**AIM:** Write a c programme to implement solution of producer consumer problem through semaphore.

**THEORY:**

* Semaphores in operating system, Inter Process Communication. 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.

Syntax:

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.

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

* To solve the Producer-Consumer problem three semaphores variable are used:
* **Full**
* The full variable is used to track the space filled in the buffer by the Producer process. It is initialized 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 initialized 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.

**CONCLUSION:**

* Producer Process produces data item and consumer process consumes data item.
* Both producer and consumer processes share a common memory buffer.
* Producer should not produce any item if the buffer is full.
* Consumer should not consume any item if the buffer is empty.
* Not more than one process should access the buffer at a time i.e mutual exclusion should be there.
* Full, Empty and mutex semaphore help to solve Producer-consumer problem.
* Full semaphore checks for the number of filled space in the buffer by the producer process
* Empty semaphore checks for the number of empty spaces in the buffer.
* mutex checks for the mutual exclusion.

**PROGRAM**

#include <stdio.h>

#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10, x = 0;

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--;

++mutex;

}

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 (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;

}

}

}

