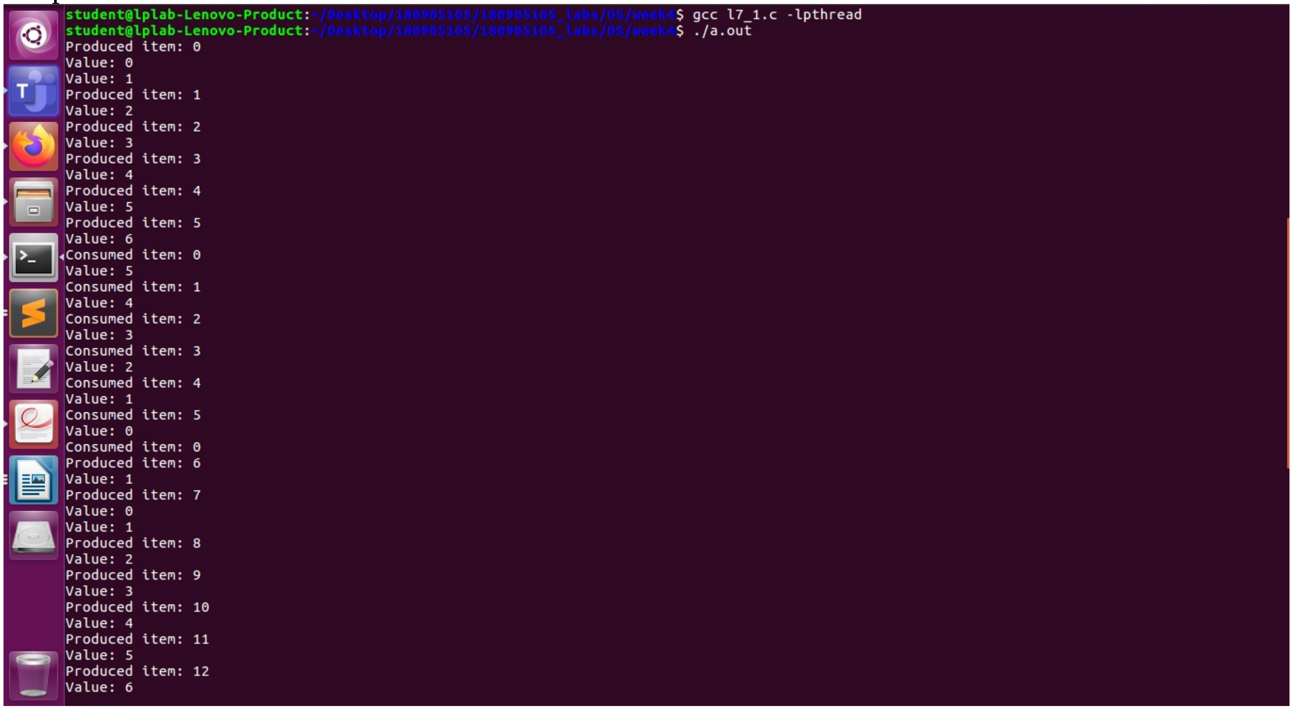
HRITIK BANSAL, 180905105, CSE A, B1
LAB7:

Q1)
**Modify the above Producer-Consumer program so that, a producer can produce at the
most 10 times more than what the consumer has consumed.**

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

int buf[10], f=0, r=0, value;
sem_t mutex, full, empty;

void * produce(void * arg)
{
    for (int i=0;i<20;i++)
    {
        sem_wait(&empty);
        sem_wait(&mutex);
        printf("Produced item: %d\n",i);
        buf[(++r)%10]=i;
        sleep(1);
        sem_post(&mutex);
        sem_post(&full);
        sem_getvalue(&full, &value);
        printf("Value: %d\n", value);
    }
}

void * consume(void * arg)
{
    int item;
    for (int i=0;i<20;i++)
    {
        sem_wait(&full);
        sem_getvalue(&full, &value);
        printf("Value: %d\n", value);
        sem_wait(&mutex);
        item=buf[(++f)%10];
        printf("Consumed item: %d\n",item);
        sleep(1);
        sem_post(&mutex);
        sem_post(&empty);
    }
}

void main()
```

```
{
    pthread_t t1,t2;
    sem_init(&mutex, 0, 1);
    sem_init(&full, 0, 1);
    sem_init(&empty, 0, 10);
    pthread_create(&t1, NULL, produce, NULL);
    pthread_create(&t2, NULL, consume, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
}
```

Output:

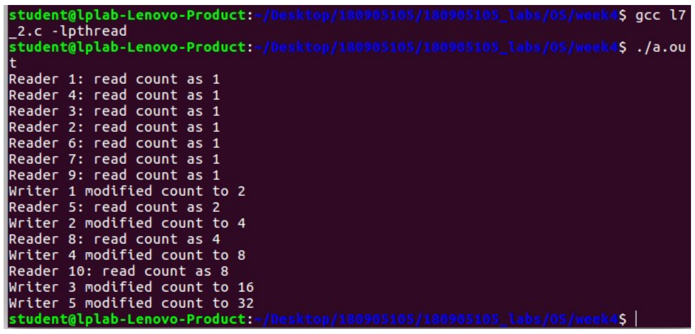


```
student@lplab-Lenovo-Product:~/Desktop/180905105/180905105_labs/OS/week4$ gcc l7_1.c -lpthread
student@lplab-Lenovo-Product:~/Desktop/180905105/180905105_labs/OS/week4$ ./a.out
Produced item: 0
Value: 0
Value: 1
Produced item: 1
Value: 2
Produced item: 2
Value: 3
Produced item: 3
Value: 4
Produced item: 4
Value: 5
Produced item: 5
Value: 6
Consumed item: 0
Value: 5
Consumed item: 1
Value: 4
Consumed item: 2
Value: 3
Consumed item: 3
Value: 2
Consumed item: 4
Value: 1
Consumed item: 5
Value: 0
Consumed item: 0
Produced item: 6
Value: 1
Produced item: 7
Value: 0
Value: 1
Produced item: 8
Value: 2
Produced item: 9
Value: 3
Produced item: 10
Value: 4
Produced item: 11
Value: 5
Produced item: 12
Value: 6
```

Q2)Write a C Program for the first readers-writers problem using semaphores.

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

sem_t wrt;
pthread_mutex_t mutex;
int cnt = 1;
int numreader = 0;

void *writer(void *wno)
{
    sem_wait(&wrt);
    cnt = cnt * 2;
    printf("Writer %d modified count to %d\n", (*((int *)wno)), cnt);
    sem_post(&wrt);
}
void *reader(void *rno)
{
    pthread_mutex_lock(&mutex);
    numreader++;

    if (numreader == 1)
        sem_wait(&wrt);

    pthread_mutex_unlock(&mutex);
    printf("Reader %d: read count as %d\n", *((int *)rno), cnt);
    pthread_mutex_lock(&mutex);

    numreader--;

    if (numreader == 0)
        sem_post(&wrt);

    pthread_mutex_unlock(&mutex);
}

int main()
{
    pthread_t read[10], write[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt, 0, 1);

    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    for (int i = 0; i < 10; i++)
        pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);

    for (int i = 0; i < 5; i++)
        pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);

    for (int i = 0; i < 10; i++)
        pthread_join(read[i], NULL);

    for (int i = 0; i < 5; i++)
        pthread_join(write[i], NULL);

    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);
    return 0;
}
```

Output:



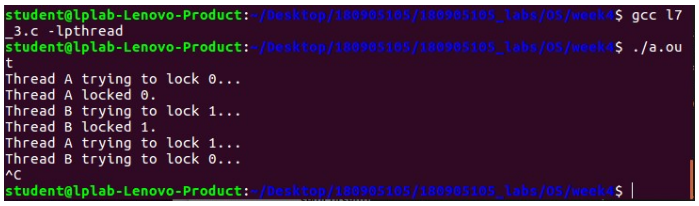
```
student@lplab-Lenovo-Product:~/Desktop/180905105/180905105_labs/OS/week4$ gcc l7
_2.c -lpthread
student@lplab-Lenovo-Product:~/Desktop/180905105/180905105_labs/OS/week4$ ./a.ou
t
Reader 1: read count as 1
Reader 4: read count as 1
Reader 3: read count as 1
Reader 2: read count as 1
Reader 6: read count as 1
Reader 7: read count as 1
Reader 9: read count as 1
Writer 1 modified count to 2
Reader 5: read count as 2
Writer 2 modified count to 4
Reader 8: read count as 4
Writer 4 modified count to 8
Reader 10: read count as 8
Writer 3 modified count to 16
Writer 5 modified count to 32
student@lplab-Lenovo-Product:~/Desktop/180905105/180905105_labs/OS/week4$ 
```

Q3)Write a code to access a shared resource which causes deadlock using improper use of semaphore.

