**Registration Number:19BCE2119**

**Name: Gaurav Kumar Singh**

**Course: CSE2005 Operating Systems**

**Digital Assignment 3**

**(a) Implement the solution for reader – writer’s problem.**

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

sem\_t mutex,writeblock;

int data = 0,rcount = 0;

void \*reader(void \*arg)

{

int f;

f = ((int)arg);

sem\_wait(&mutex);

rcount = rcount + 1;

if(rcount==1)

sem\_wait(&writeblock);

sem\_post(&mutex);

printf("Data read by the reader%d is %d\n",f,data);

sleep(1);

sem\_wait(&mutex);

rcount = rcount - 1;

if(rcount==0)

sem\_post(&writeblock);

sem\_post(&mutex);

}

void \*writer(void \*arg)

{

int f;

f = ((int) arg);

sem\_wait(&writeblock);

data++;

printf("Data writen by the writer%d is %d\n",f,data);

sleep(1);

sem\_post(&writeblock);

}

int main()

{

int i,b;

pthread\_t rtid[5],wtid[5];

sem\_init(&mutex,0,1);

sem\_init(&writeblock,0,1);

for(i=0;i<=2;i++)

{

pthread\_create(&wtid[i],NULL,writer,(void \*)i);

pthread\_create(&rtid[i],NULL,reader,(void \*)i);

}

for(i=0;i<=2;i++)

{

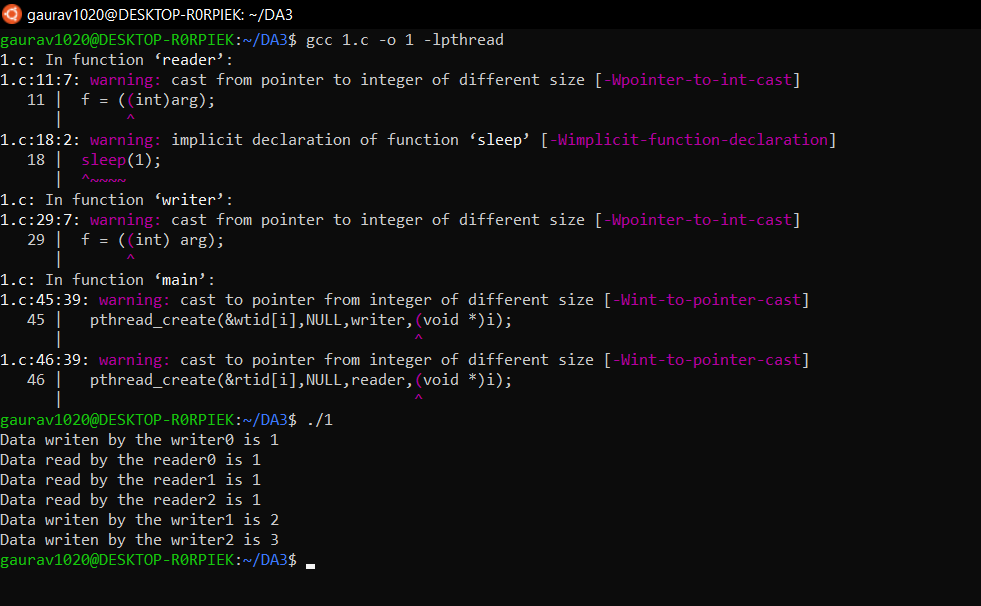
pthread\_join(wtid[i],NULL);

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

}

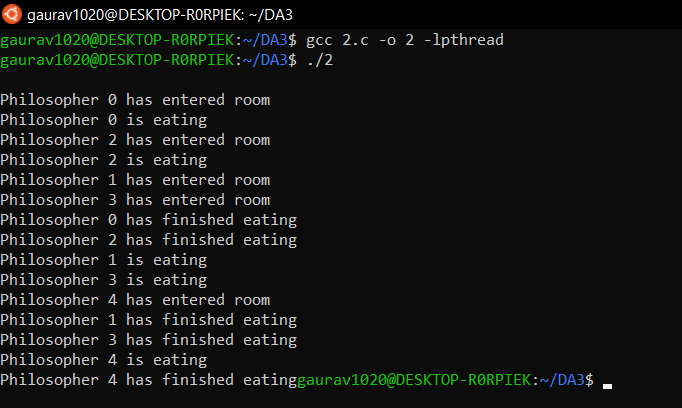
return 0;

