**OPERATING SYSTEM LAB 7**

**Q1.**

PROGRAM:

#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);

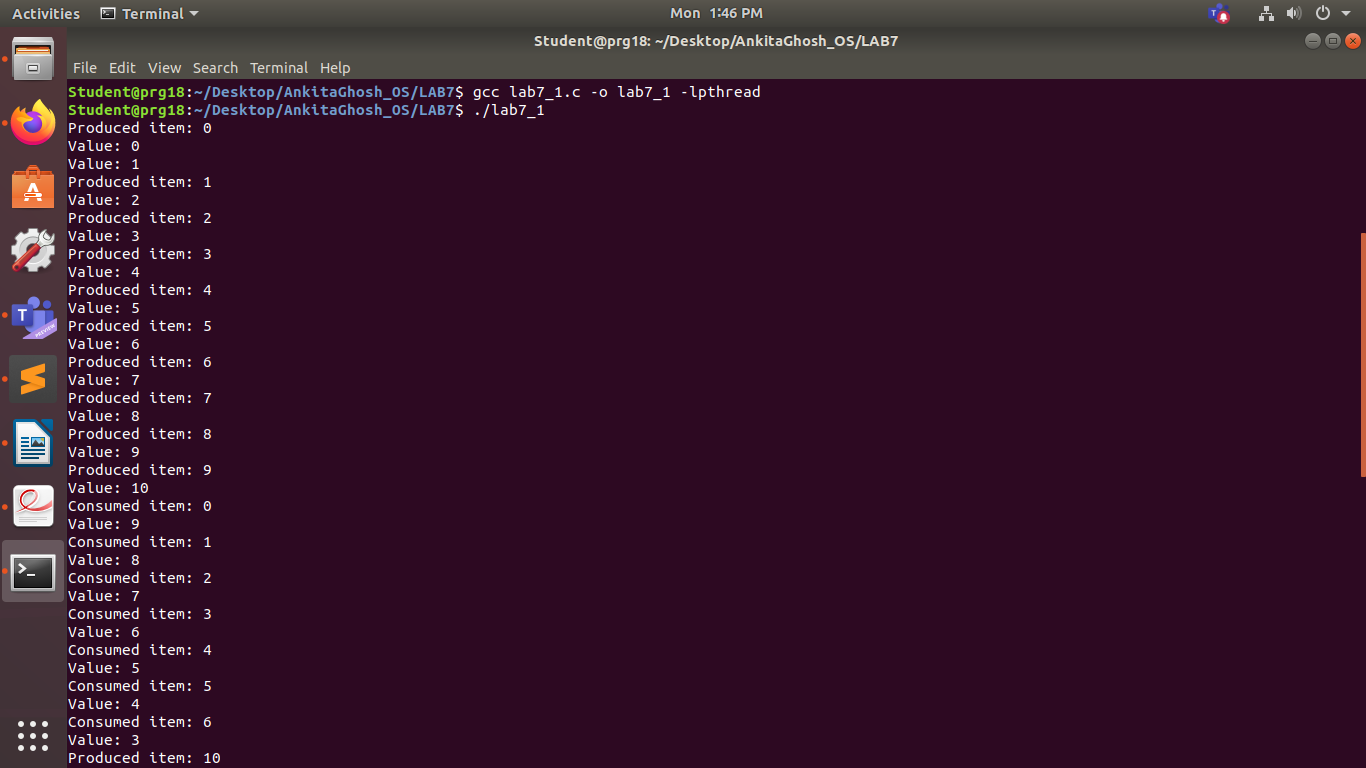
pthread\_create(&t1, NULL, produce, NULL);

pthread\_create(&t2, NULL, consume, NULL);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

}  
  
OUTPUT:



**Q2.**

PROGRAM:

#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);

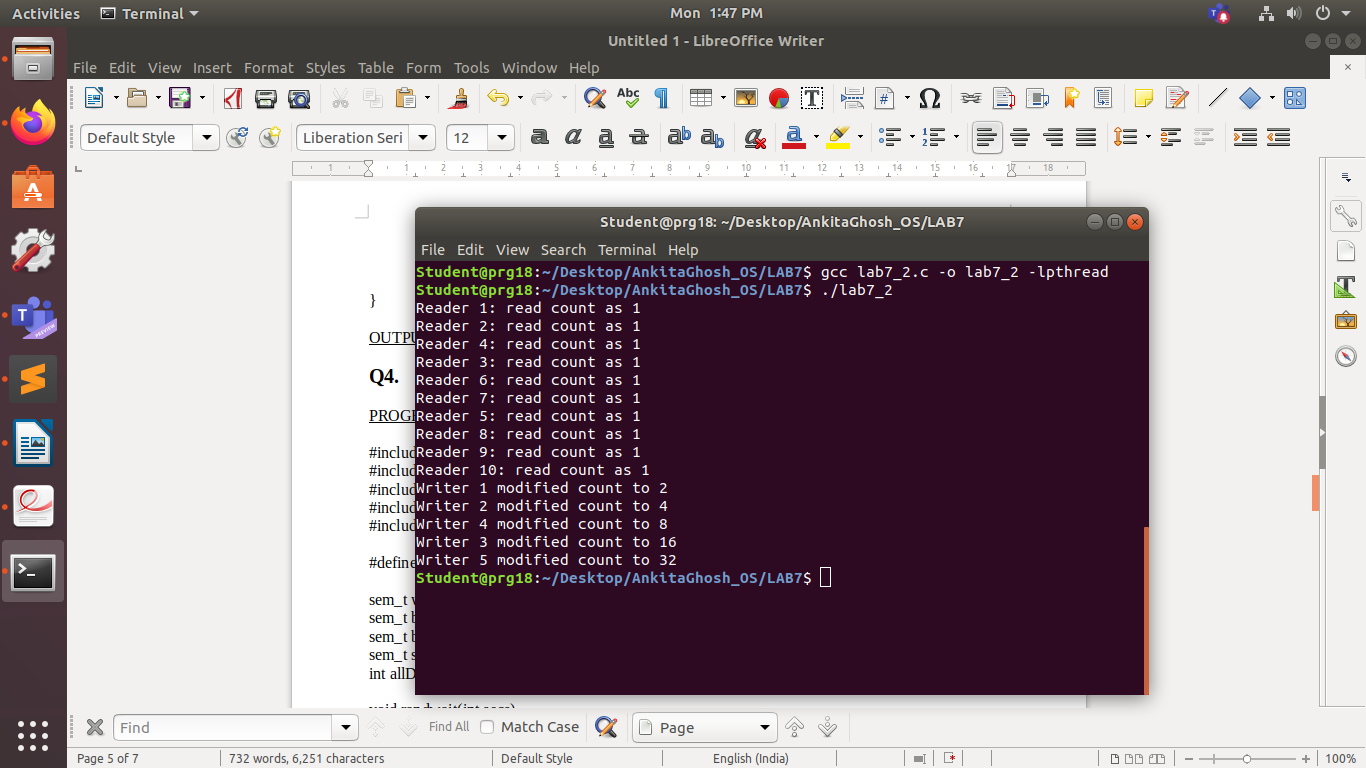
pthread\_mutex\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

