SLEEPING BARBER PROBLEM

A

Mini Project Report

Submitted for the course

Operating Systems Lab

Semester-IV, CSE-B

By

B. AKHIL1602-18-733-063

L. SACHIN1602-18-733-094

****

**Department of Computer Science and Engineering**

**VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)**

**9-5-81, Ibrahimbagh, Hyderabad-31**

**2019-20**

**ACKNOWLEDGEMENT**

We take this opportunity with pride and enormous gratitude, to express the deeply embedded feeling and gratefulness to our respectable guide **Mr. R Sateesh Kumar,** Department of Computer Science and Engineering. Whose guidance was unforgettable and innovative ideas as well as his constructive suggestions has made the presentation of my thesis a grand success.

We are thankful to **Dr. T. Adilakshmi,** Head of Department (CSE), **Vasavi College of Engineering** for their help during our course work.

Finally, at last but not least express our heart full thanks to the management of our college, **Vasavi College of Engineering** for providing the necessary arrangements and support to complete my seminar work successively.

We convey our heartful thanks to our external guide **B. Bapu Reddy** who has helped us throughout our project work.

**ABSTRACT**

Topic: Sleeping Barber problem in Process Synchronization

Prerequisite – [Inter Process Communication](https://www.geeksforgeeks.org/inter-process-communication/)

**Problem:** 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.

**Solutions:**

Many possible solutions are available. The key element of each is a mutex, which ensures that only one of the participants can change state at once. The barber must acquire/enforce this mutual exclusion (of room status) before checking for customers and release it when they begin either to sleep or cut hair. A customer must acquire it before entering the shop and release it once they are sitting in either a waiting room chair or the barber chair, and also when they leave the shop because no seats were available. This eliminates both of the problems mentioned in the previous section. A number of [semaphores](https://en.wikipedia.org/wiki/Semaphore_(programming)) is also required to indicate the state of the system. For example, one might store the number of people in the waiting room.

A multiple sleeping barbers problem has the additional complexity of coordinating several barbers among the waiting customers.

**TABLE OF CONTENTS**

[1.INTRODUCTION 1](#_Toc40291603)

[2. PROBLEM DESCRIPTION 2](#_Toc40291604)

[3. PREREQUISITE 3](#_Toc40291605)

[3.1 Inter Process Communication: 3](#_Toc40291606)

[3.2 Semaphores in Process Synchronization 4](#_Toc40291607)

[4. SOLUTION 5](#_Toc40291608)

[5. PSEUDO ALGORITHM 6](#_Toc40291609)

[6. IMPLEMENTATION 8](#_Toc40291610)

[7. RESULTS 15](#_Toc40291611)

[7.1 Output 1 15](#_Toc40291612)

[7.2 Output 2 16](#_Toc40291613)

[7.3 Output 3 17](#_Toc40291614)

[7.4 Output 4 18](#_Toc40291615)

[7.5 Output 5 19](#_Toc40291616)

[7.6 Output 6 20](#_Toc40291617)

[8. REFERENCES 21](#_Toc40291618)

# 

# INTRODUCTION

Another classical IPC problem takes place in a barber shop. The barber shop has one barber, one barber chair, and n chairs for waiting customers, if any, to sit on. If there are no customers present, the barber sits down in the barber chair and falls asleep. When a customer arrives, he has to wake up the sleeping barber. If additional customers arrive while the barber is cutting a customer's hair, they either sit down (if there are empty chairs) or leave the shop (if all chairs are full). The problem is to program the barber and the customers without getting into race conditions. This problem is similar to various queueing situations, such as a multi person helpdesk with a computerized call waiting system for holding a limited number of incoming calls. Our solution uses three semaphores, customers, which counts waiting customers (excluding the customer in the barber chair, who is not waiting), barbers, the number of barbers (0 or 1) who are idle, waiting for customers, and mutex, which is used for mutual exclusion. We also need a variable, waiting, which also counts the waiting customers. The reason for having waiting is that there is no way to read the current value of a semaphore.

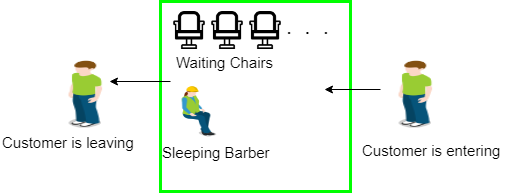
In this solution, a customer entering the shop has to count the number of waiting customers. If it is less than the number of chairs, he stays; otherwise, he leaves. When the barber shows up for work in the morning, he executes the procedure barber, causing him to block on the semaphore customers because it is initially 0. The barber then goes to sleep, He stays asleep until the first customer shows up. When a customer arrives, he executes customer, starting by acquiring mutex to enter a critical region. If another customer enters shortly thereafter, the second one will no be able to do anything until the first one has released mutex. The customer then checks to see if the number of waiting customers is less than the number of chairs. If not, he releases mutex and leaves without a haircut. If there is an available chair, the customer increments the integer variable, waiting. Then he does an Up on the semaphore customers, thus waking up the barber. At this point, the customer and the barber are both awake. When the customer releases mutex, the barber grabs it, does some housekeeping, and begins the haircut.