}



**(b) Implement the solution for dining philosopher’s problem.**

#include<stdio.h>  
#include<stdlib.h>  
#include<pthread.h>  
#include<semaphore.h>  
#include<unistd.h>  
  
sem\_t room;  
sem\_t chopstick[5];  
  
void \* philosopher(void \*);  
void eat(int);  
int main()  
{  
 int i,a[5];  
 pthread\_t tid[5];  
  
 sem\_init(&room,0,4);  
  
 for(i=0;i<5;i++)  
 sem\_init(&chopstick[i],0,1);  
  
 for(i=0;i<5;i++){  
 a[i]=i;  
 pthread\_create(&tid[i],**NULL**,philosopher,(void \*)&a[i]);  
 }  
 for(i=0;i<5;i++)  
 pthread\_join(tid[i],**NULL**);  
}  
  
void \* philosopher(void \* num)  
{  
 int phil=\*(int \*)num;  
  
 sem\_wait(&room);  
 printf("\nPhilosopher %d has entered room",phil);  
 sem\_wait(&chopstick[phil]);  
 sem\_wait(&chopstick[(phil+1)%5]);  
  
 eat(phil);  
 sleep(2);  
 printf("\nPhilosopher %d has finished eating",phil);  
  
 sem\_post(&chopstick[(phil+1)%5]);  
 sem\_post(&chopstick[phil]);  
 sem\_post(&room);  
}  
  
void eat(int phil)  
{  
 printf("\nPhilosopher %d is eating",phil);  
}



**(c) Implement the solution for producer consumer problem**

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>

#define MaxItems 5

#define BufferSize 5

sem\_t empty;

sem\_t full;

int in = 0;

int out = 0;

int buffer[BufferSize];

pthread\_mutex\_t mutex;

void \*producer(void \*pno)

{

int item;

for(int i = 0; i < MaxItems; i++) {

item = rand(); // Produce an random item

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[in] = item;

printf("Producer %d: Insert Item %d at %d\n", \*((int \*)pno),buffer[in],in);

in = (in+1)%BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

}

void \*consumer(void \*cno)

{

for(int i = 0; i < MaxItems; i++) {

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[out];

printf("Consumer %d: Remove Item %d from %d\n",\*((int \*)cno),item, out);

out = (out+1)%BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

}

int main()

{

pthread\_t pro[5],con[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&empty,0,BufferSize);

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

int a[5] = {1,2,3,4,5}; //Just used for numbering the producer and consumer

for(int i = 0; i < 5; i++) {

pthread\_create(&pro[i], NULL, (void \*)producer, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

pthread\_create(&con[i], NULL, (void \*)consumer, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