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

#define PERM 0660

int semId;

int initSem(int id, int num, int initVal)
{
	return semctl(id, num, SETVAL, initVal);
}

int P(int id, int num)
{
	struct sembuf operationList[1];
	operationList[0].sem_num = num;
	operationList[0].sem_op = -1;
	operationList[0].sem_flg = 0;
	return semop(id, operationList, 1);
}

int V(int id, int num)
{
	struct sembuf operationList[1];
	operationList[0].sem_num = num;
	operationList[0].sem_op = 1;
	operationList[0].sem_flg = 0;
	return semop(id, operationList, 1);
}


void* func1(void* no)
{
	printf("Thread A trying to lock 0...\n");
	P(semId, 0);
	printf("Thread A locked 0.\n");
	usleep(5 * 1000);
	printf("Thread A trying to lock 1...\n");
	P(semId, 1);
	printf("Thread A locked 1.\n");
	V(semId, 0);
	V(semId, 1);
}
```

```
void* func2(void* no)
{
        printf("Thread B trying to lock 1...\n");
        P(semId, 1);
        printf("Thread B locked 1.\n");
        usleep(50 * 1000);
        printf("Thread B trying to lock 0...\n");
        P(semId, 0);
        printf("Thread B locked 0.\n");
        V(semId, 0);
        V(semId, 1);
}


int main(int argc, char* argv[])
{
        semId = semget(ftok(argv[0], 'A'), 2, IPC_CREAT | PERM);
        initSem(semId, 0, 1);
        initSem(semId, 1, 1);

        pthread_t t[2];
        pthread_create(&t[0], NULL, func1, NULL);
        pthread_create(&t[1], NULL, func2, NULL);
        for (int i = 0 ; i < 2 ; i++)
                pthread_join(t[i], NULL);

        printf("This is not printed in case of deadlock\n");
        semctl(semId, 0, IPC_RMID, 0);
        semctl(semId, 1, IPC_RMID, 0);
        return 0;
}
```

OUTPUT:



```
student@lplab-Lenovo-Product:~/Desktop/180905105/180905105_labs/OS/week4$ gcc l7
_3.c -lpthread
student@lplab-Lenovo-Product:~/Desktop/180905105/180905105_labs/OS/week4$ ./a.ou
t
Thread A trying to lock 0...
Thread A locked 0.
Thread B trying to lock 1...
Thread B locked 1.
Thread A trying to lock 1...
Thread B trying to lock 0...
^C
student@lplab-Lenovo-Product:~/Desktop/180905105/180905105_labs/OS/week4$ |
```

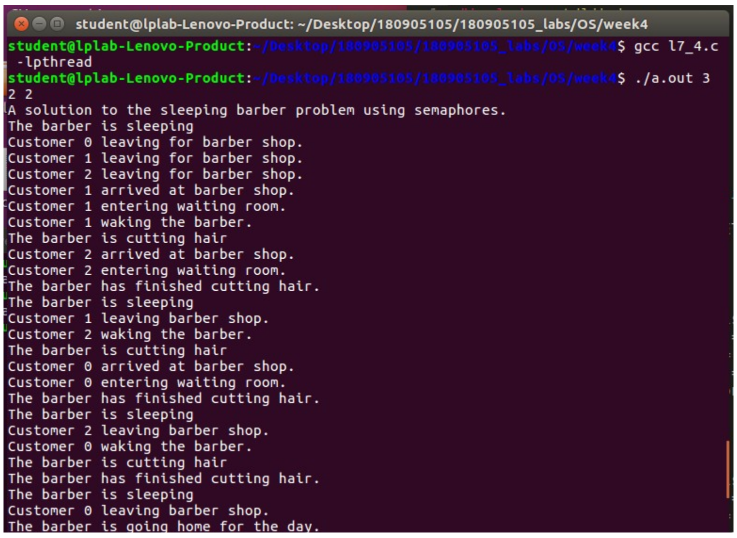
**Q4)Write a program using semaphone to demonstrate the working of sleeping barber
problem.**