# 

# 2. PROBLEM DESCRIPTION

 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.



# 

# 3. PREREQUISITE

## 3.1 Inter Process Communication:

A process can be of two type:

* Independent process.
* Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently but in practical, there are many situations when co-operative nature can be utilised for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message passing

# 

## 3.2 Semaphores in Process Synchronization

Semaphore was proposed by Dijkstra in 1965 which is a very significant technique to manage concurrent processes by using a simple integer value, which is known as a semaphore. Semaphore is simply a variable which is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.

Semaphores are of two types:

1. **Binary Semaphore –** This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problem with multiple processes.
2. **Counting Semaphore –** Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

# 4. SOLUTION

The solution to this problem includes three [semaphores](https://www.geeksforgeeks.org/semaphores-operating-system/). First is for the customer which counts the number of customers present in the waiting room (customer in the barber chair is not included because he is not waiting). Second, the barber 0 or 1 is used to tell whether the barber is idle or is working, And the third mutex is used to provide the mutual exclusion which is required for the process to execute. In the solution, the customer has the record of the number of customers waiting in the waiting room if the number of customers is equal to the number of chairs in the waiting room then the upcoming customer leaves the barbershop.

When the barber shows up in the morning, he executes the procedure barber, causing him to block on the semaphore customers because it is initially 0. Then the barber goes to sleep until the first customer comes up.

When a customer arrives, he executes customer procedure the customer acquires the mutex for entering the critical region, if another customer enters thereafter, the second one will not be able to anything until the first one has released the mutex. The customer then checks the chairs in the waiting room if waiting customers are less then the number of chairs then he sits otherwise he leaves and releases the mutex.

If the chair is available then customer sits in the waiting room and increments the variable waiting value and also increases the customer’s semaphore this wakes up the barber if he is sleeping.

At this point, customer and barber are both awake and the barber is ready to give that person a haircut. When the haircut is over, the customer exits the procedure and if there are no customers in waiting room barber sleeps.

# 5. PSEUDO ALGORITHM

Semaphore Customers = 0;

Semaphore Barber = 0;

Mutex Seats = 1;

int FreeSeats = N;

Barber {

while(true) {

/\* waits for a customer (sleeps). \*/

down(Customers);

/\* mutex to protect the number of available seats.\*/

down(Seats);

/\* a chair gets free.\*/

FreeSeats++;

/\* bring customer for haircut.\*/

up(Barber);

/\* release the mutex on the chair.\*/

up(Seats);

/\* barber is cutting hair.\*/

}

}

Customer {

while(true) {

/\* protects seats so only 1 customer tries to sit

in a chair if that's the case.\*/

down(Seats); //This line should not be here.

if(FreeSeats > 0) {

/\* sitting down.\*/

FreeSeats--;

/\* notify the barber. \*/

up(Customers);

/\* release the lock \*/

up(Seats);

/\* wait in the waiting room if barber is busy. \*/

down(Barber);

// customer is having hair cut

} else {

/\* release the lock \*/

up(Seats); // customer leaves

}

}

}

# 

# 6. IMPLEMENTATION

#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("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*WELCOME TO BARBER SHOP\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Enter 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);

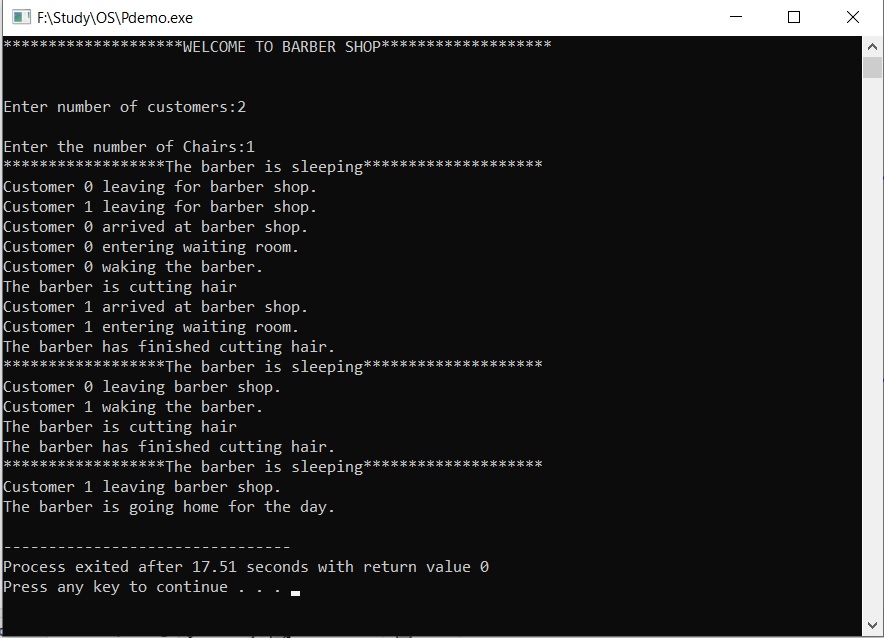
sleep(len);

