CSE 2005

OPERATING SYSTEMS

Assessment – 3

L7+L8 | PLBG17

WINTER SEMESTER 2020-21

by

SHARADINDU ADHIKARI

19BCE2105

Problems

Process Synchronization

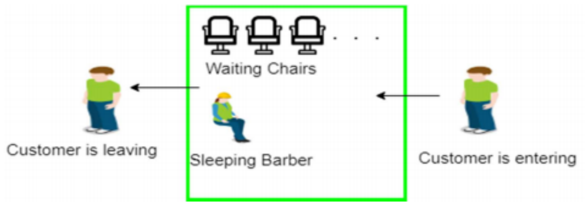
(a) Implement the solution for reader – writer's problem. **(Easy)**

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

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

(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.



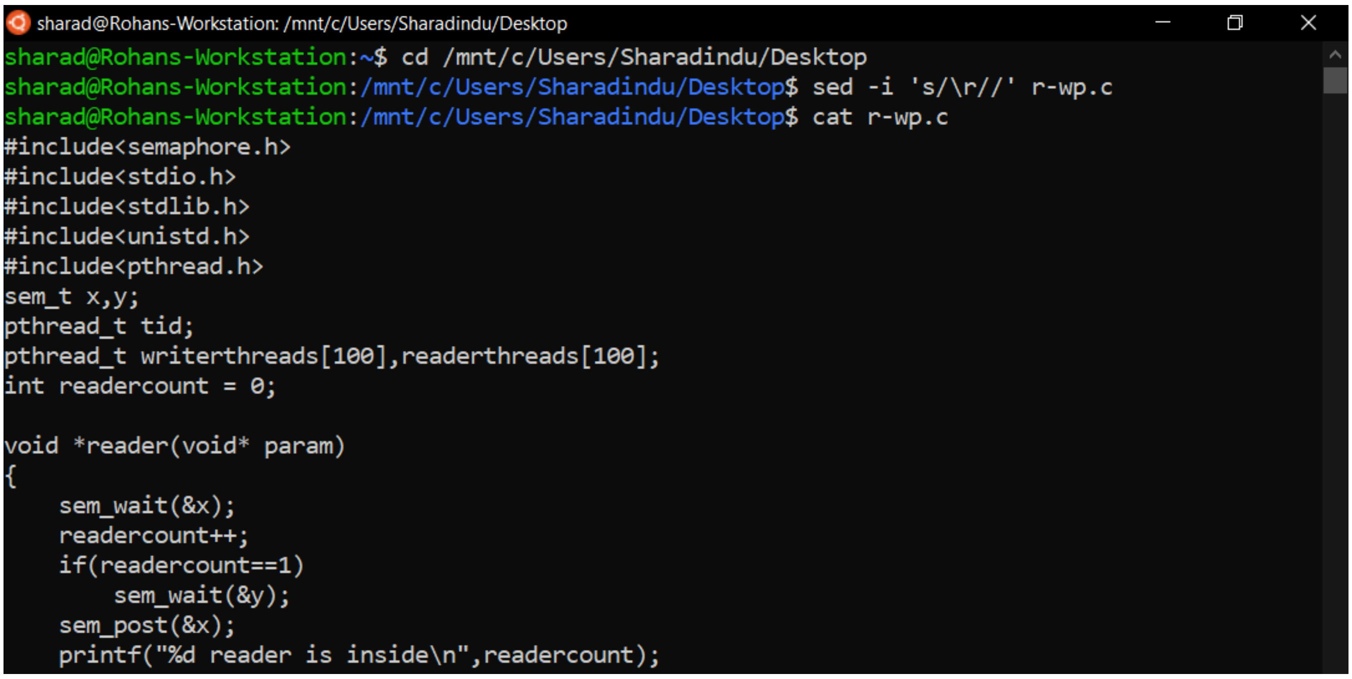
(Medium)

