OS PROJECT

**Name :-** Vijay Singh Kholiya

**Reg. No. :-** 11603123

**Section :-** K1607

**Email ID :-** <kholiyavijay123@gmail.com>

**Github Link :-** [**https://github.com/vjkholiya123/Producer\_Consumer**](https://github.com/vjkholiya123/Producer_Consumer)

**Q. PRODUCER – CONSUMER PROBLEM**

**CODE:**

# include <unistd.h>

# include <stdio.h>

# include <semaphore.h>

# include <pthread.h>

# include <stdlib.h>

int item;

int buffer\_size;

int buffer[20];

sem\_t empty\_sema;

sem\_t full\_sema;

pthread\_mutex\_t mutex\_lock;

void \*thread\_producer();

void \*thread\_consumer();

int main()

{

x:

printf("\nEnter the size of the buffer (max 20): ");

scanf("%d",&buffer\_size);

if(buffer\_size<=20)

{

sem\_init(&empty\_sema,2,buffer\_size);

sem\_init(&full\_sema,2,0);

int i;

for(i=0;i<buffer\_size;i++)

{

buffer[i]=0;

}

pthread\_mutex\_init(&mutex\_lock,NULL);

pthread\_t thread1[buffer\_size];

pthread\_t thread2[buffer\_size];

int m=0;

for(m=0;m<buffer\_size;m++)

{

pthread\_create(&thread1[m],NULL,thread\_producer,NULL);

pthread\_create(&thread2[m],NULL,thread\_consumer,NULL);

}

for(m=0;m<buffer\_size;m++)

{

pthread\_join(thread1[m],NULL);

pthread\_join(thread2[m],NULL);

}

}

else

{

printf("\nBuffer size exceeded the limit .\nPlease enter the buffer size again and <=20.\n");

goto x;

}

}

void \*thread\_producer()

{

sem\_wait ( &empty\_sema );

pthread\_mutex\_lock ( &mutex\_lock );

sleep(1);

int i;

for(i=0;buffer[i]!=0;i++)

{

}

buffer[i] = rand()%99;

printf("\nPRODUCER is producing the data which is :- %d .\n",buffer[i]);

pthread\_mutex\_unlock ( &mutex\_lock );

sem\_post ( &full\_sema );

}

void \*thread\_consumer()

{

sem\_wait ( &full\_sema );

pthread\_mutex\_lock ( &mutex\_lock );

sleep(1);

int i;

for(i=15;buffer[i]==0;i--)

{

}

printf("\n---Consumer is consuming the data from the buffer that is :- %d .\n",buffer[i]);

buffer[i]=0;

pthread\_mutex\_unlock ( &mutex\_lock );

sem\_post ( &empty\_sema );

}

**Algorithm :-**

sem\_t empty\_sema;

sem\_t full\_sema;

pthread\_mutex\_t mutex1;

thread producer

{

sem\_wait(&empty\_sema);

pthread\_mutex\_lock(&mutex1);

// Producer performs its task

pthread\_mutex\_unlock(&mutex1);

sem\_post(&full\_sema);

}

thread consumer

{

sem\_wait (&full\_sema);

pthread\_mutex\_lock(&mutex1);

// Consumer performing its task

pthread\_mutex\_unlock(&mutex1);

sem\_post(&empty\_sema);

}

**Complexity :-**

**Average case complexity is:**

O(buffer\_size)

Buffer\_size is taken as input from the user and it is used to access the buffer array.

**Worst case complexity is :**

O(20)

It is the maximum capacity of the buffer array and above 20 the program prompts for re-entering the buffer size;

**Description :-**

The producer consumer problem or the bounded buffer problem is a problem of multi-process synchronization. According to this problem we have a total of two threads namely producer and consumer and these threads share a common fixed sized buffer.

Both the threads performs different actions to the buffer. Producer produces the items and fills the buffer and consumer utilizes/consumes the item and empties the buffer.

Producer cannot produce if the buffer is full and consumer cannot consume if the buffer is empty.

There are some constraints when the producer is producing:

1. No other producer can produce the item.
2. No consumer can consume the item at that time.

When consumer consumes the constraints are:

1. No other consumer can consume the item.
2. No producer should produce the item

We can achieve these constraints with the help of semaphores or mutex locks or monitors. When the producer thread is executing it firstly decrements the empty semaphore that is a counting semaphore initialized with the size of the buffer and it holds a mutex lock on the shared variable . After the execution of the producing task it then unlocks the mutex lock and increments the full semaphore which is also a counting semaphore initialized to 0.