OUTPUT:



**Q3.**

PROGRAM:

#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);

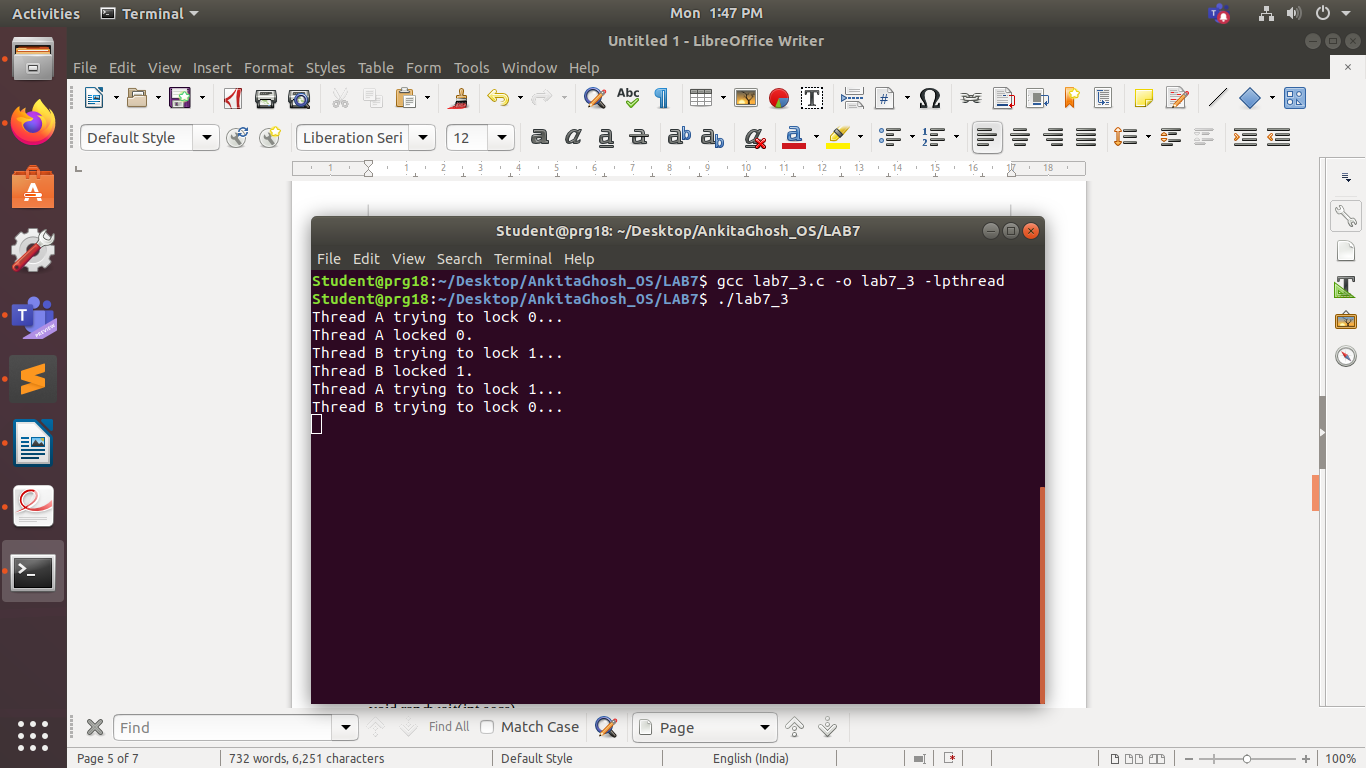
printf("This is not printed in case of deadlock\n");

semctl(semId, 0, IPC\_RMID, 0);

semctl(semId, 1, IPC\_RMID, 0);

return 0;

}  
  
OUTPUT:



**Q4.**

PROGRAM:

#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:

