT: ";

if(DISPLAY\_BUFF.empty()==true)

{

cout<<"QUEUE IS EMPTY"<<endl;

return;

}

while(DISPLAY\_BUFF.empty()==false)

{

cout<<DISPLAY\_BUFF.front()<<" ";

DISPLAY\_BUFF.pop();

}

}

void PRODUCE()

{

int r=rand()%100;

if(FULL)

{

cout<<"\n\nQUEUE IS FULL";

return;

}

WAIT; cout<<"\n\nSEMAPHORE WAIT: "<<s<<endl;

BUFFER.push(r);

SLOT\_OCCUPIED;

SIGNAL; cout<<"SEMAPHORE SIGNAL: "<<s<<endl;

DISPLAY();

}

void CONSUME()

{

if(BUFFER.empty()==true)

{

cout<<"\n\nQUEUE IS EMPTY"<<endl;

return;

}

WAIT; cout<<"\n\nSEMAPHORE WAIT: "<<s<<endl;

cout<<"PROCESS CONSUMED: "<<BUFFER.front()<<endl;

BUFFER.pop();

SIGNAL; cout<<"SEMAPHORE SIGNAL: "<<s<<endl;

DISPLAY();

}

int main()

{

int ch;

do

{

cout<<"\n\n1: PRODUCE\n2: CONSUME\n0: EXIT\n";

cout<<"ENTER YOUR CHOICE ";

cin>>ch;

switch(ch)

{

case 1: PRODUCE();

break;

case 2: CONSUME();

break;

case 0: break;

default : cout<<"INVALID CHOICE ";

break;

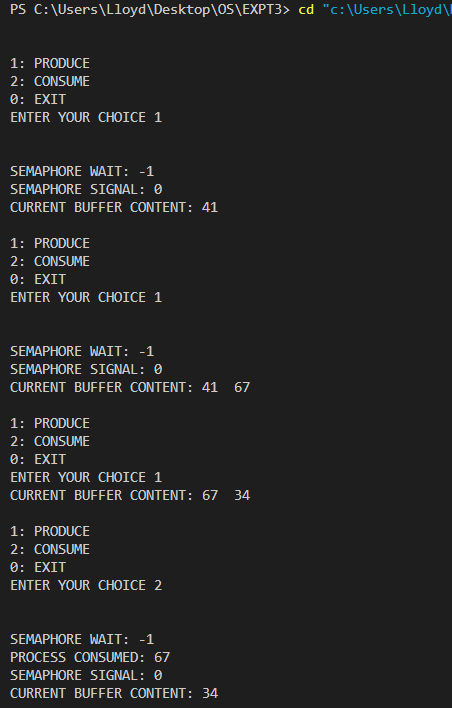
}

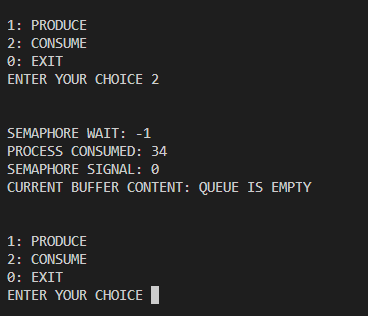
}while(ch);

return 0;

}

**OUTPUT**

****

****

**CONCLUSION**

The Producer-Consumer Problem using semaphores was comprehended and was successfully implemented.