A project report on

Producer & Consumer Problem

Submitted by
Sarbesh Prasad Gupta
Section- K1608
Roll No- B44
RegNo-11614206
Email-sarveshg821@gmail.com
GithubLink-
https://github.com/guptasarbesh/OS_PROJECT

Submitted to,
Harshpreet singh

Table Contents

Producer-Consumer Problem

Producer-consumer problem (also known as the bounded-buffer problem) is a multi-process synchronization problem.
The problem describes two processes, the producer and the consumer, who share a common fixed-size buffer.

Producer: The producer's job is to generate a piece of data, put it into the buffer and start again.

Consumer: The consumer is consuming the data(i.e. removing it from the buffer) one piece at a time.

Problem:

The problem is to make sure that the product won't try to add data into the buffer if if it's full and that the consumer won't try to remove data from an empty buffer.

Solution For The Producer

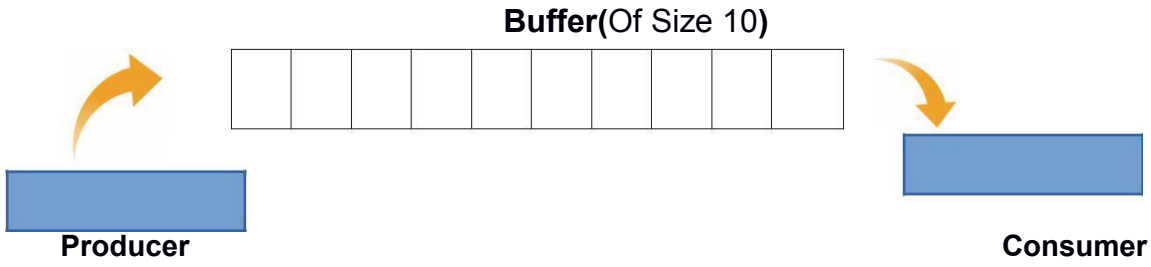
Producer either go to sleep or discard data if the buffer is full.

The next time the consumer removes an item from the buffer and it notifies the producer who starts to fill the buffer again

Solution For The Consumer

Consumer can go to sleep if it finds the buffer to be empty.

The next time the producer puts data into the buffer, it wakes up the sleeping consumer.



Algorithm

Data Structure :-

Buffer: Buffer is taken of size 10.A buffer is a region of memory used to temporarily hold data while it is being moved from producer to consumer or consumer to producer.

Mutex: A mutex provides mutual exclusion, either producer or consumer can have the key (mutex) and proceed with their work. As long as the buffer is filled by producer, the consumer needs to wait, and vice versa.

Full: Full is used for counting the number of slots that are full.

Empty: Empty is used for counting the number of slots that are empty.

Algorithm :-

For Producer:

1.Let us assume buffer is of size of n and counter is initialized to 0.

2.while(True)

3. item=produce_item()

Producer produces an item.

4.if(count==N)

The buffer is full i.e. sleep(empty).

6.lock(mutex)

7.Insert an item, i.e. insert_item(item)

8.Increment counter

9.unlock(mutex)

10.if(counter==1)

Producer wakes-up and starts producing an item. wakeup(full)

For Consumer:

1.while(True)

2.if(count==0)

The buffer is empty, i.e. sleep(full).

3.lock(mutex)

4.item=remove_item()

It removes item from the buffer.

5.Decrement the counter.

6.unlock(mutex)

7.if(count==N-1)

The consumer removes one item from the buffer, i.e.wakeup(empty)

8.consume_item

Boundary Conditions

The various Boundary condition, for Producer-Consmer problem, are as described.,

Producer produces,but the buffer is full:

When the buffer is full the producer has to wait until the consumer consumes the data from the buffer.The producer checks if the buffer is full . If it is full then it sleep in that case.On the other hand if the consumer consumes one data then producer wakes up and now the producer can insert an item into the buffer.

Consumer consumes,but buffer is empty:

When the buffer is empty consumer has to wait until the fills data into the buffer.If the consumer finds that buffer is empty then it is going to sleep on the particular mutex called full and it will block on these mutex until it gets wakeup from the producer.When the producers insert the item then it pass the wakeup signal to the consumer.Now the consumer wakes up and consumes the data.

The no of producer should be more than the consumer.

Test Cases

Input: First line consists of integer input ‘n’ which tells how much producer has to produce.Second line consists of integer input ’n’ which tells how much consumer has to consume.
Output: It consists of ‘n’ line output telling how much data producer is producing and how much data is consumed by consumer.

Explanation:

Here, the producer is inserting an item in the buffer of size’n’. It inserts an item till there is space in a buffer. If the buffer is full then it goes to sleep condition. Similary,Consumer consumes from the buffer if the item is present in the buffer. It can consumes only if there is some data present in the buffer.If the buffer is empty, then it waits until the producer inserts an item into the buffer. The process go on till it terminates.

Test-Case 1:

Input:

Enter the number of producers:3

Enter the number of consumers:4

Output:

Producer has produced item:3

Producer has produced item:7

Producer has produced item:3

Consumer has consumed item:3

Consumer has consumed item:7

Consumer has consumed item:3

Test-Case 2:

Input:

Enter the number of producers:6

Enter the number of consumers:5

Output:

Producer has produced item:3

Producer has produced item:7

Producer has produced item:3

Producer has produced item:6

Producer has produced item:9

Producer has produced item:2

Consumer has consumed item:2

Consumer has consumed item:9

Consumer has consumed item:6

Consumer has consumed item:3

Consumer has consumed item:7

Complexity

For n as a producer and m as a consumer, n producer produces and m consumer consumes, Hence complexity is an order of O(m+n).

Solution Code

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<semaphore.h>
#include<pthread.h>
#include<stdlib.h>
#define buffersize 10
pthread_mutex_t mutex;
pthread_t p1[20],p2[20];
sem_t full,empty;
int counter;
int buffer[buffersize];
void done()
{
        pthread_mutex_init(&mutex,NULL);
        sem_init(&full,1,0);
        sem_init(&empty,1,buffersize);
        counter=0;
}
void write1(int item)
{
        buffer[counter++]=item;
}
int read1()
{
        return(buffer[--counter]);
}
void *producer(void *param)
{
        int waittime,item,i;
        item=rand()%10;
        waittime=rand()%10;
        sem_wait(&empty);
```

```
        pthread_mutex_lock(&mutex);
        printf("\n Producer has produced item:%d\n",item);
        write1(item);
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
}
void *consumer(void *param)
{
        int waittime,item;
        item=rand()%10;
        waittime=rand()%10;
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        item=read1();
        printf("\n Consumer has consumed item:%d\n",item);
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
}
int main()
{
        int a,b,i;
        done();
        printf("Enter the number of producers:");
        scanf("%d",&a);
        printf("Enter the number of consumers:");
        scanf("%d",&b);
        for(i=0;i<a;i++)
                {
                        pthread_create(&p1[i],NULL,producer,NULL);
                }
        for(i=0;i<b;i++)
                {
                        pthread_create(&p2[i],NULL,consumer,NULL);
                }
        for(i=0;i<a;i++)
                {
                        pthread_join(p1[i],NULL);
                }
        for(i=0;i<b;i++)
                {
                        pthread_join(p2[i],NULL);
```

```
                }
                exit(0);
}
```