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

}

for(int i = 0; i < 5; i++) {

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

}

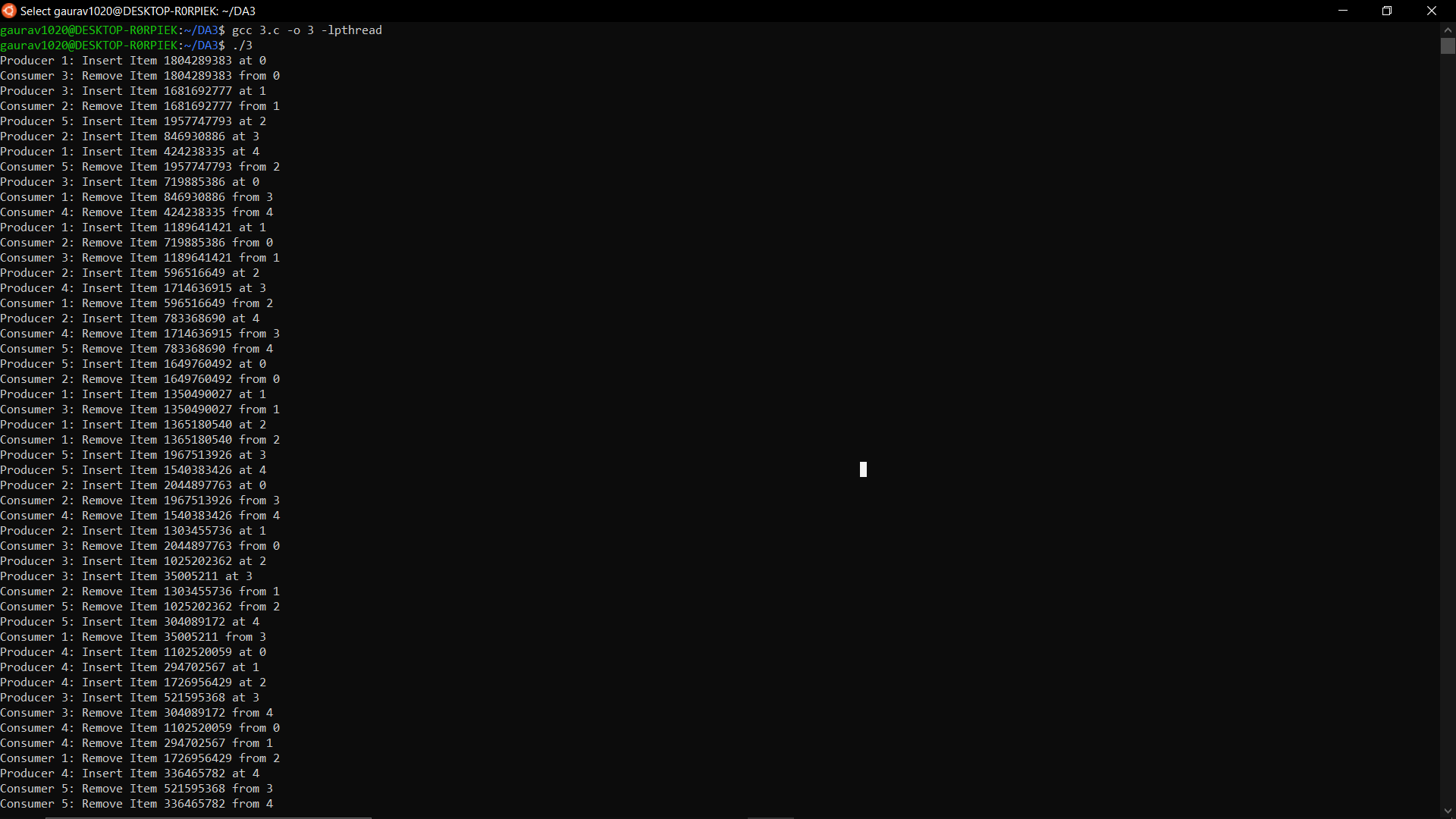
pthread\_mutex\_destroy(&mutex);

sem\_destroy(&empty);

sem\_destroy(&full);

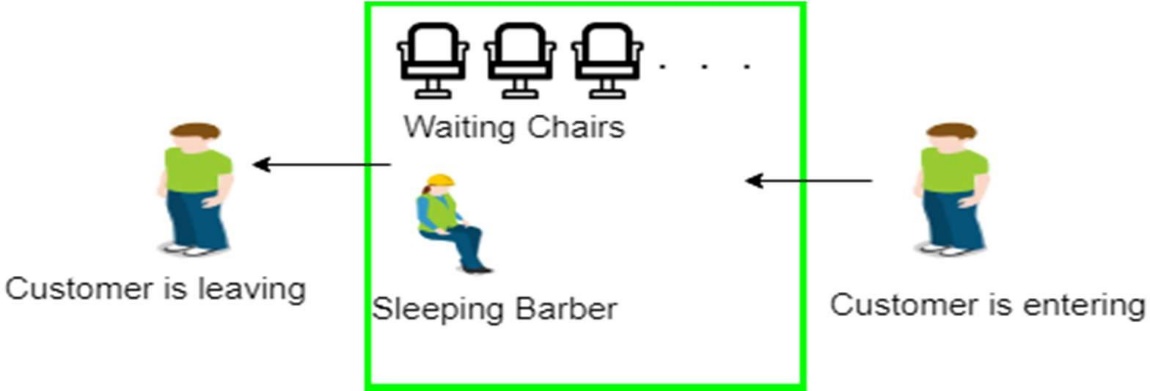
return 0;

}



**(d) The analogy is based upon a hypothetical barber shop with one barber. There is a barber shop which has one barber, one barber chair, and n chairs for waiting for customers if there are any to sit on the chair.**

* + **If there is no customer, then the barber sleeps in his own chair.**
  + **When a customer arrives, he has to wake up the barber.**
  + **If there are many customers and the barber is cutting a customer’s hair, then the remaining customers either wait if there are empty chairs in the waiting room or they leave if no chairs are empty.**