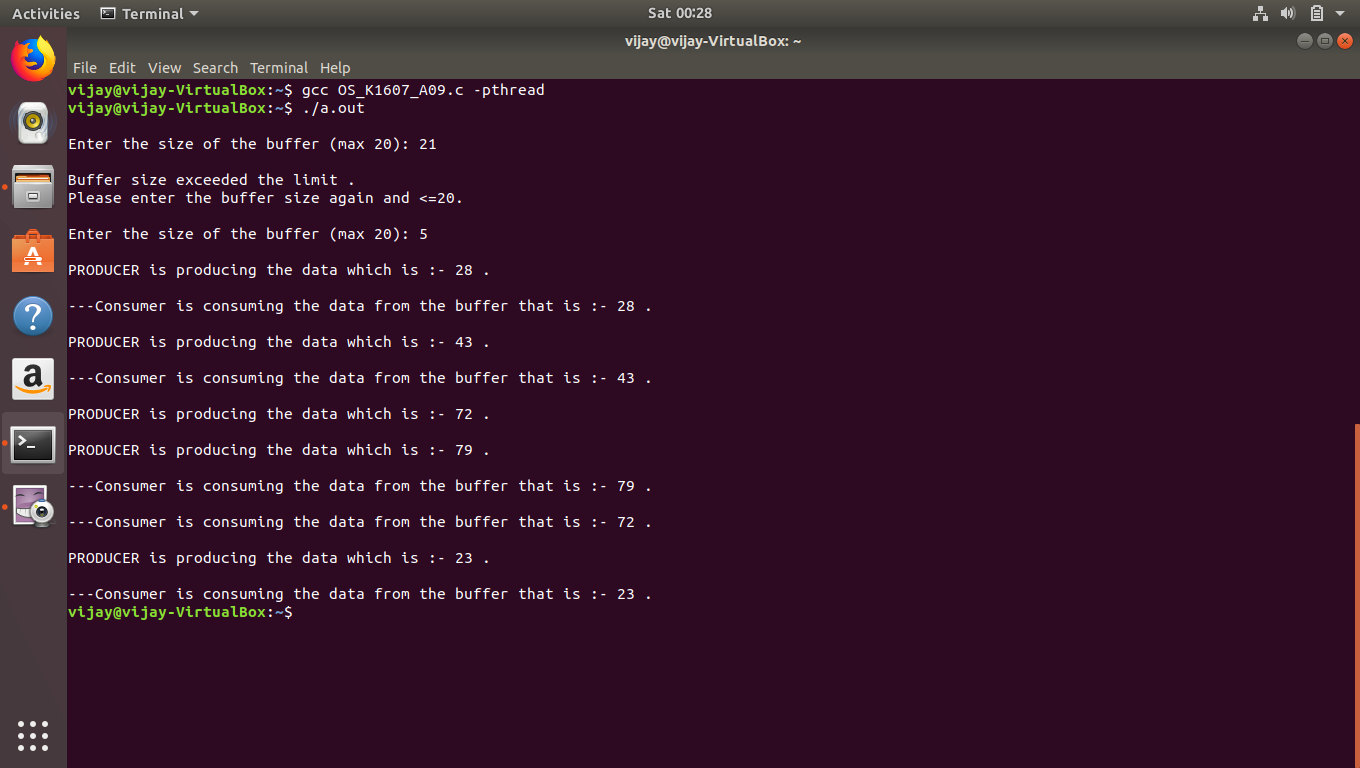
Similarly when the consumer thread runs it firstly decrements full semaphore and holds mutex lock on the shared variable and performs its task of consuming the item. Then it unlocks the mutex lock and increments the empty semaphore.

**Boundary Conditions :-**

1. Not feasible for large buffer size as execution time will be increased due to sleep() system call.
2. It is not fixed pre-defined that if any number of producer can come any time whether continuously or alternately as consumer arrives.
3. Producer thread will always run first then the consumer thread will execute due to process synchronization and mutex locks.

**Test Cases :-**

1. If the user enter the size of the buffer which exceeds the maximum capacity of the buffer array then program prompts the user to re-enter the buffer size. The maximum capacity of the buffer is 20 so user has to enter less than or equal to its maximum capacity.
2. If the user enters buffer size less than 20 then normal synchronization of both the producer and consumer threads will be done. Producer will firstly put the item into the buffer and then consumer is going to remove the item from the buffer. Rand() function is used to generate random items produced by the producer and consumer is going to consume item in a LIFO manner from the buffer.



1. When the buffer size is 13 then the code outputs the producer producing items like 28,43,72,etc. and consumer consumes items from the buffer. If there is no item in the buffer and consumer process wants to execute but it will not be executed due to semaphores and the producer is going to be executed and will be producing the item and put into the buffer then the consumer can consume the item from the buffer.

