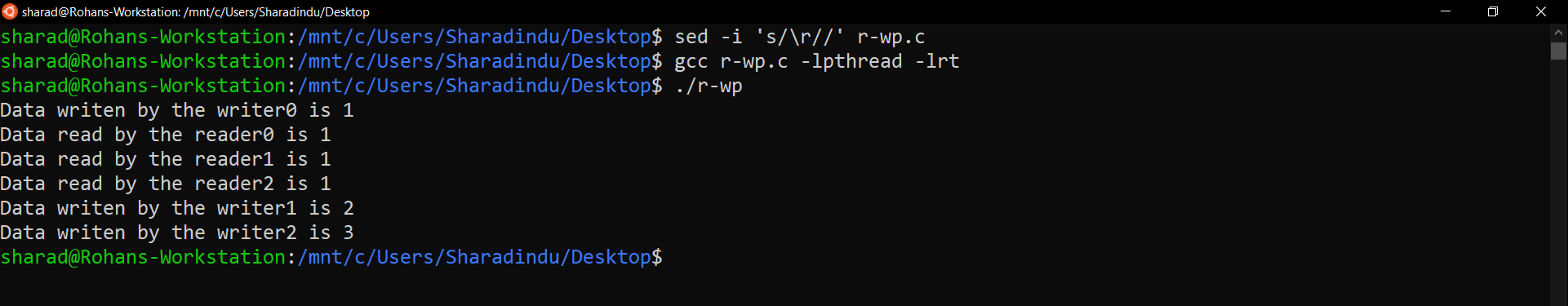
**OS Vtop Lab Assessment 3 Draft**

d/l Apr 28

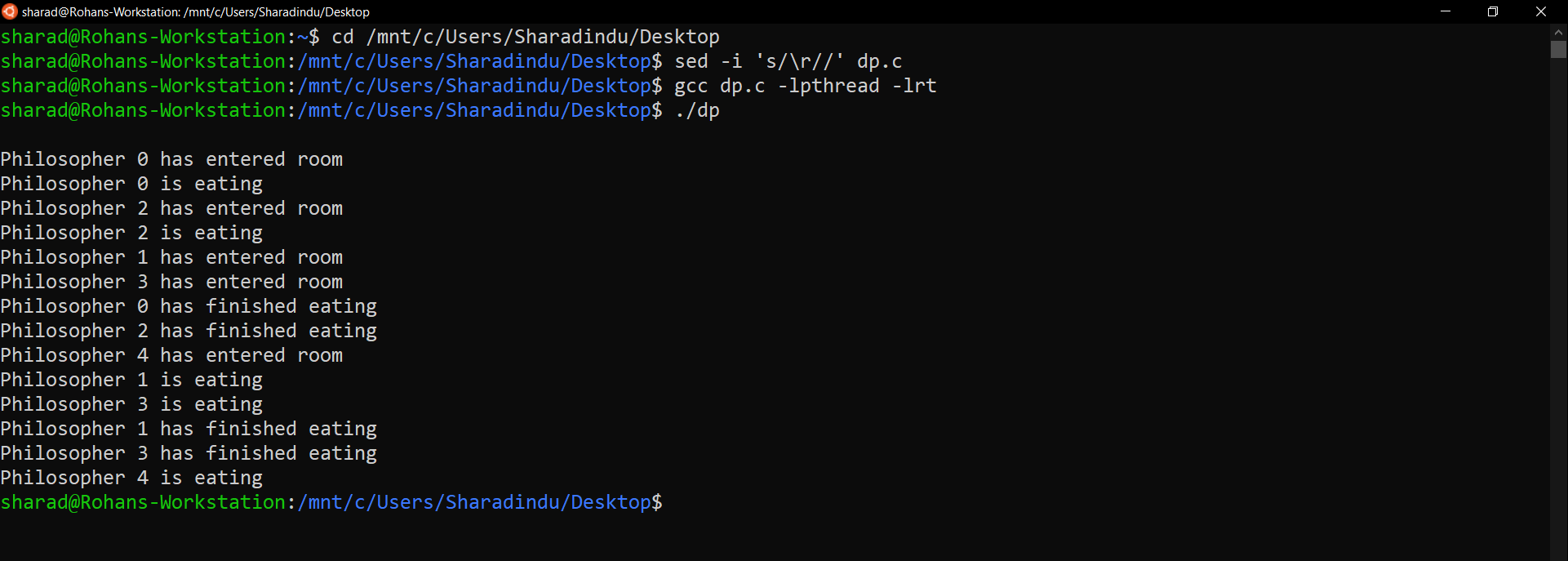
**(a) Implement the solution for reader – writer’s problem.**