(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. **(Medium)**

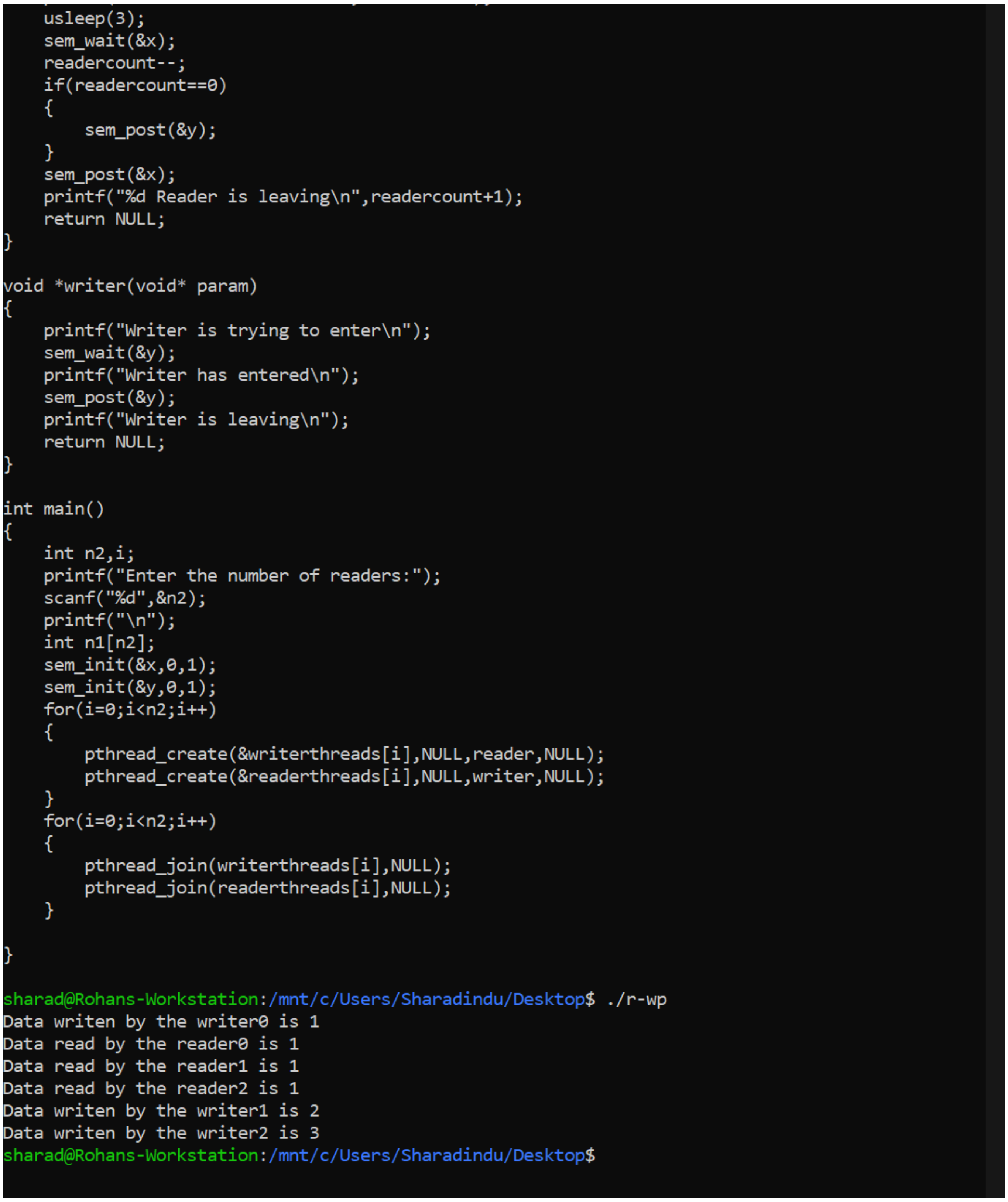
Solutions

(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);
    }

}
```



```
sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop
sharad@Rohans-Workstation:~$ cd /mnt/c/Users/Sharadindu/Desktop
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ sed -i 's/\r//' r-wp.c
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat r-wp.c
#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);
```

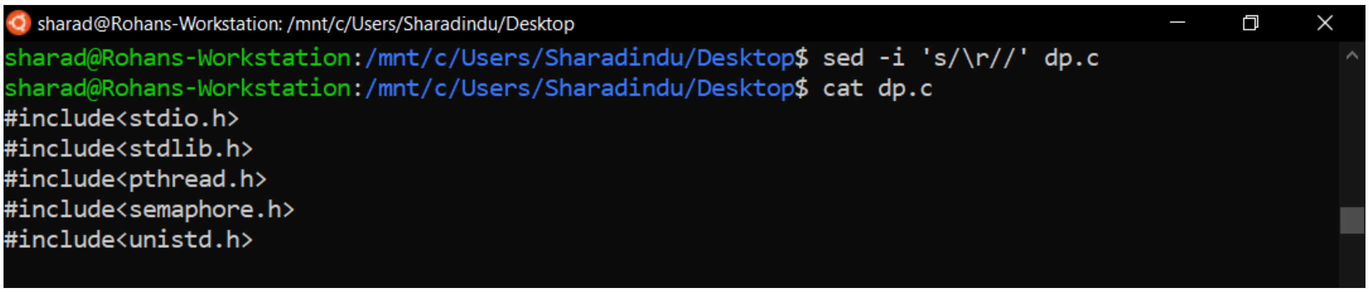


```
    usleep(3);
    sem_wait(&x);
    readercount--;
    if(readercount==0)
    {
        sem_post(&y);
    }
    sem_post(&x);
    printf("%d Reader is leaving\n",readercount+1);
    return NULL;
}

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);
    }

}
```

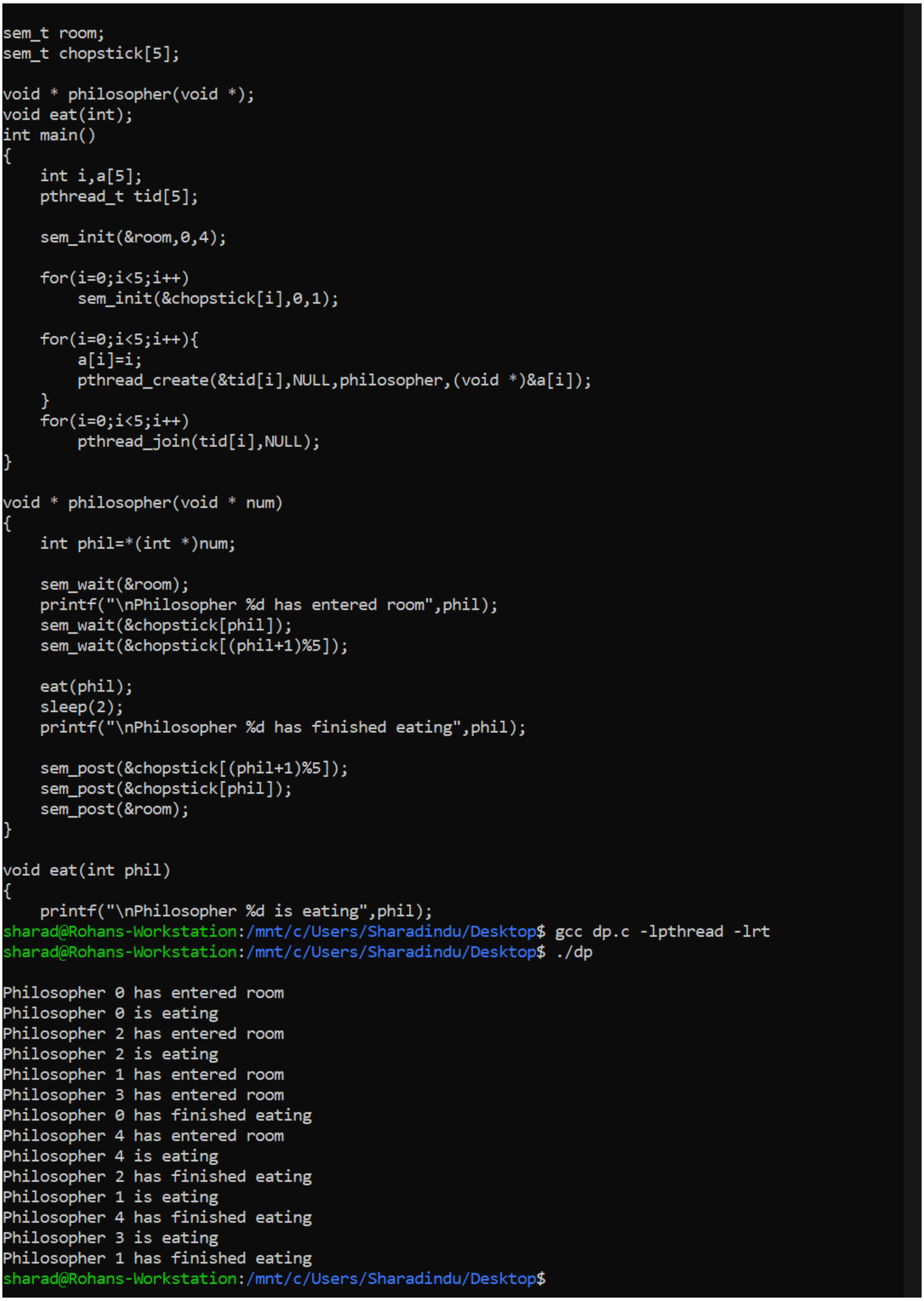
```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./r-wp
Data writen by the writer0 is 1
Data read by the reader0 is 1
Data read by the reader1 is 1
Data read by the reader2 is 1
Data writen by the writer1 is 2
Data writen by the writer2 is 3
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