#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <time.h>

#include <pthread.h>

#include <semaphore.h>

#define MAX\_CUSTOMERS 25

void \*customer(void \*num);

void \*barber(void \*);

void randwait(int secs);

sem\_t waitingRoom;

sem\_t barberChair;

sem\_t barberPillow;

sem\_t seatBelt;

int allDone = 0;

int main(int argc, char \*argv[]) {

pthread\_t btid;

pthread\_t tid[MAX\_CUSTOMERS];

long RandSeed;

int i, numCustomers, numChairs;

int Number[MAX\_CUSTOMERS];

printf("Enter the number of Customers : "); scanf("%d",&numCustomers) ;

printf("Enter the number of Chairs : "); scanf("%d",&numChairs);

if (numCustomers > MAX\_CUSTOMERS) {

printf("The maximum number of Customers is %d.\n", MAX\_CUSTOMERS);

exit(-1);

}

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]);

sleep(1);

}

for (i=0; i<numCustomers; i++) {

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

sleep(1);

}

allDone = 1;

sem\_post(&barberPillow);

pthread\_join(btid,NULL);

}

void \*customer(void \*number);

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(2);

printf("The barber has finished cutting hair.\n");

sem\_post(&seatBelt);

}

else {

printf("The barber is going home for the day.\n");

}

}

}

void randwait(int secs) {

int len;

len = (int) ((1 \* secs) + 1);

sleep(len);

}

void \*customer(void \*number) {

int num = \*(int \*)number;

printf("Customer %d leaving for barber shop.\n", num);

randwait(2);

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);

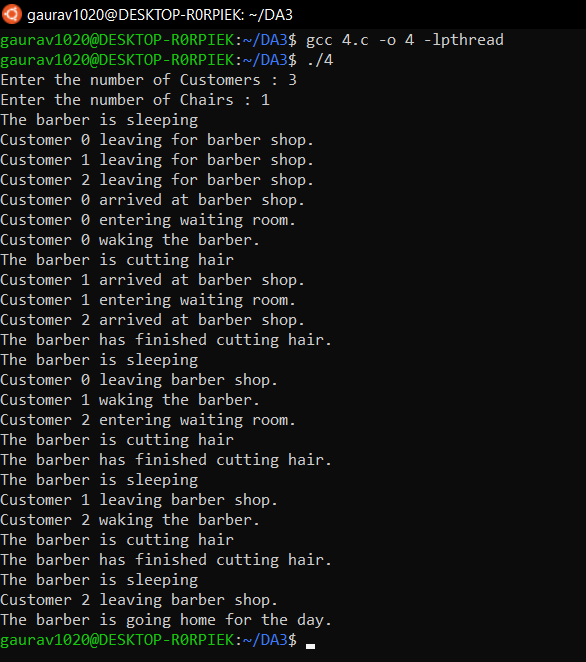
sem\_post(&barberPillow);

sem\_wait(&seatBelt);

sem\_post(&barberChair);

printf("Customer %d leaving barber shop.\n", num);

}



**(e) A pair of processes involved in exchanging a sequence of integers. The number of integers that can be produced and consumed at a time is limited to 100. Write a Program to implement the producer and consumer problem using POSIX semaphore for the above scenario.**

#include<stdio.h>

#include<semaphore.h>

#include<pthread.h>

#include<stdlib.h>

#define buffersize 100

pthread\_mutex\_t mutex;

pthread\_t tidP[100],tidC[100];

sem\_t full,empty;

int counter;

int buffer[buffersize];

void initialize()

{

pthread\_mutex\_init(&mutex,NULL);

sem\_init(&full,1,0);

sem\_init(&empty,1,buffersize);

counter=0;

}

void write(int item)

{

buffer[counter++]=item;

}

int read()

{

return(buffer[--counter]);

}

void \* producer (void \* param)

{

int waittime,item,i;

item=rand()%5;

waittime=rand()%5;

sem\_wait(&empty);pthread\_mutex\_lock(&mutex);

printf("\nProducer has produced item: %d\n",item);

write(item);

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

void \* consumer (void \* param)

{

int waittime,item;

waittime=rand()%5;

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

item=read();

printf("\nConsumer has consumed item: %d\n",item);

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

int main() {

int n1, n2, i;

initialize();

printf("\nEnter the no of producers: ");

scanf("%d", &n1);

printf("\nEnter the no of consumers: ");

scanf("%d", &n2);

for (i = 0; i < n1; i++)

pthread\_create(&tidP[i], NULL, producer, NULL);

for (i = 0; i < n2; i++)

pthread\_create(&tidC[i], NULL, consumer, NULL);

for (i = 0; i < n1; i++)

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

for (i = 0; i < n2; i++)

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

exit(0);

}

