**Student Name:** Ritesh Singh

**Student ID:** 11602212

**Email Address:** [gettoriteshsingh@gmail.com](mailto:gettoriteshsingh@gmail.com)

**GitHub Link:**  <https://github.com/ritesingh/K-1607-11602212-Producer_Consumer>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CODE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

//A menu driven program for bounded buffer problem, 11602212-Ritesh Singh.

#include<semaphore.h>

#include<pthread.h>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#define len 10

sem\_t full,empty; //standard semaphore variables

pthread\_t prod[10],cons[10];

pthread\_mutex\_t mutex; //mutex lock

int buff[len];

int count=0;

void \*consumers(void \*y) //consumer thread begins

{

int val;

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

val=buff[--count];

buff[count+1]=0;

printf("Consumed item::%d\n",val);

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

pthread\_exit(NULL);

} //consumers thread ends

void \*producers(void \*y) //producers thread begins

{

printf("\n\nEnter the number to be produced and wait 3 sec: ");

int val;

scanf("%d",&val);

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

printf("\nJust produced %d in buffer::",val);

buff[count++]=val;

printf("\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf("\n |");

for (int n=0; n<len; n++)

{

if(buff[n]!=0)

printf(" %d |",buff[n]);

}

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

pthread\_exit(NULL);

} //end of producer thread

int main()

{

int a,b,ch;

sem\_init(&empty,0,5); //intially 5

sem\_init(&full,0,0); //initially 0

while(1)

{

printf("\n\n 1.Produce\n 2.Consume\n 3.Exit\n What to do? ");

scanf("%d",&ch);

switch(ch)

{

case 1: {

printf("\nHow many items would you like to produce: ");

scanf("%d",&a);

for(int i=0;i<a;i++)

{

pthread\_create(&prod[i],NULL,producers,NULL);

sleep(3);

}

for(int i=0;i<a;i++)

{

pthread\_join(prod[i],NULL);

}

}break;

case 2: { printf("\nHow many items would you like to consume: ");

scanf("%d",&b);

for(int i=0;i<b;i++)

{

if(count==0)

{

printf("\nBuffer is empty first Produce Item \n");

break;

}

pthread\_create(&cons[0],NULL,consumers,NULL);

sleep(1);

}

for(int i=0;i<b;i++)

{

pthread\_join(cons[0],NULL); }

}break;

case 3: {

exit(0);

}break;

default: {

printf("NOT VALID!!");

}

}//end of switch

}//end of while

return 0;

}//end of main function

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DESCRIPTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Programming Solution to Bounded-Buffer Problem. I have used standard counting semaphores empty, full for counting the empty and full slots in the buffer. A mutex lock is used rather than a binary semaphore for obtaining mutual exclusion. There are two threads Producer and Consumer where Producer adds an item to the buffer and Consumer obtains an item from the buffer. This programming solution has been done for a Linux platform using thread library and will require the same to compile it successfully. Rather than automating the producer thread to produce data on its own using some random numbers, I have made the code user friendly where the user is asked to enter the data to be produced and than the amount of data to be consumed recursively. The number of calls to the producer and the consumer threads completely depends on the user. Main objective of this project is to achieve process synchronization such that both the consumer and producer threads may work exclusively in harmony.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ALGORITHM\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. Take two standard semaphore variables full, empty and a mutex lock for achieving mutual exclusion.
2. Create two threads producer and consumer with following pseudocodes:

**Producer:**

while (true)

{

/\* produce an item in nextproduced \*/

while (counter == BUFFER\_SIZE) ;

/\* do nothing \*/

buffer[in] = next\_produced;

in = (in + 1) % BUFFER\_SIZE;

counter++;

}

**Consumer:**

while (true)

{

while (counter == 0) ; /\* do nothing \*/

next\_consumed = buffer[out];

out = (out + 1) % BUFFER\_SIZE;

counter--;

/\* consume the item in next consumed \*/

}

1. Put the above written pseudocode (Critical Section) under the mutex lock to achieve mutual exclusion such that at one time only a single thread can execute its critical section.
2. As the user produces data the value of semaphore full is incremented and empty is decremented, opposite happens when user consumes data.
3. If the buffer is empty and user calls consumer thread than provoke the user to produce some data in the buffer.