(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);
}
```


sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

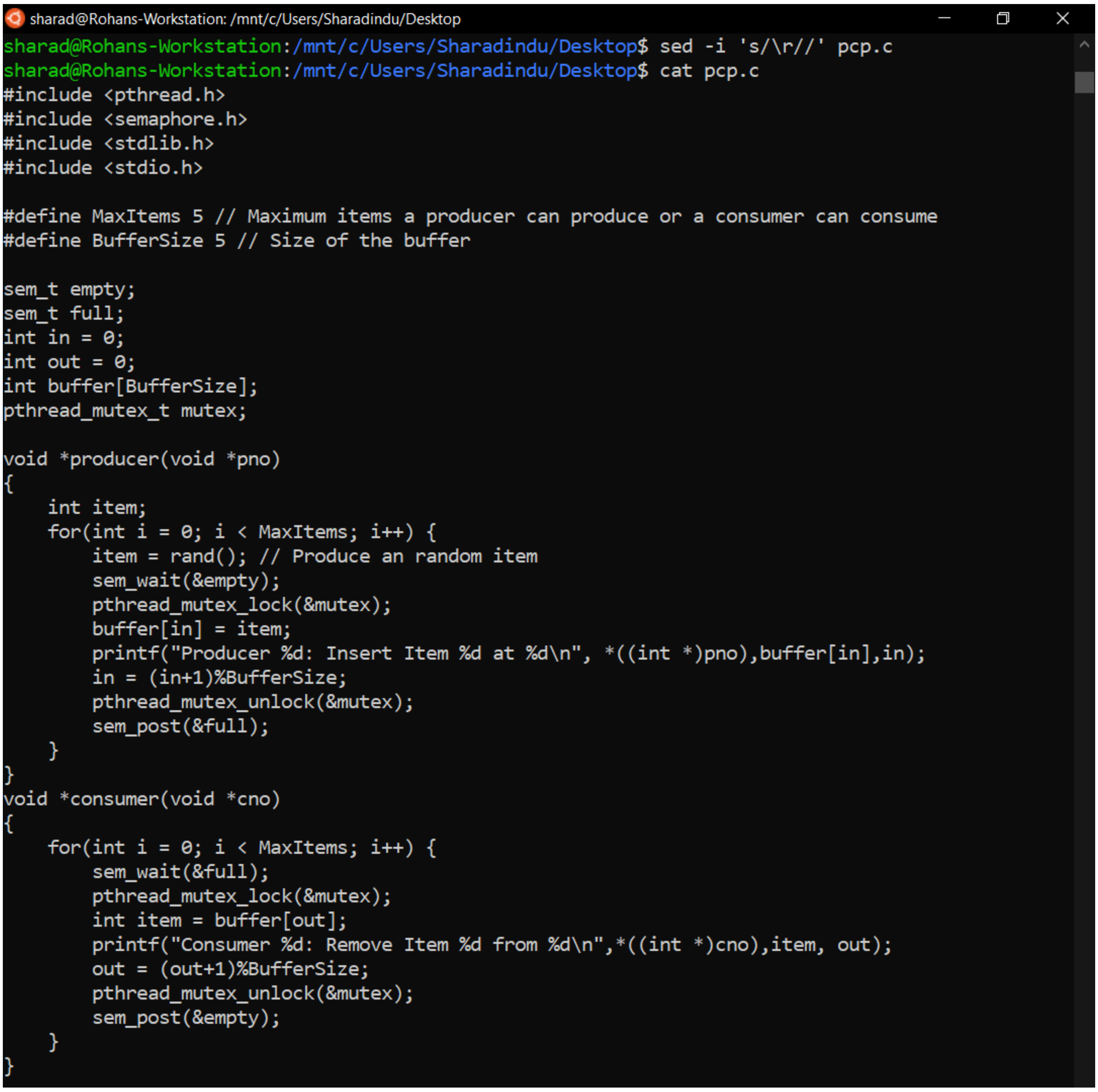
```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ sed -i 's/\r//' dp.c
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat dp.c
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
#include<unistd.h>
```



```
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);
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc dp.c -lpthread -lrt
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./dp

Philosopher 0 has entered room
Philosopher 0 is eating
Philosopher 2 has entered room
Philosopher 2 is eating
Philosopher 1 has entered room
Philosopher 3 has entered room
Philosopher 0 has finished eating
Philosopher 4 has entered room
Philosopher 4 is eating
Philosopher 2 has finished eating
Philosopher 1 is eating
Philosopher 4 has finished eating
Philosopher 3 is eating
Philosopher 1 has finished eating
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