#include<semaphore.h>  
#include<stdio.h>  
#include<stdlib.h>  
#include<unistd.h>  
#include<pthread.h>  
sem\_t x,y;  
pthread\_t tid;  
pthread\_t writerthreads[100],readerthreads[100];  
int readercount = 0;  
  
void \*reader(void\* param)  
{  
 sem\_wait(&x);  
 readercount++;  
 if(readercount==1)  
 sem\_wait(&y);  
 sem\_post(&x);  
 printf("%d reader is inside\n",readercount);  
 usleep(3);  
 sem\_wait(&x);  
 readercount--;  
 if(readercount==0)  
 {  
 sem\_post(&y);  
 }  
 sem\_post(&x);  
 printf("%d Reader is leaving\n",readercount+1);  
 return **NULL**;  
}  
  
void \*writer(void\* param)  
{  
 printf("Writer is trying to enter\n");  
 sem\_wait(&y);  
 printf("Writer has entered\n");  
 sem\_post(&y);  
 printf("Writer is leaving\n");  
 return **NULL**;  
}  
  
int main()  
{  
 int n2,i;  
 printf("Enter the number of readers:");  
 scanf("%d",&n2);  
 printf("\n");  
 int n1[n2];  
 sem\_init(&x,0,1);  
 sem\_init(&y,0,1);  
 for(i=0;i<n2;i++)  
 {  
 pthread\_create(&writerthreads[i],**NULL**,reader,**NULL**);  
 pthread\_create(&readerthreads[i],**NULL**,writer,**NULL**);  
 }  
 for(i=0;i<n2;i++)  
 {  
 pthread\_join(writerthreads[i],**NULL**);  
 pthread\_join(readerthreads[i],**NULL**);  
 }  
  
}



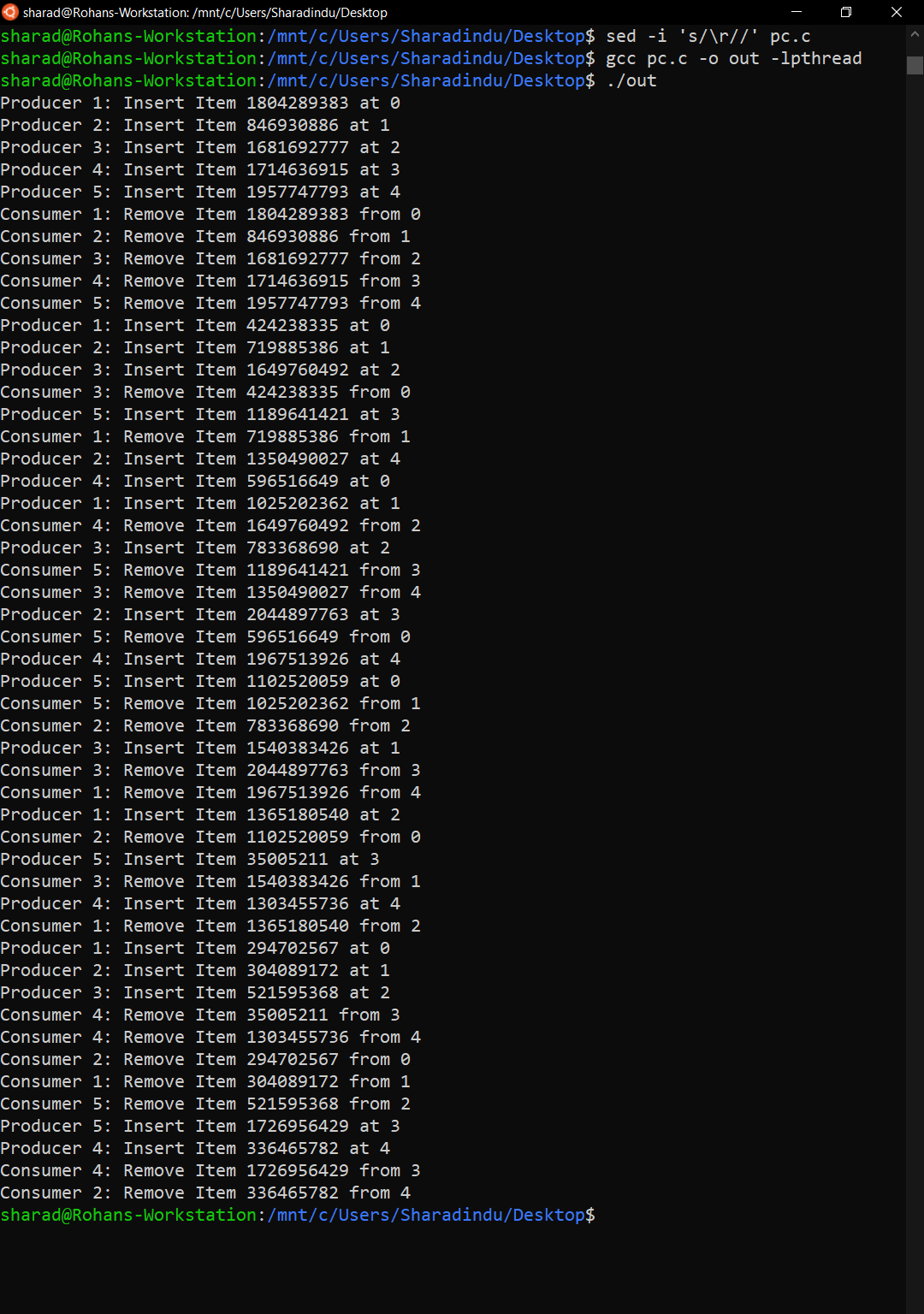
**(b) Implement the solution for dining philosopher’s problem.**