**COMPLEXITY:**

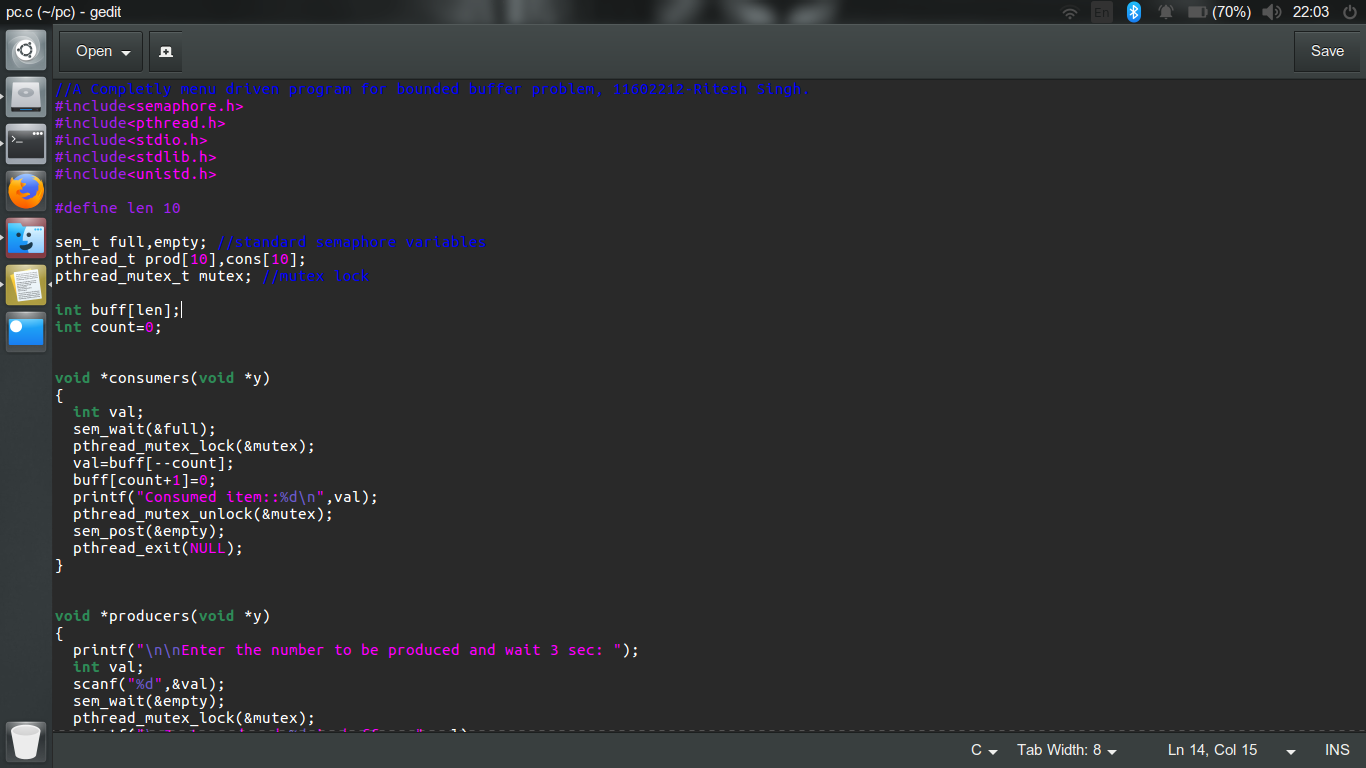
Complexity of producer thread: In my code it is **O(n)** as I have traversed the buffer to print it in a tabular manner where n=10.