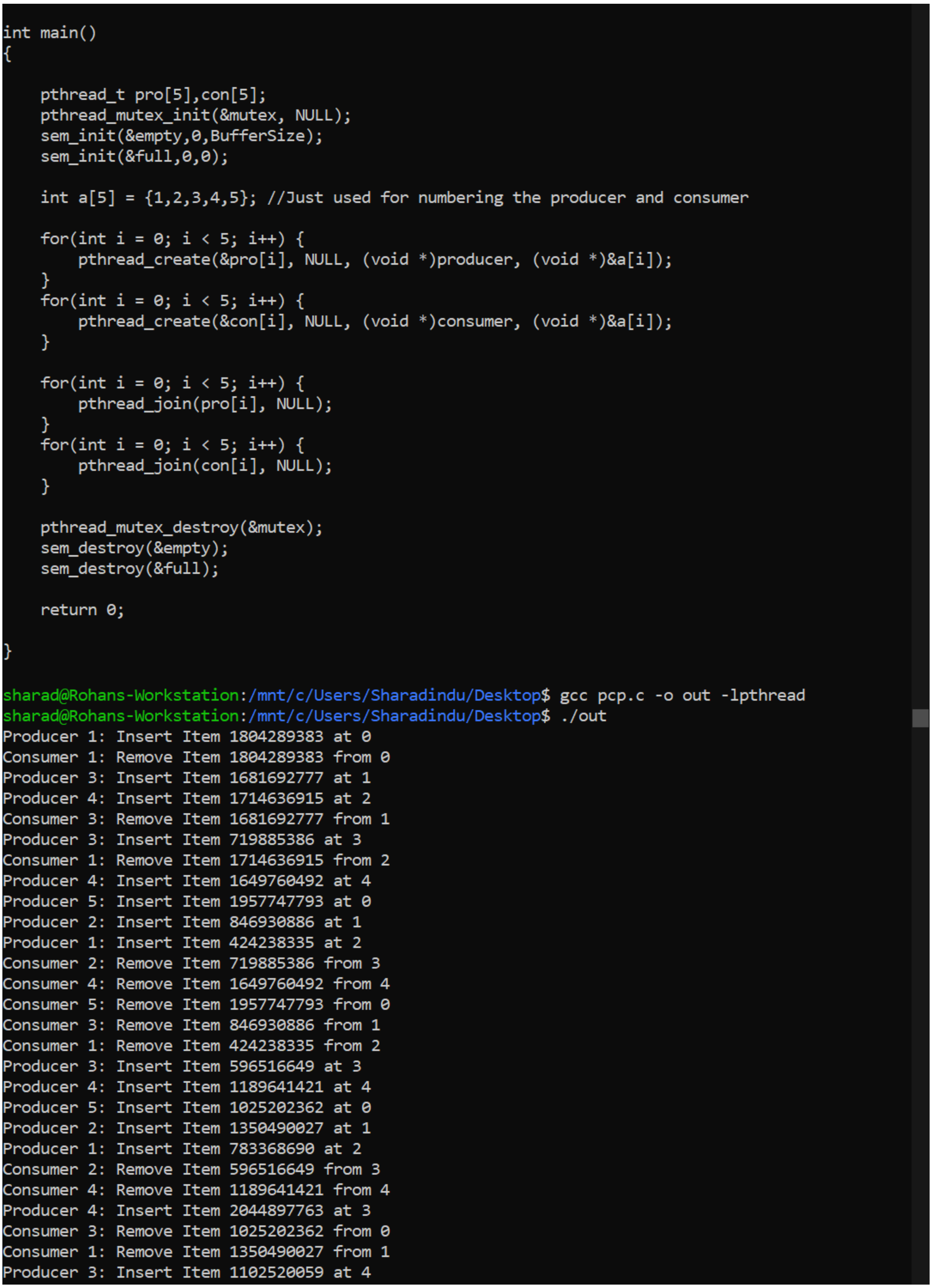
(c) Implement the solution for producer consumer problem.

```
#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;

}
```



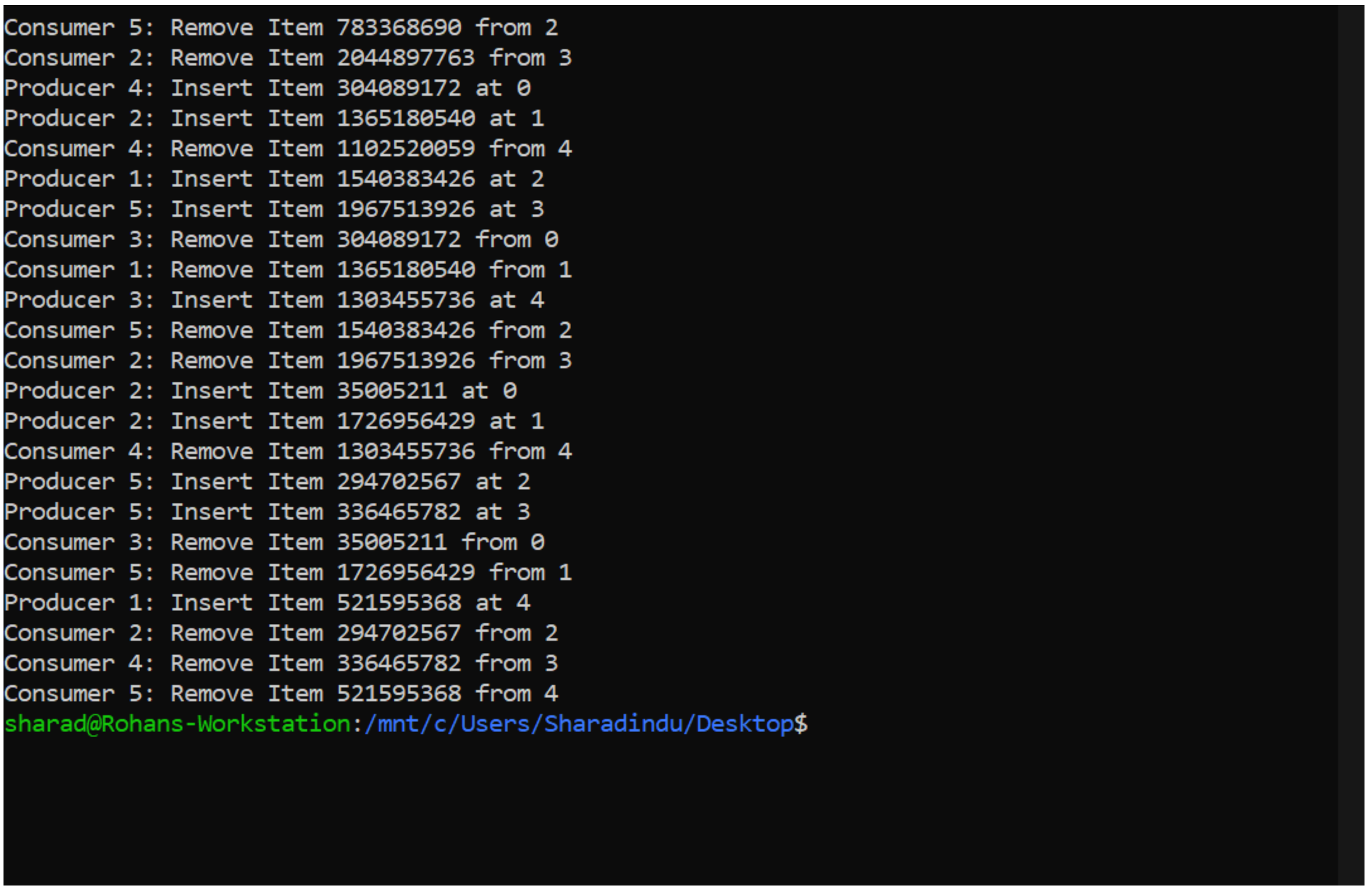
sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ sed -i 's/\r//' pcp.c
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat pcp.c
#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;

