Assignment 4: **Thread synchronization using counting semaphores and mutual exclusion using mutex. Application to demonstrate: producer-consumer problem with counting semaphores and mutex.**

Name:	**Chaitanya Suresh Uge**
Roll Number:	**333064**
GR Number:	**21910718**
Division: C	**Batch: C3**

Theory:

1. **Thread synchronization:**
    - It is a mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as a critical section.
    - Processes' access to critical section is controlled by using synchronization techniques
    - When one thread starts executing the critical section, the other thread should wait until the first thread finishes. If proper synchronization techniques are not applied, it may cause a **race condition**
    - where the values of variables may be unpredictable and vary depending on the timings of context switches of the processes or threads.

2. **Producer Consumer Problem:**
    - It is an example of a multi-process synchronization problem.
    - The producer and the consumer share a common fixed-size buffer memory.

- The producer’s job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time.

3. **Mutex:**
    - A Mutex is a lock that we set before using a shared resource and released after using it.
    - When the lock is set, no other thread can access the locked region of code.
    - So we see that even if thread 2 is scheduled while thread 1 was not done accessing the shared resource and the code is locked by thread 1 using mutexes then thread 2 cannot even access that region of code.
    - So this ensures synchronized access of shared resources in the code.

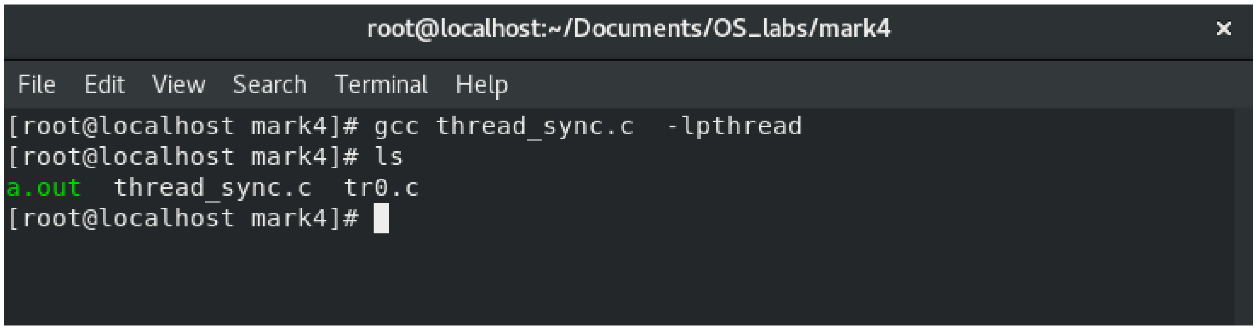
Source Code:

```
#include<stdio.h>
#include<semaphore.h>
#include<sys/types.h>
#include<pthread.h>
#include<unistd.h>
#include<stdlib.h>

#define BUFFER_SIZE 10

// Prototypes..
void *producer();
void *consumer();
void insert_item(int);
int remove_item();

// Declaring mutex
pthread_mutex_t mutex;

sem_t empty,full;

// Buffer is shared by both producer & consumer
int buffer[BUFFER_SIZE];
```

```c

// Counter is the global & shared variable
int counter;

pthread_t thread;


void initialize()
{
 printf("\n");
 pthread_mutex_init(&mutex,NULL);
 sem_init(&full,0,0);
 sem_init(&empty,0,BUFFER_SIZE);
}


void *producer()
{
 int item,wait_time;
 wait_time=rand()%5;
 sleep(wait_time)%5;
 item=rand()%10;
 sem_wait(&empty);
 pthread_mutex_lock(&mutex);

 // Produce / create item
 printf("Producer produced: %d \n",item);

 // Inserting item into buffer
 insert_item(item);
 pthread_mutex_unlock(&mutex);
 sem_post(&full);
}

void *consumer()
{
 int item,wait_time;
 wait_time=rand()%5;
 sleep(wait_time);
 sem_wait(&full);
 pthread_mutex_lock(&mutex);

 // Removing item from buffer for further processing
 item=remove_item();
 printf("Consumer consumed: %d\n",item);
 pthread_mutex_unlock(&mutex);
 sem_post(&empty);
}

// Insert item
void insert_item(int item)
{
 buffer[counter++]=item;
}


// Remove item
int remove_item()
{
 return buffer[--counter];
}



int main()
{
 int n1,n2;
 int i;
 printf("Enter number of Producers: ");
 scanf("%d",&n1);
 printf("\nEnter number of Consumers: ");
 scanf("%d",&n2);
 initialize();

 // create threads for all producers & consumers
 for(i=0;i<n1;i++)
  pthread_create(&thread,NULL,producer,NULL);
 for(i=0;i<n2;i++)
  pthread_create(&thread,NULL,consumer,NULL);
 sleep(5);
 exit(0);
}
```

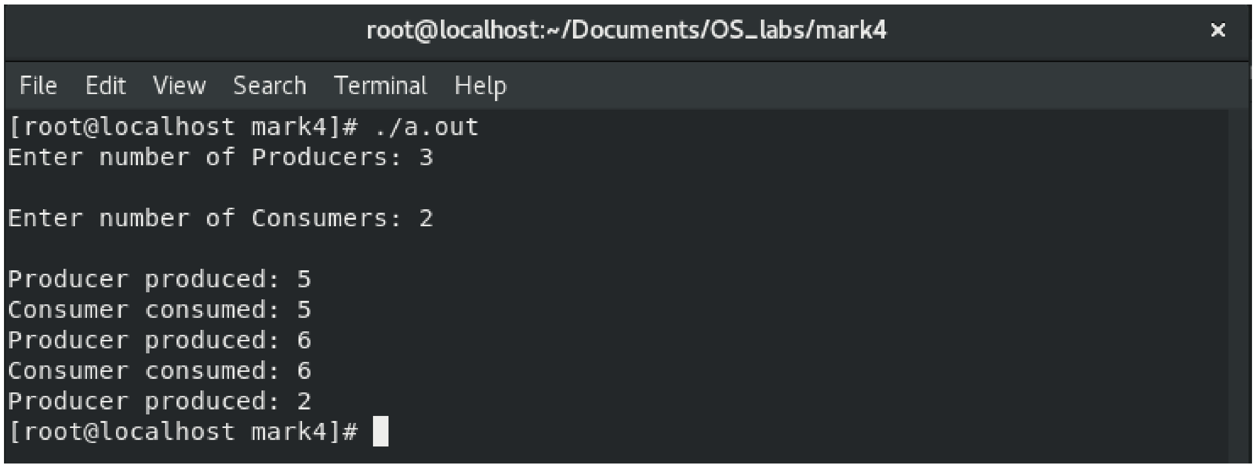
OUTPUT:

Compilation:


root@localhost:~/Documents/OS_labs/mark4
File Edit View Search Terminal Help

```
[root@localhost mark4]# gcc thread_sync.c  -lpthread
[root@localhost mark4]# ls
a.out  thread_sync.c  tr0.c
[root@localhost mark4]# 
```

Execution:


root@localhost:~/Documents/OS_labs/mark4
File Edit View Search Terminal Help

```
[root@localhost mark4]# ./a.out
Enter number of Producers: 3

Enter number of Consumers: 2

Producer produced: 5
Consumer consumed: 5
Producer produced: 6
Consumer consumed: 6
Producer produced: 2
[root@localhost mark4]#
```