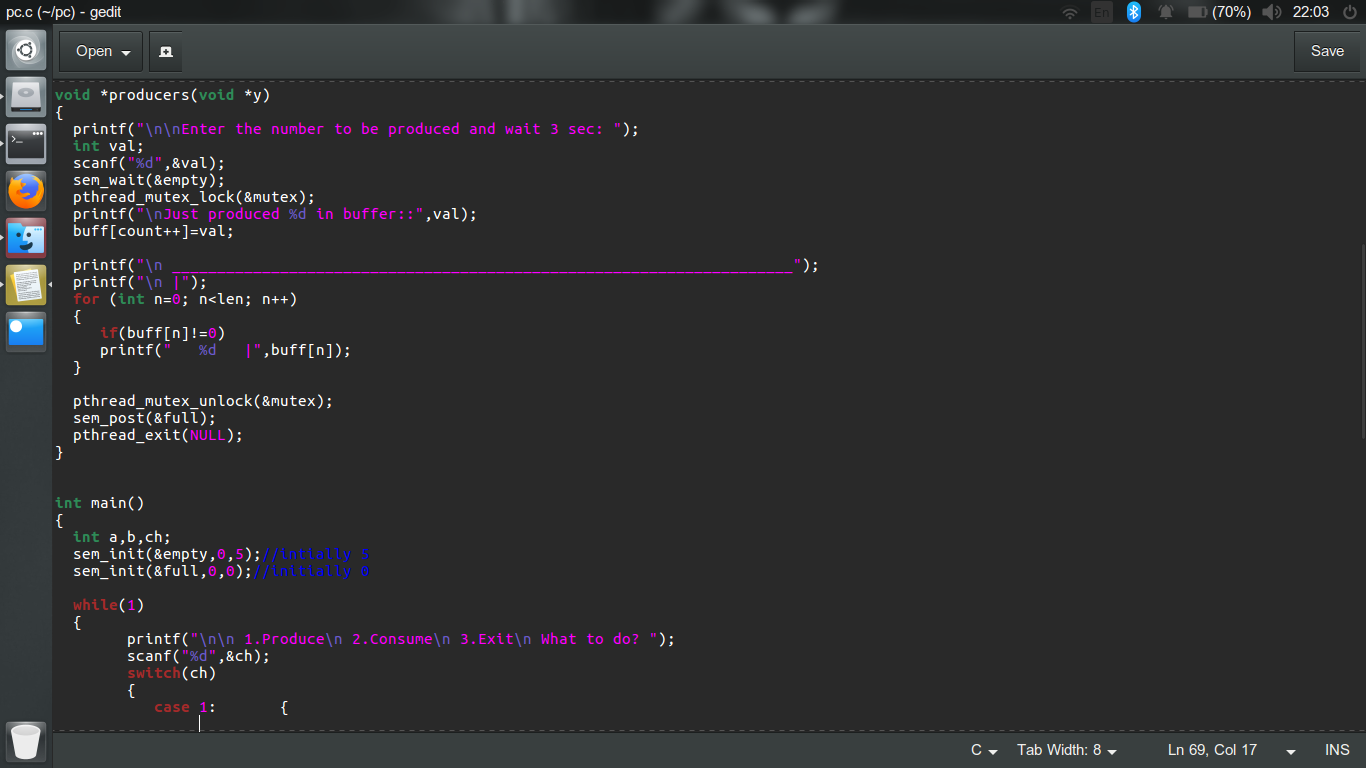
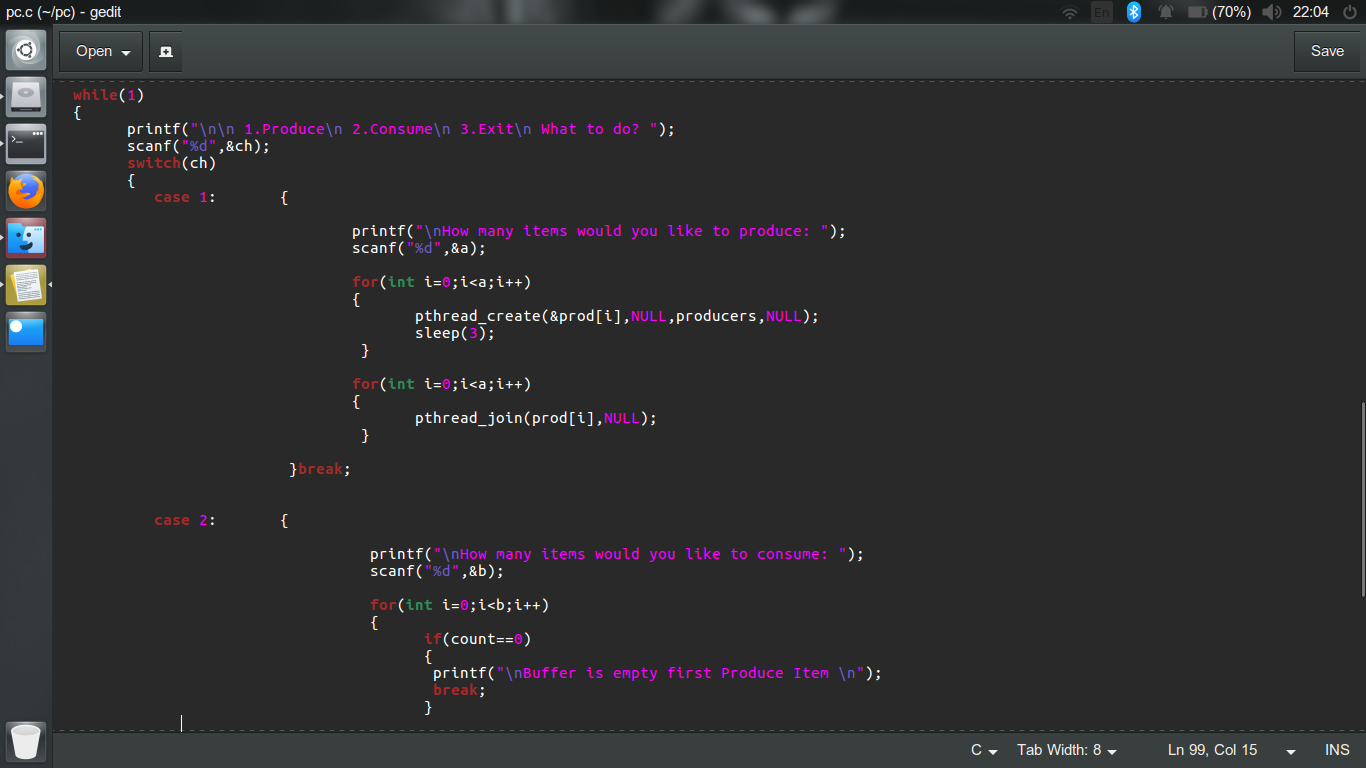
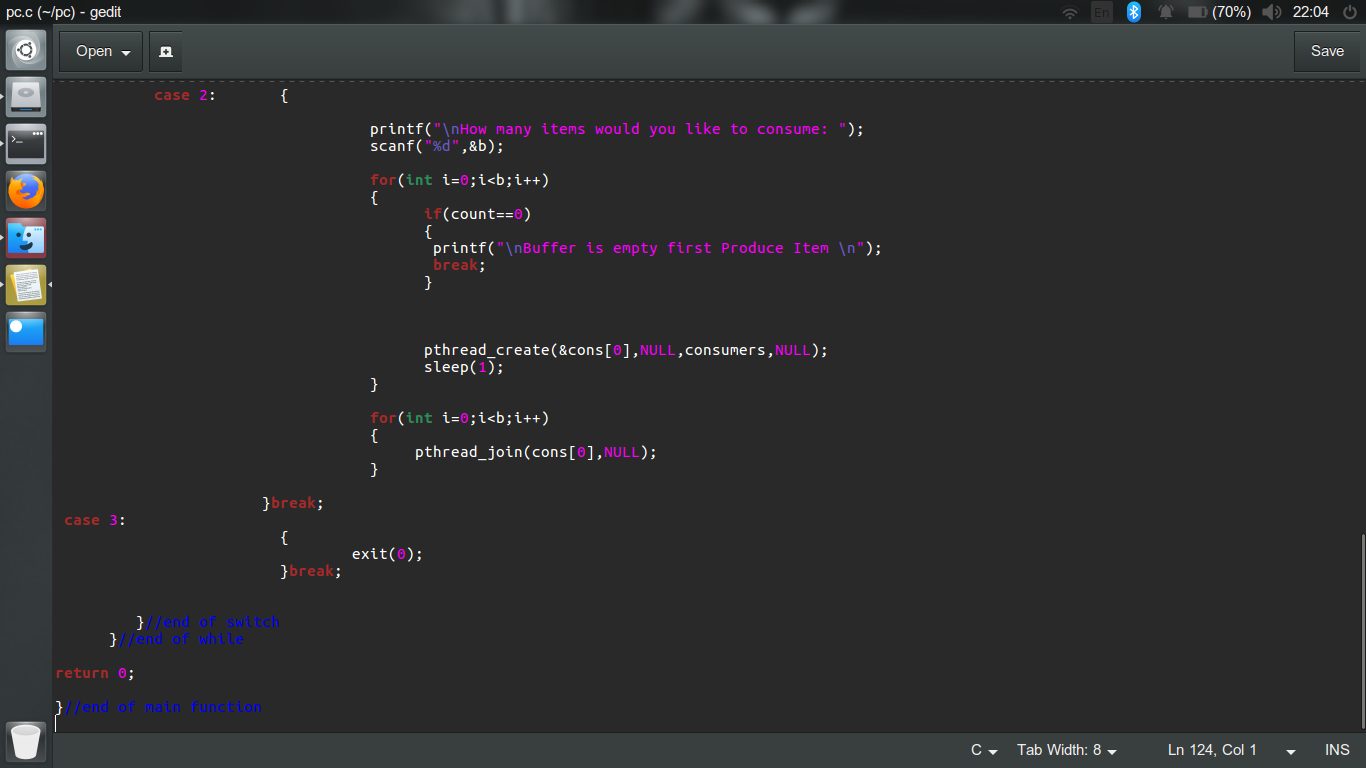
Complexity of the consumer thread: Here I have just decremented the counter without any traversal so the complexity is **O(1).**