}
```

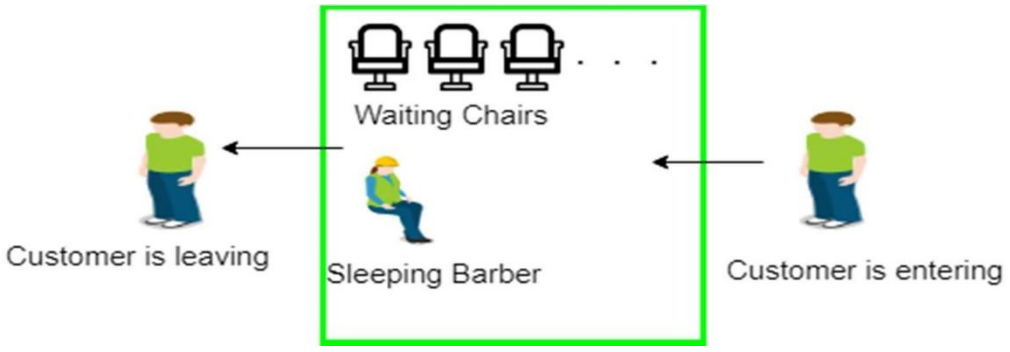
```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc pcp.c -o out -lpthread
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./out
Producer 1: Insert Item 1804289383 at 0
Consumer 1: Remove Item 1804289383 from 0
Producer 3: Insert Item 1681692777 at 1
Producer 4: Insert Item 1714636915 at 2
Consumer 3: Remove Item 1681692777 from 1
Producer 3: Insert Item 719885386 at 3
Consumer 1: Remove Item 1714636915 from 2
Producer 4: Insert Item 1649760492 at 4
Producer 5: Insert Item 1957747793 at 0
Producer 2: Insert Item 846930886 at 1
Producer 1: Insert Item 424238335 at 2
Consumer 2: Remove Item 719885386 from 3
Consumer 4: Remove Item 1649760492 from 4
Consumer 5: Remove Item 1957747793 from 0
Consumer 3: Remove Item 846930886 from 1
Consumer 1: Remove Item 424238335 from 2
Producer 3: Insert Item 596516649 at 3
Producer 4: Insert Item 1189641421 at 4
Producer 5: Insert Item 1025202362 at 0
Producer 2: Insert Item 1350490027 at 1
Producer 1: Insert Item 783368690 at 2
Consumer 2: Remove Item 596516649 from 3
Consumer 4: Remove Item 1189641421 from 4
Producer 4: Insert Item 2044897763 at 3
Consumer 3: Remove Item 1025202362 from 0
Consumer 1: Remove Item 1350490027 from 1
Producer 3: Insert Item 1102520059 at 4
```



```
Consumer 5: Remove Item 783368690 from 2
Consumer 2: Remove Item 2044897763 from 3
Producer 4: Insert Item 304089172 at 0
Producer 2: Insert Item 1365180540 at 1
Consumer 4: Remove Item 1102520059 from 4
Producer 1: Insert Item 1540383426 at 2
Producer 5: Insert Item 1967513926 at 3
Consumer 3: Remove Item 304089172 from 0
Consumer 1: Remove Item 1365180540 from 1
Producer 3: Insert Item 1303455736 at 4
Consumer 5: Remove Item 1540383426 from 2
Consumer 2: Remove Item 1967513926 from 3
Producer 2: Insert Item 35005211 at 0
Producer 2: Insert Item 1726956429 at 1
Consumer 4: Remove Item 1303455736 from 4
Producer 5: Insert Item 294702567 at 2
Producer 5: Insert Item 336465782 at 3
Consumer 3: Remove Item 35005211 from 0
Consumer 5: Remove Item 1726956429 from 1
Producer 1: Insert Item 521595368 at 4
Consumer 2: Remove Item 294702567 from 2
Consumer 4: Remove Item 336465782 from 3
Consumer 5: Remove Item 521595368 from 4
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

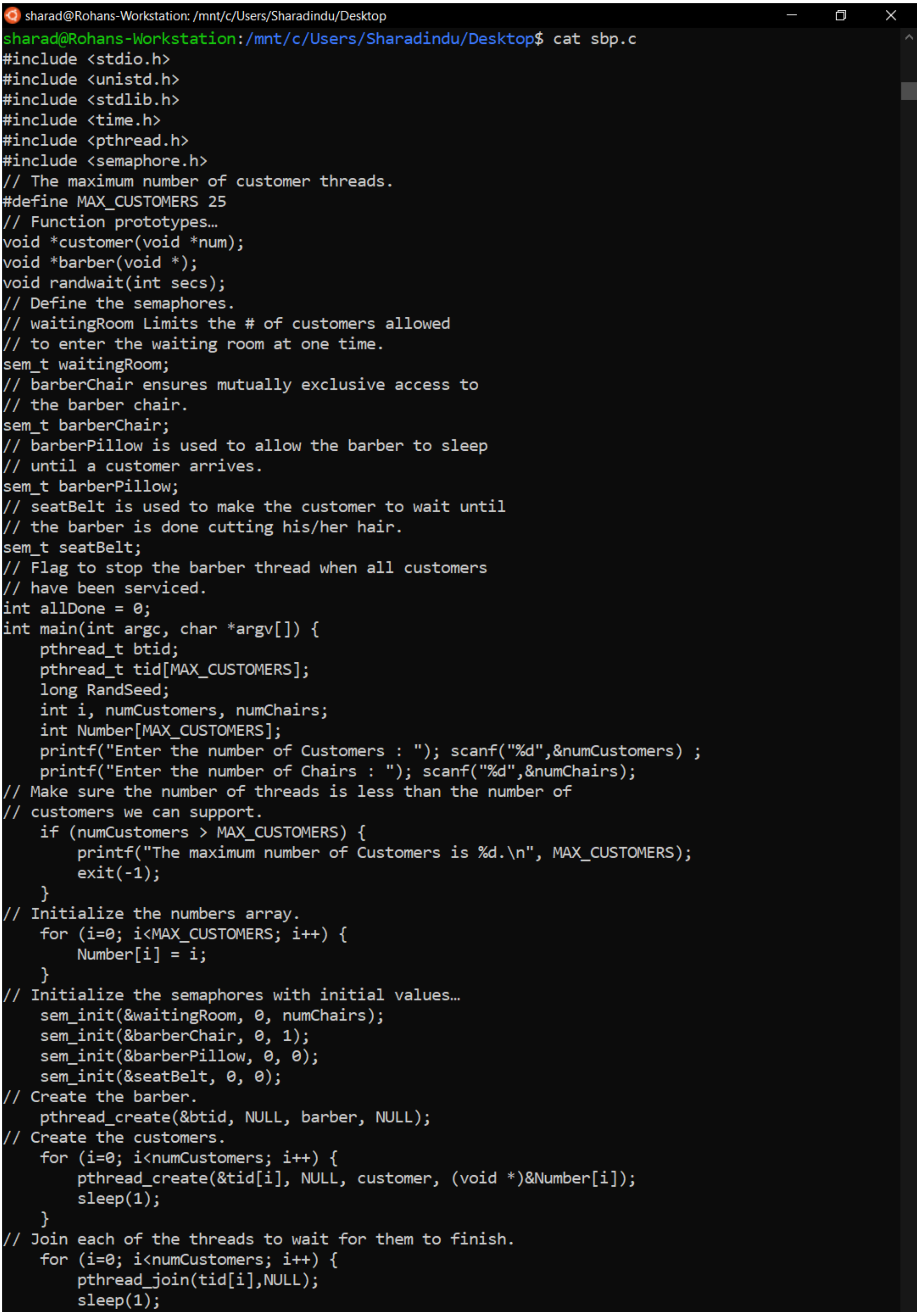
(d) The analogy is based upon a hypothetical barber shop with one barber. There is a barber shop which has one barber, one barberchair, and n chairs for waiting for customers if there are any to sit on the chair.