#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);  
}



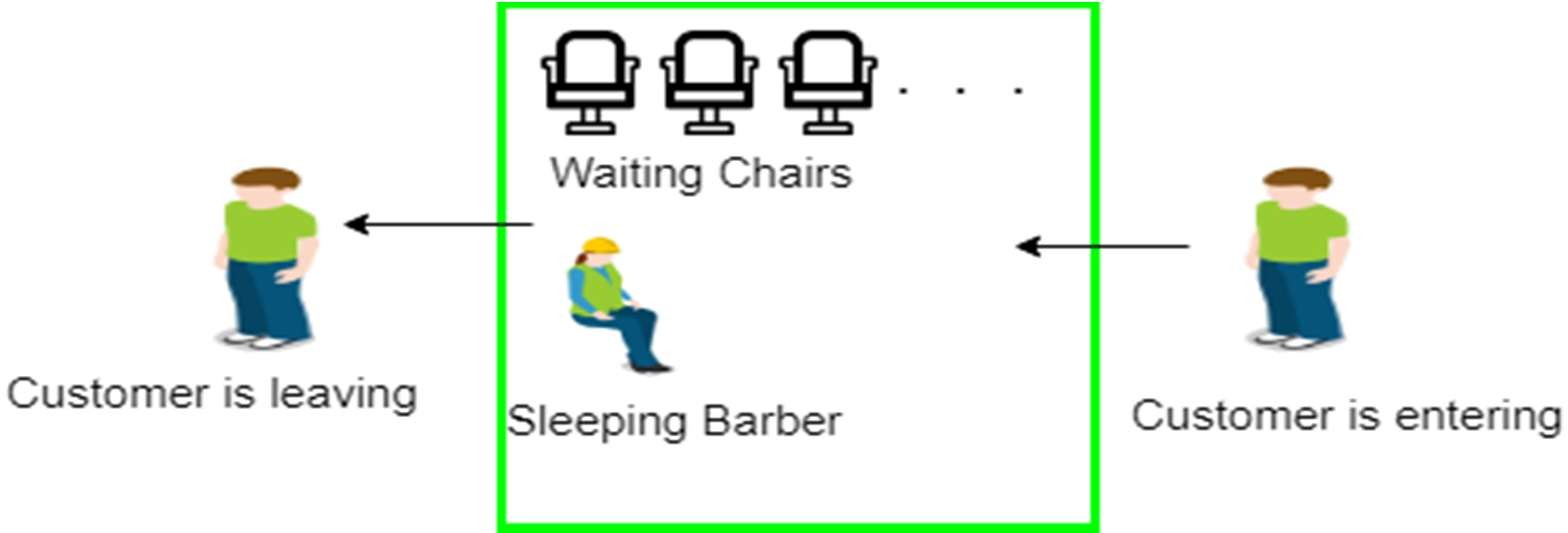
**(c) Implement the solution for producer consumer problem**

