NAME- KRISHNA KUMAR MAHTO
REGISTRATION- 16BIT0453

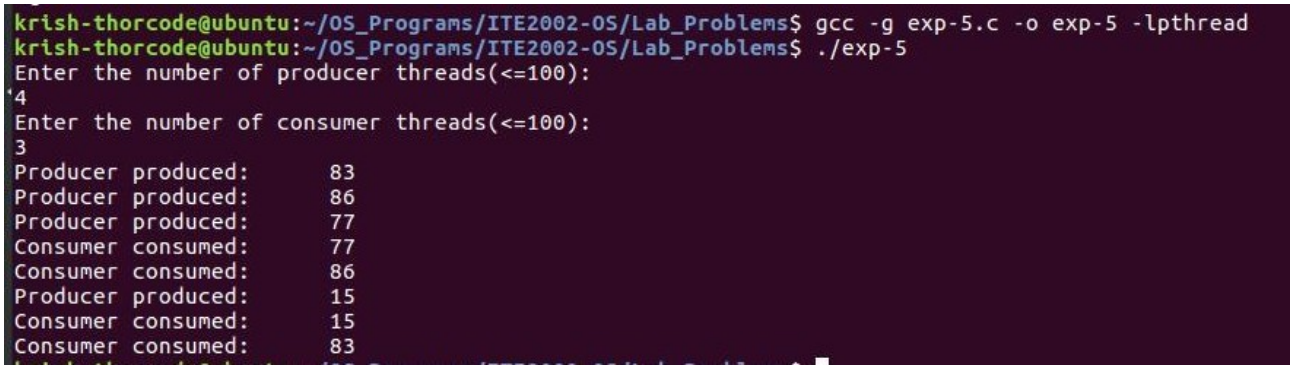
EXPERIMENT-5

CODE:

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

int shmid; // for storing return value from shmget; declaring
globally so that each thread func can access it without needing to
pass it as arg

sem_t semaphore;
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER; // one mutex for
client as well as producer so that reading/writing is in
synchronization and no race condtion occurs in accessing the
counter of buffer

int buffer[100],buff_counter=0;

void *consumer_thread(void *arg)
{
do
{
/*first wait, and then lock the mutex. doing otherwise will create
a deadlock in case when semaphore=0*/

sem_wait(&semaphore);
pthread_mutex_lock(&mutex);
printf("Consumer consumed:\t%d\n",buffer[--buff_counter]);
pthread_mutex_unlock(&mutex);
sleep(1);
}while(buff_counter>0); //do while so that if buffer is yet to be
filled, still consumers can enter

return NULL;
}

void *producer_thread(void *arg)
{
pthread_mutex_lock(&mutex);
buffer[buff_counter]=rand()%100;
printf("Producer produced:\t%d\n",buffer[buff_counter++]);
sem_post(&semaphore);
int gf=0;
sem_getvalue(&semaphore,&gf);
```

```
//printf("hahaha:%d \n\n",gf);
pthread_mutex_unlock(&mutex); // once the mutex is unlocked any
thread (consum/prod) may lock it

return NULL;
}

int main(int argc,char **argv)
{
//int gf=0;
sem_init(&semaphore,0,0);
//sem_getvalue(&semaphore,&gf);
//printf("haha: %d\n\n",gf);

int i,num_prod=101,num_consum=101; //assigning any random
number>100 to both so that the two loops which follow immediately
can run

for(;num_prod>100;)
{
puts("Enter the number of producer threads(<=100): ");
scanf("%d",&num_prod);
}

for(;num_consum>100;)
{
puts("Enter the number of consumer threads(<=100): ");
scanf("%d",&num_consum);
}

pthread_t producer_id[num_prod], consumer_id[num_consum];

for(i=0;i<num_prod;i++)
pthread_create(&producer_id[i],NULL,producer_thread,NULL);

for(i=0;i<num_consum;i++)
pthread_create(&consumer_id[i],NULL,consumer_thread,NULL);

for(i=0;i<num_prod;i++)
pthread_join(producer_id[i],NULL);

for(i=0;i<num_consum;i++)
pthread_join(consumer_id[i],NULL);


return 0;

}
```

OUTPUT:


krish-thorcode@ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems$ gcc -g exp-5.c -o exp-5 -lpthread
krish-thorcode@ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems$ ./exp-5
Enter the number of producer threads(<=100):
4
Enter the number of consumer threads(<=100):
3
Producer produced: 83
Producer produced: 86
Producer produced: 77
Consumer consumed: 77
Consumer consumed: 86
Producer produced: 15
Consumer consumed: 15
Consumer consumed: 83