```
#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>
```

```
#include <semaphore.h>

#define MAX_CUSTOMERS 25

sem_t waitingRoom;
sem_t barberChair;
sem_t barberPillow;
sem_t seatBelt;
int allDone = 0;

void randwait(int secs)
{
   int len;
   len = (int) ((drand48() * secs) + 1);
   sleep(len);
}

void* customer(void *number)
{
   int num = *(int *)number;
   printf("Customer %d leaving for barber shop.\n", num);
   randwait(5);
   printf("Customer %d arrived at barber shop.\n", num);
   sem_wait(&waitingRoom);
   printf("Customer %d entering waiting room.\n", num);
   sem_wait(&barberChair);
   sem_post(&waitingRoom);
   printf("Customer %d waking the barber.\n", num);
   sem_post(&barberPillow);
   sem_wait(&seatBelt);
   sem_post(&barberChair);
   printf("Customer %d leaving barber shop.\n", num);
}

void* barber(void *junk)
```

```
{
    while (!allDone)
    {
        printf("The barber is sleeping\n");
        sem_wait(&barberPillow);

        if (!allDone)
        {
            printf("The barber is cutting hair\n");
            randwait(3);
            printf("The barber has finished cutting hair.\n");
            sem_post(&seatBelt);
        }
        else printf("The barber is going home for the day.\n");
    }
}

int main(int argc, char *argv[])
{
    pthread_t btid;
    pthread_t tid[MAX_CUSTOMERS];
    long RandSeed;
    int i, numCustomers, numChairs;
    int Number[MAX_CUSTOMERS];

    if (argc != 4)
    {
        printf("Use: SleepBarber <Num Customers> <Num Chairs> <rand seed>\n");
        exit(-1);
    }
    numCustomers = atoi(argv[1]);
    numChairs = atoi(argv[2]);
    RandSeed = atol(argv[3]);

    if (numCustomers > MAX_CUSTOMERS)
    {
        printf("The maximum number of Customers is %d.\n", MAX_CUSTOMERS);
        exit(-1);
    }
    printf("A solution to the sleeping barber problem using semaphores.\n");
    srand48(RandSeed);

    for (i = 0; i < MAX_CUSTOMERS; i++)
        Number[i] = i;

    sem_init(&waitingRoom, 0, numChairs);
    sem_init(&barberChair, 0, 1);
    sem_init(&barberPillow, 0, 0);
    sem_init(&seatBelt, 0, 0);
    pthread_create(&btid, NULL, barber, NULL);

    for (i = 0; i < numCustomers; i++)
        pthread_create(&tid[i], NULL, customer, (void *)&Number[i]);

    for (i = 0; i < numCustomers; i++)
        pthread_join(tid[i], NULL);

    allDone = 1;
    sem_post(&barberPillow);
    pthread_join(btid, NULL);
}
```

Output:



```
student@lplab-Lenovo-Product: ~/Desktop/180905105/180905105_labs/OS/week4
student@lplab-Lenovo-Product:~/Desktop/180905105/180905105_labs/OS/week4$ gcc l7_4.c
 -lpthread
student@lplab-Lenovo-Product:~/Desktop/180905105/180905105_labs/OS/week4$ ./a.out 3
2 2
A solution to the sleeping barber problem using semaphores.
The barber is sleeping
Customer 0 leaving for barber shop.
Customer 1 leaving for barber shop.
Customer 2 leaving for barber shop.
Customer 1 arrived at barber shop.
Customer 1 entering waiting room.
Customer 1 waking the barber.
The barber is cutting hair
Customer 2 arrived at barber shop.
Customer 2 entering waiting room.
The barber has finished cutting hair.
The barber is sleeping
Customer 1 leaving barber shop.
Customer 2 waking the barber.
The barber is cutting hair
Customer 0 arrived at barber shop.
Customer 0 entering waiting room.
The barber has finished cutting hair.
The barber is sleeping
Customer 2 leaving barber shop.
Customer 0 waking the barber.
The barber is cutting hair
The barber has finished cutting hair.
The barber is sleeping
Customer 0 leaving barber shop.
The barber is going home for the day.
```