#include <pthread.h>  
#include <semaphore.h>  
#include <stdlib.h>  
#include <stdio.h>  
  
#define **MaxItems** 5 *// Maximum items a producer can produce or a consumer can consume*#define **BufferSize** 5 *// Size of the buffer*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;  
  
}

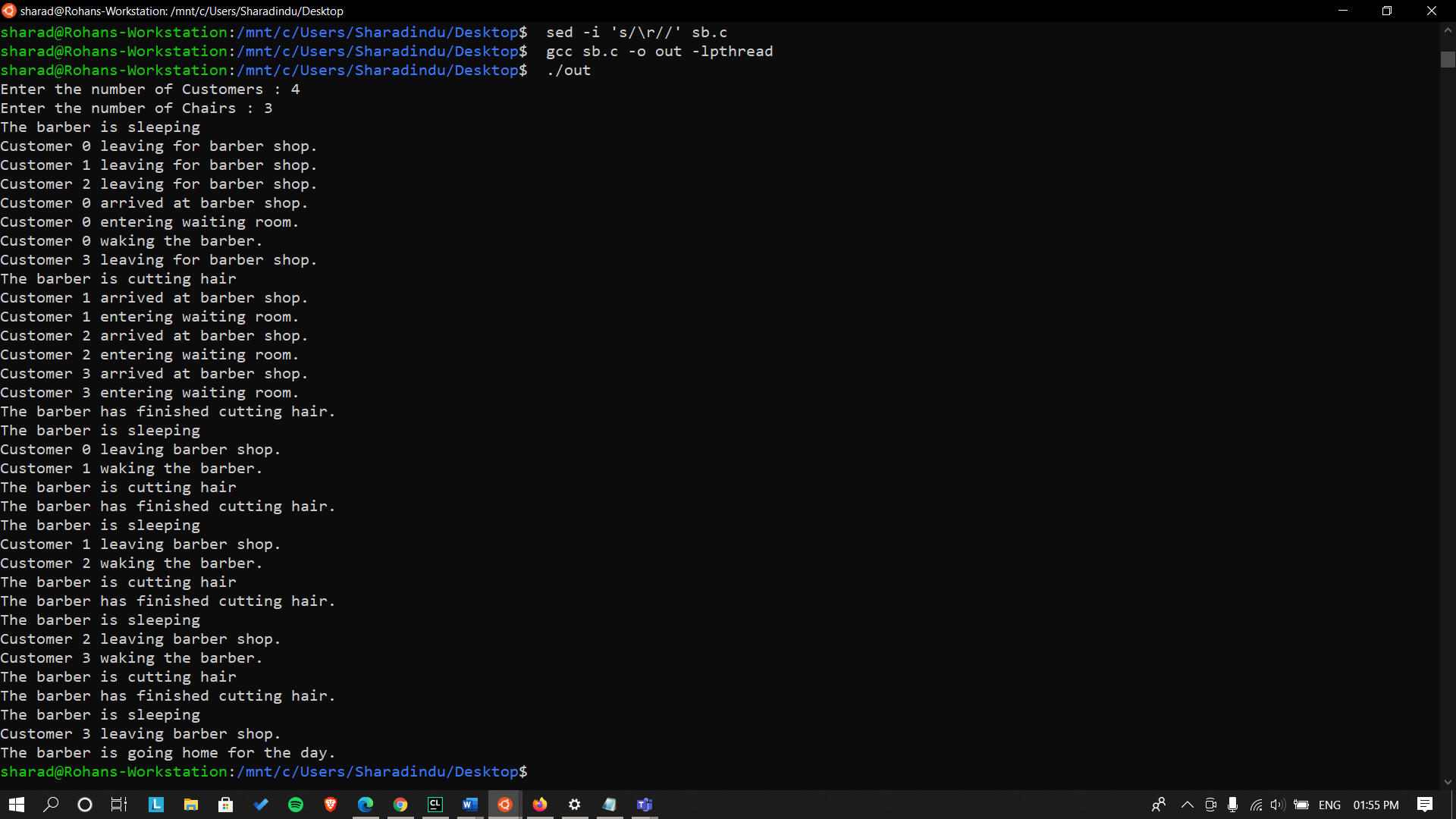


**(d) The analogy is based upon a hypothetical barber shop with one barber. There is a barber shop which has one barber, one barber chair, and n chairs for waiting for customers if there are any to sit on the chair.**