Complexity of the main function: **O(n).**

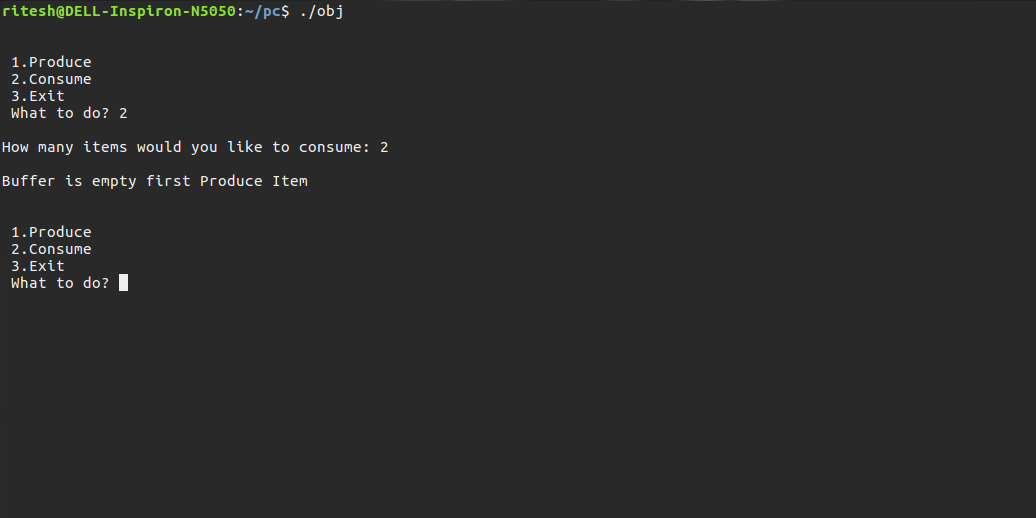
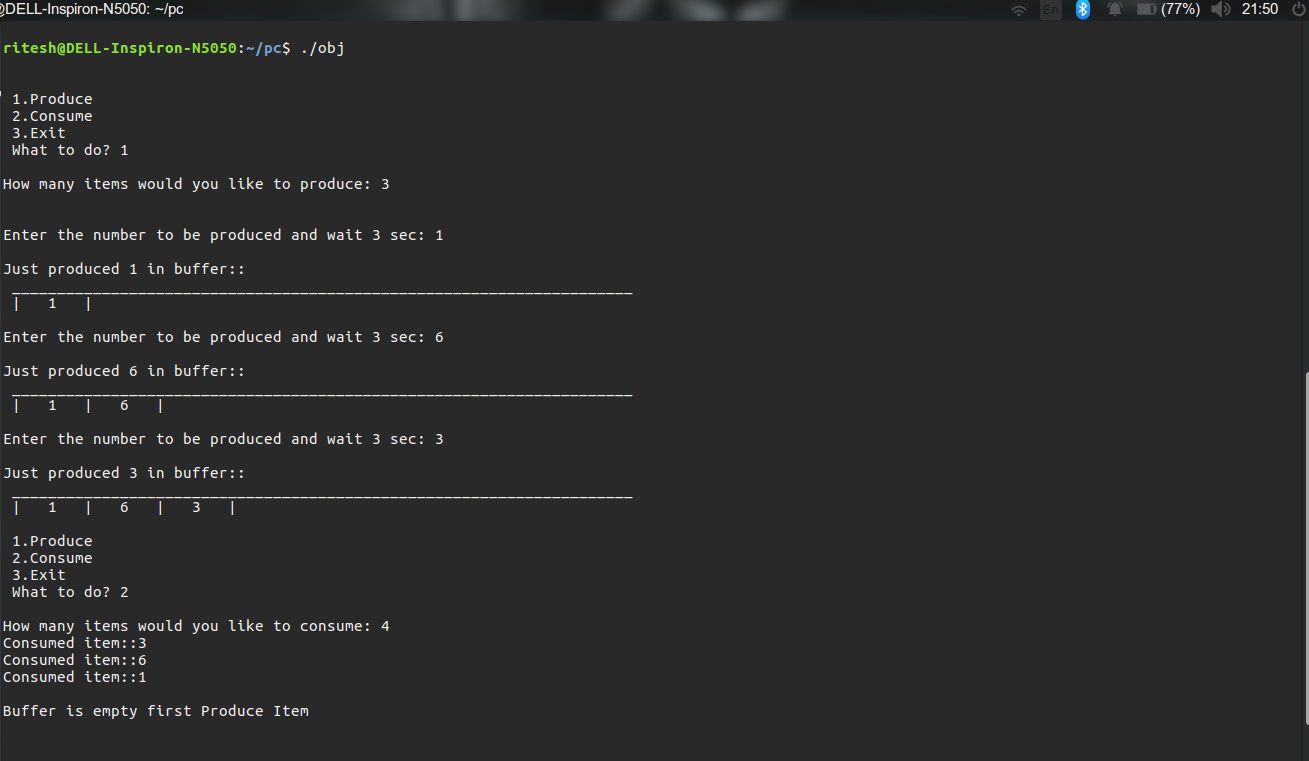
Overall Complexity: O(n)+O(1)+O(n) => **O(n)**. Where n is the size of buffer.