- **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'shair, then the remaining customers either wait if there are empty chairs in the waiting room or they leave if no chairs are empty.**



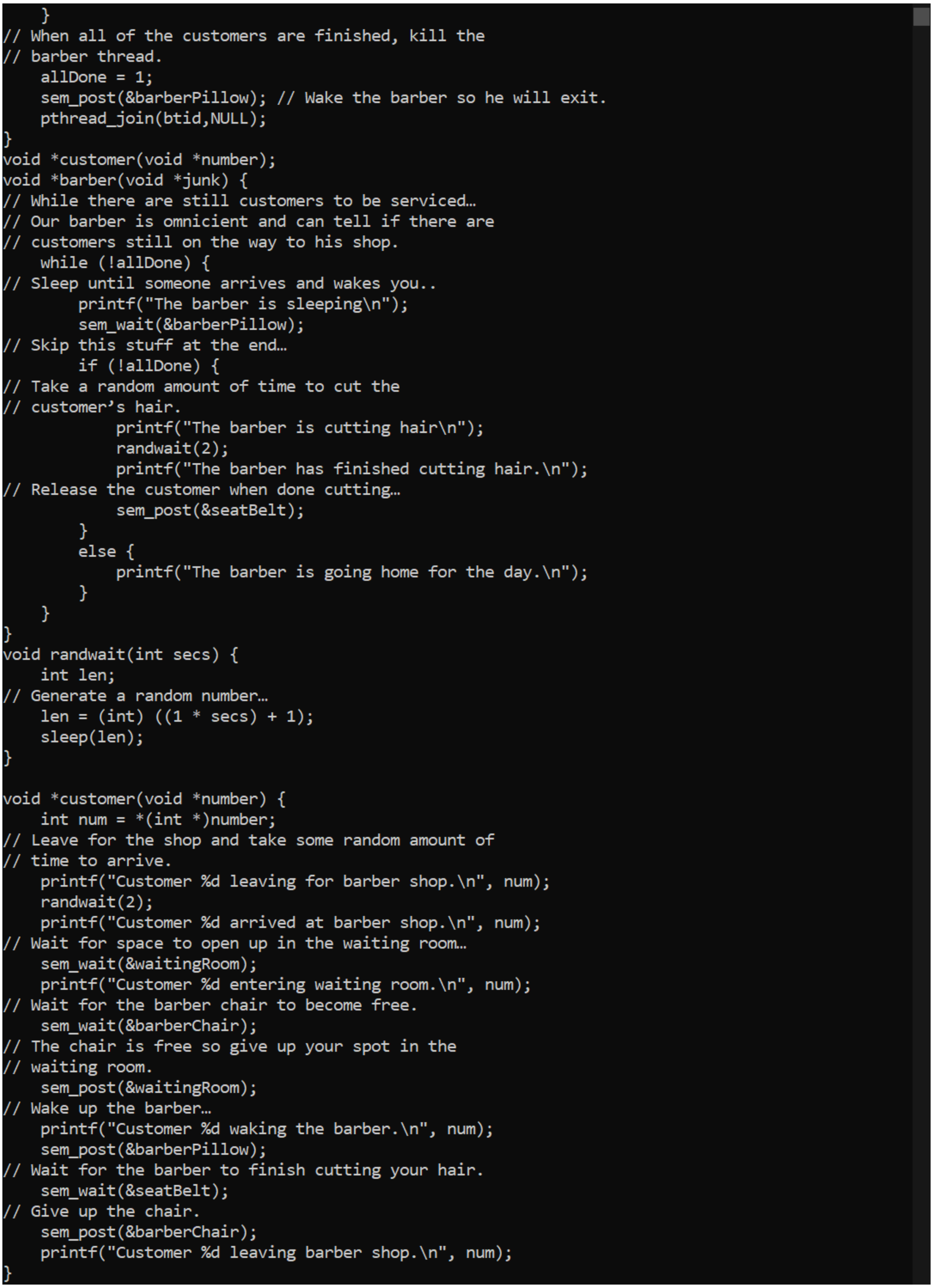
```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <semaphore.h>
// The maximum number of customer threads.
#define MAX_CUSTOMERS 25
// Function prototypes…
void *customer(void *num);
void *barber(void *);
void randwait(int secs);
// Define 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) {
    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);
}
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);
}
```