* + **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>  
*// The maximum number of customer threads.*#define **MAX\_CUSTOMERS** 25  
*// Function prototypes…*void \*customer(void \*num);  
void \*barber(void \*);  
void randwait(int secs);  
*// Define the semaphores.  
// waitingRoom Limits the # of customers allowed  
// to enter the waiting room at one time.*sem\_t waitingRoom;  
*// barberChair ensures mutually exclusive access to  
// the barber chair.*sem\_t barberChair;  
*// barberPillow is used to allow the barber to sleep  
// until a customer arrives.*sem\_t barberPillow;  
*// seatBelt is used to make the customer to wait until  
// the barber is done cutting his/her hair.*sem\_t seatBelt;  
*// Flag to stop the barber thread when all customers  
// have been serviced.*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);  
*// Make sure the number of threads is less than the number of  
// customers we can support.* if (numCustomers > **MAX\_CUSTOMERS**) {  
 printf("The maximum number of Customers is %d.\n", **MAX\_CUSTOMERS**);  
 exit(-1);  
 }  
*// Initialize the numbers array.* for (i=0; i<**MAX\_CUSTOMERS**; i++) {  
 Number[i] = i;  
 }  
*// Initialize the semaphores with initial values…* sem\_init(&waitingRoom, 0, numChairs);  
 sem\_init(&barberChair, 0, 1);  
 sem\_init(&barberPillow, 0, 0);  
 sem\_init(&seatBelt, 0, 0);  
*// Create the barber.* pthread\_create(&btid, **NULL**, barber, **NULL**);  
*// Create the customers.* for (i=0; i<numCustomers; i++) {  
 pthread\_create(&tid[i], **NULL**, customer, (void \*)&Number[i]);  
 sleep(1);  
 }  
*// Join each of the threads to wait for them to finish.* for (i=0; i<numCustomers; i++) {  
 pthread\_join(tid[i],**NULL**);  
 sleep(1);  
 }  
*// When all of the customers are finished, kill the  
// barber thread.* allDone = 1;  
 sem\_post(&barberPillow); *// Wake the barber so he will exit.* pthread\_join(btid,**NULL**);  
}  
void \*customer(void \*number);  
void \*barber(void \*junk) {  
*// While there are still customers to be serviced…  
// Our barber is omnicient and can tell if there are  
// customers still on the way to his shop.* while (!allDone) {  
*// Sleep until someone arrives and wakes you..* printf("The barber is sleeping\n");  
 sem\_wait(&barberPillow);  
*// Skip this stuff at the end…* if (!allDone) {  
*// Take a random amount of time to cut the  
// customer’s hair.* printf("The barber is cutting hair\n");  
 randwait(2);  
 printf("The barber has finished cutting hair.\n");  
*// Release the customer when done cutting…* sem\_post(&seatBelt);  
 }  
 else {  
 printf("The barber is going home for the day.\n");  
 }  
 }  
}  
void randwait(int secs) {  
 int len;  
*// Generate a random number…* len = (int) ((1 \* secs) + 1);  
 sleep(len);  
}  
  
void \*customer(void \*number) {  
 int num = \*(int \*)number;  
*// Leave for the shop and take some random amount of  
// time to arrive.* printf("Customer %d leaving for barber shop.\n", num);  
 randwait(2);  
 printf("Customer %d arrived at barber shop.\n", num);  
*// Wait for space to open up in the waiting room…* sem\_wait(&waitingRoom);  
 printf("Customer %d entering waiting room.\n", num);  
*// Wait for the barber chair to become free.* sem\_wait(&barberChair);  
*// The chair is free so give up your spot in the  
// waiting room.* sem\_post(&waitingRoom);  
*// Wake up the barber…* printf("Customer %d waking the barber.\n", num);  
 sem\_post(&barberPillow);  
*// Wait for the barber to finish cutting your hair.* sem\_wait(&seatBelt);  
*// Give up the chair.* 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**);  
*//sleep(5);* exit(0);  
}