**CODE SNIPPETS:**

****

**** **** 

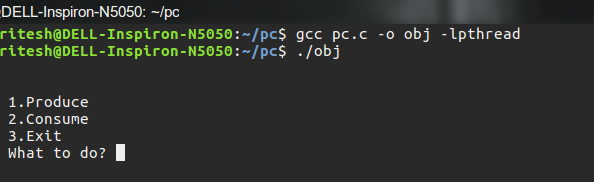
**CONSTRAINTS:**

1. If the producer has not produced anything and the user wants to consume than a message is shown to first produce data.
2. If the amount of data to be consumed is more than the buffer size than also user is prompted to produce more data.

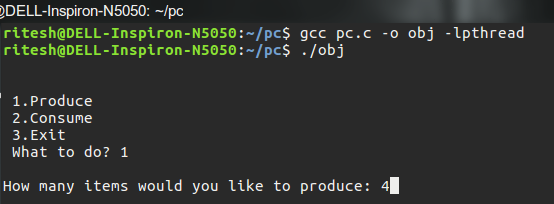
**BOUNDARY CONDITIONS:**

1. As I have taken the global buffer size of 10, maximum 10 elements can be kept in the buffer. Consumer needs to consume for adding more elements.
2. The code does not exits until the user wants to stop producing or consuming as my program is menu driven.