sharadindu.adhikari2019@vitstudent.ac.in

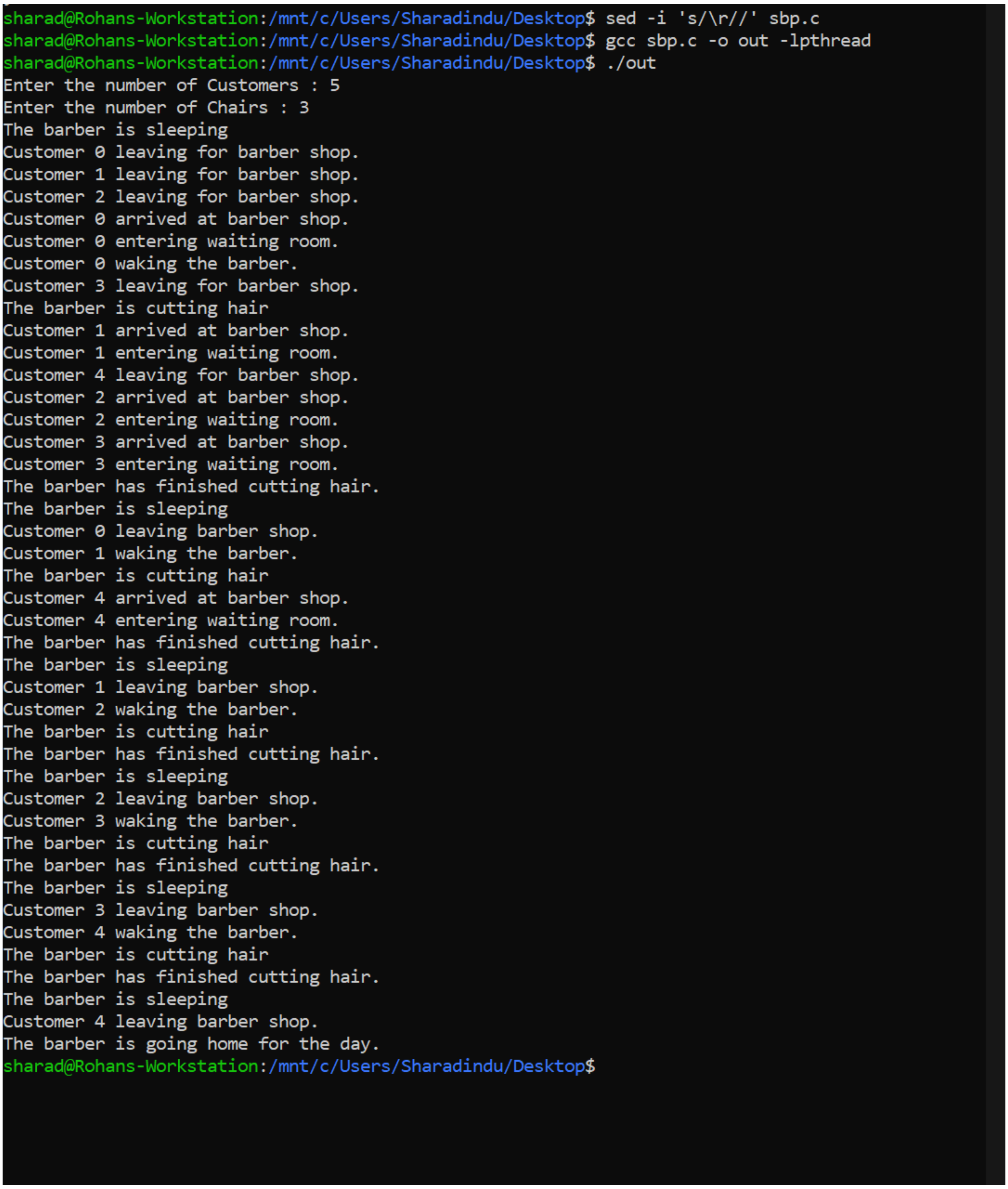


sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat sbp.c
#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);
```



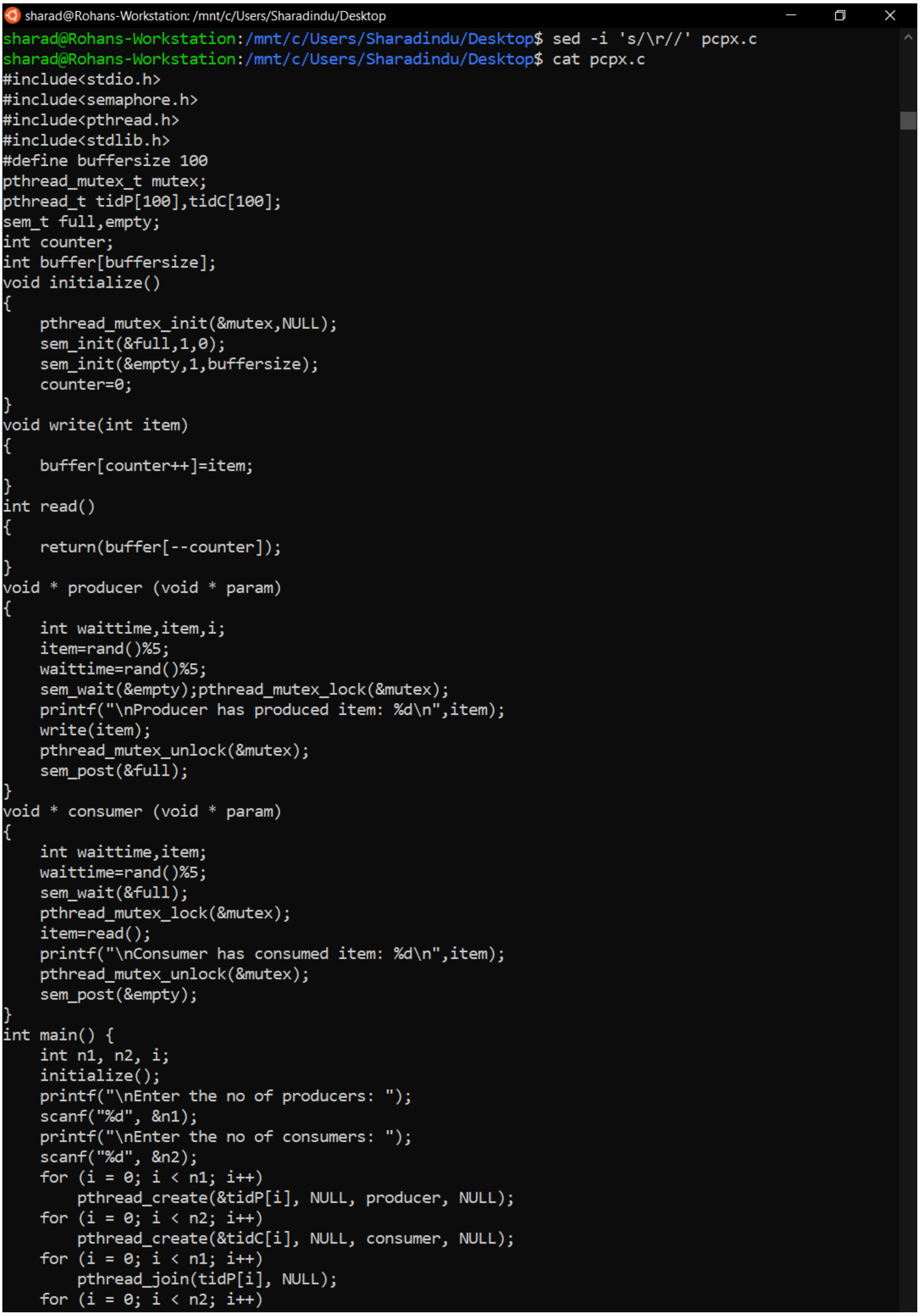
```
    }
// When all of the customers are finished, kill the
// barber thread.
    allDone = 1;
    sem_post(&barberPillow); // Wake the barber so he will exit.
    pthread_join(btid,NULL);
}
void *customer(void *number);
void *barber(void *junk) {
// While there are still customers to be serviced…
// 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);
}
```



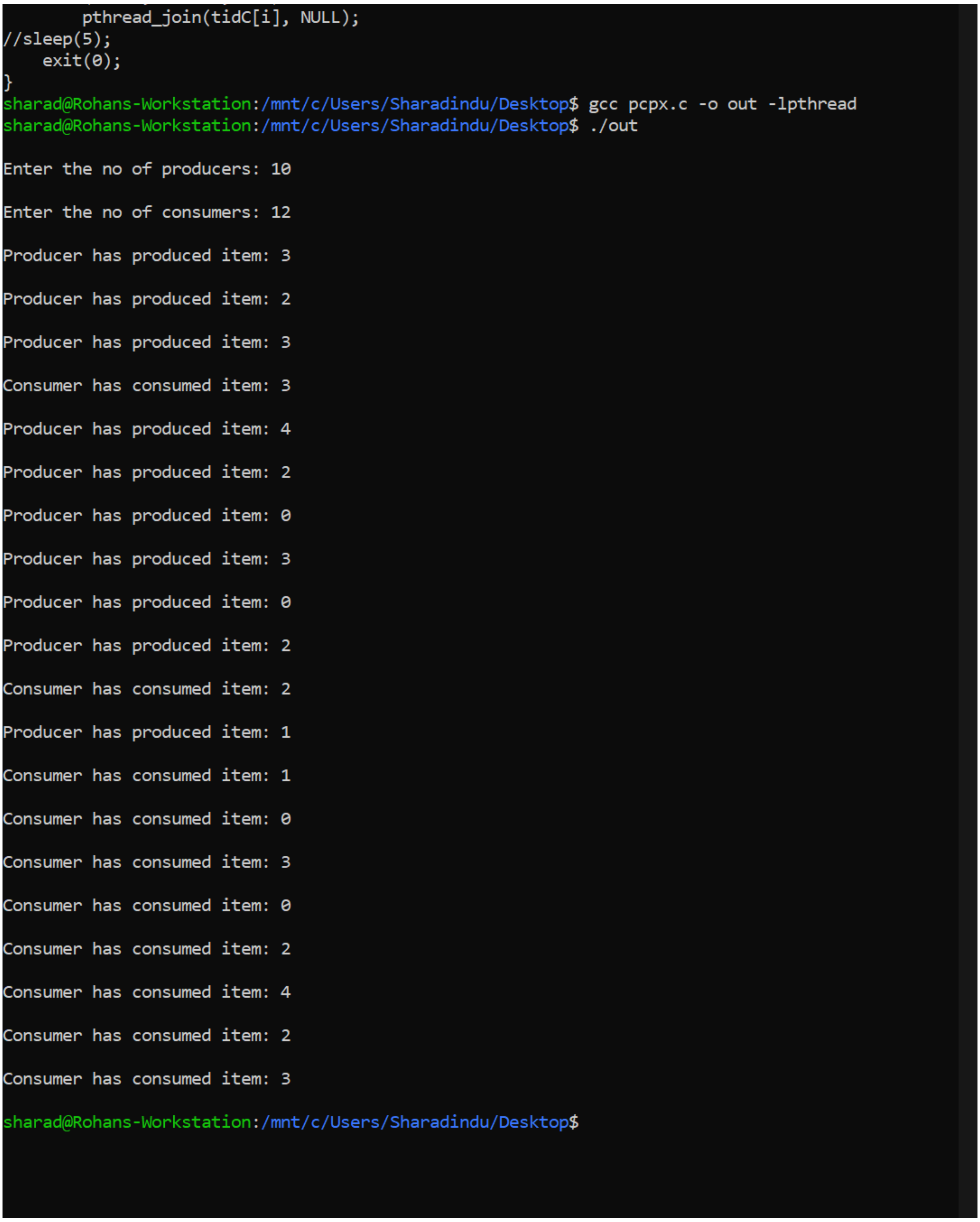
```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ sed -i 's/\r//' sbp.c
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc sbp.c -o out -lpthread
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./out
Enter the number of Customers : 5
Enter the number of Chairs : 3
The barber is sleeping
Customer 0 leaving for barber shop.
Customer 1 leaving for barber shop.
Customer 2 leaving for barber shop.
Customer 0 arrived at barber shop.
Customer 0 entering waiting room.
Customer 0 waking the barber.
Customer 3 leaving for barber shop.
The barber is cutting hair
Customer 1 arrived at barber shop.
Customer 1 entering waiting room.
Customer 4 leaving for barber shop.
Customer 2 arrived at barber shop.
Customer 2 entering waiting room.
Customer 3 arrived at barber shop.
Customer 3 entering waiting room.
The barber has finished cutting hair.
The barber is sleeping
Customer 0 leaving barber shop.
Customer 1 waking the barber.
The barber is cutting hair
Customer 4 arrived at barber shop.
Customer 4 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
The barber has finished cutting hair.
The barber is sleeping
Customer 2 leaving barber shop.
Customer 3 waking the barber.
The barber is cutting hair
The barber has finished cutting hair.
The barber is sleeping
Customer 3 leaving barber shop.
Customer 4 waking the barber.
The barber is cutting hair
The barber has finished cutting hair.
The barber is sleeping
Customer 4 leaving barber shop.
The barber is going home for the day.
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

(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 islimited 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);
}
```


sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ sed -i 's/\r//' pcpx.c
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat pcpx.c
#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);
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc pcpx.c -o out -lpthread
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./out

Enter the no of producers: 10

Enter the no of consumers: 12

Producer has produced item: 3

Producer has produced item: 2

Producer has produced item: 3

Consumer has consumed item: 3

Producer has produced item: 4

Producer has produced item: 2

Producer has produced item: 0

Producer has produced item: 3

Producer has produced item: 0

Producer has produced item: 2

Consumer has consumed item: 2

Producer has produced item: 1

Consumer has consumed item: 1

Consumer has consumed item: 0

Consumer has consumed item: 3

Consumer has consumed item: 0

Consumer has consumed item: 2

Consumer has consumed item: 4

Consumer has consumed item: 2

Consumer has consumed item: 3

sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```