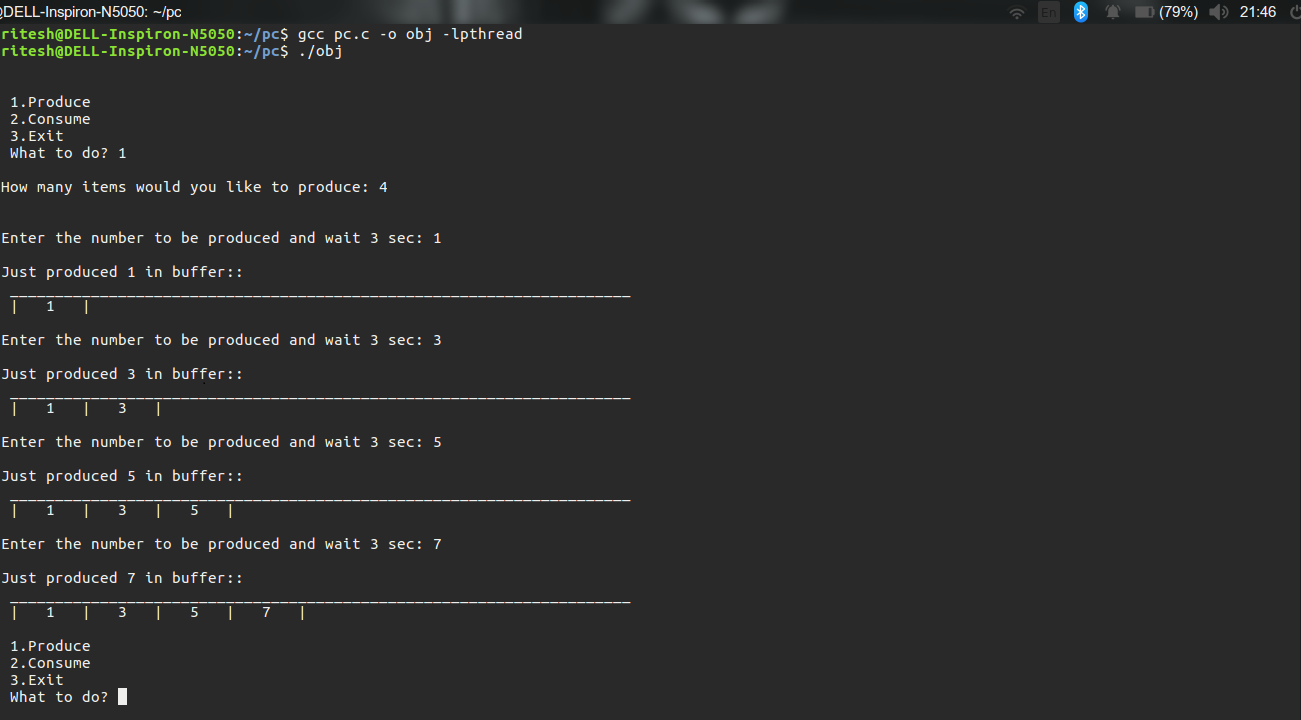
**TEST CASES WITH OUTPUT:**

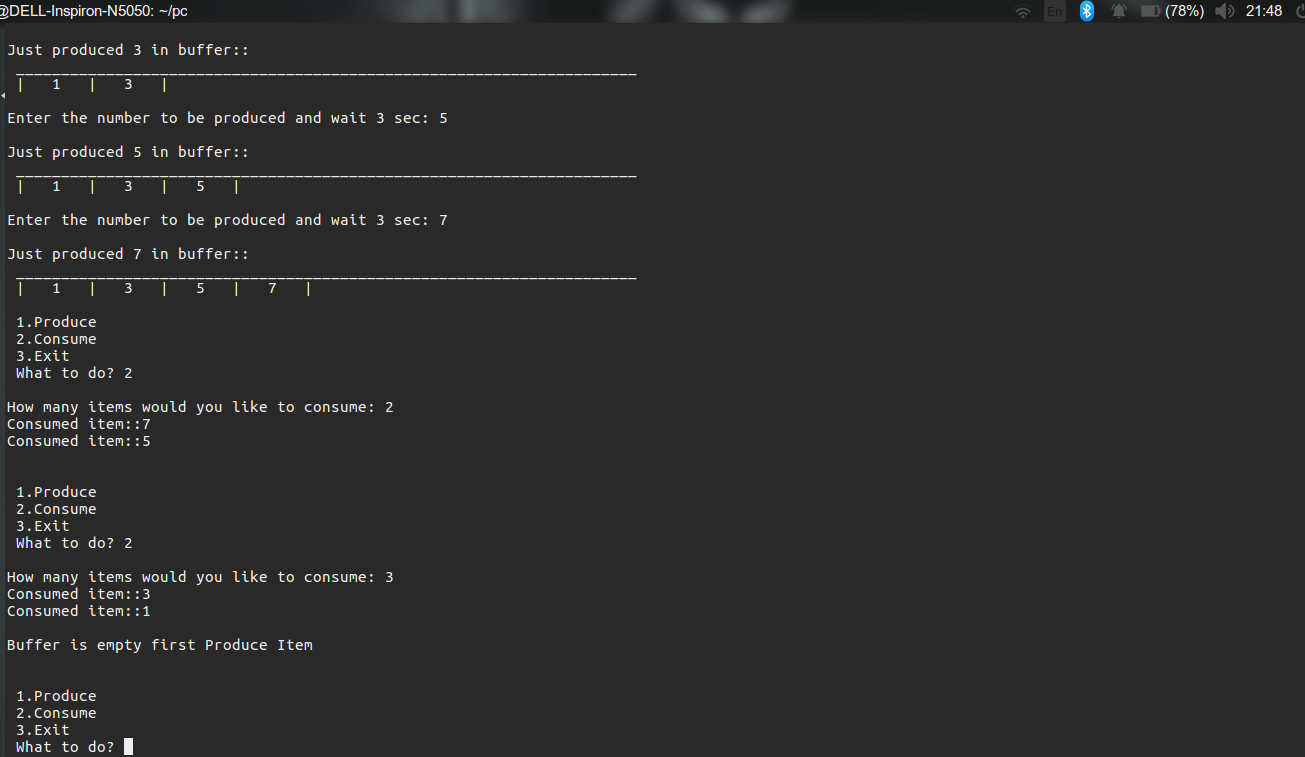
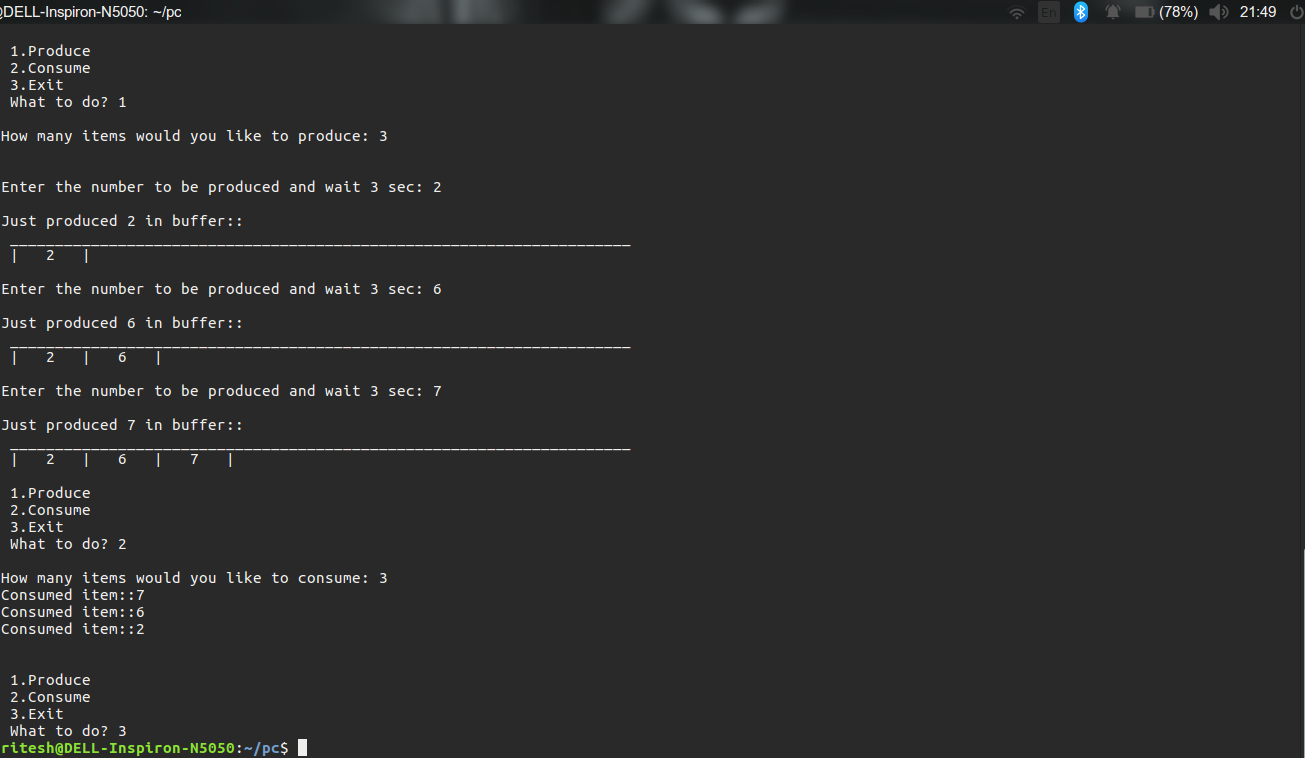
****

1. Choosing to produce 4 items.



1. Producing 1,3,5,7.



1. Consuming 2 items than again consuming 3 items.
2. Exiting the program.

I have applied every kind of input and the code has successful output for every input and every possibility to a deadlock is avoided.

**GITHUB LINK :** <https://github.com/ritesingh/K-1607-11602212-Producer_Consumer>