}

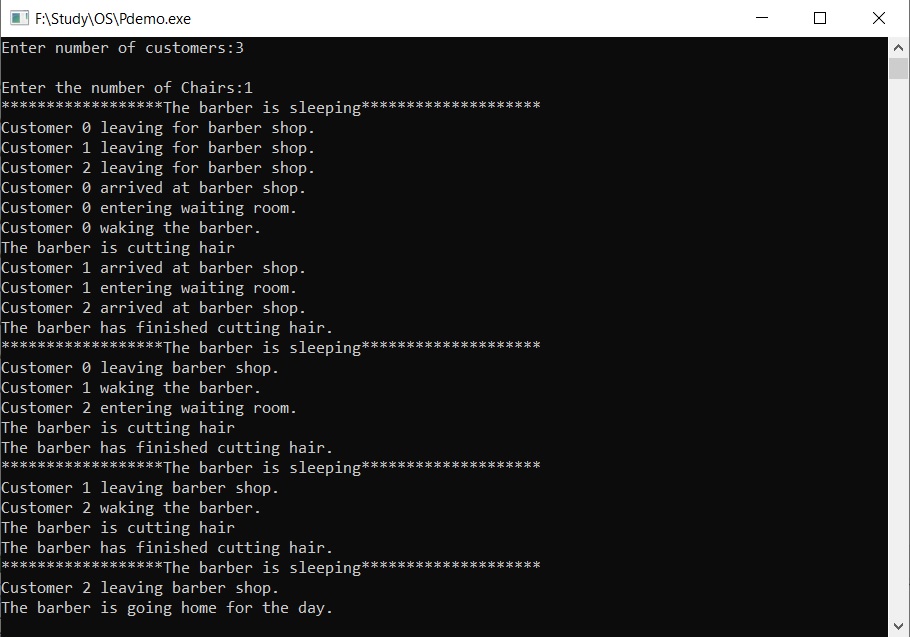
# 

# 7. RESULTS

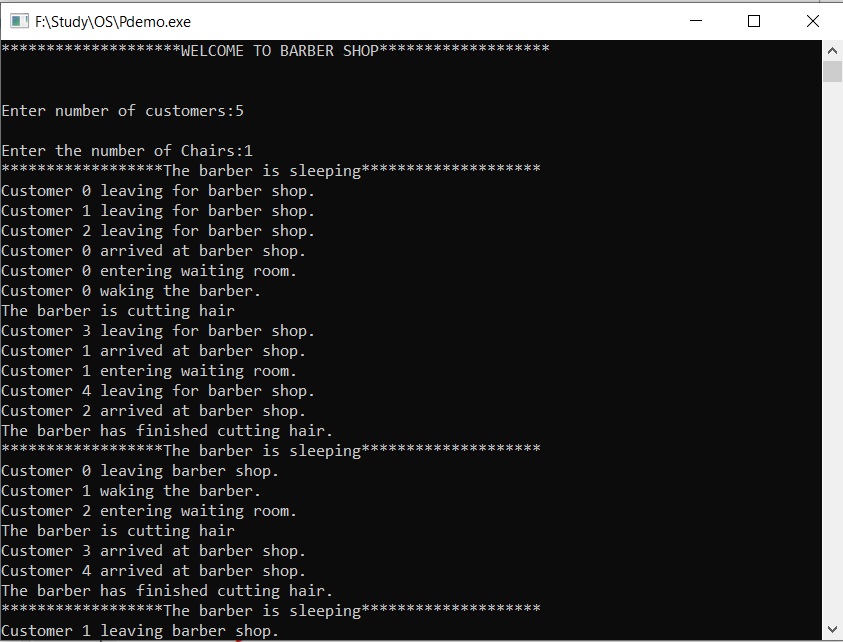
## 7.1 Output 1

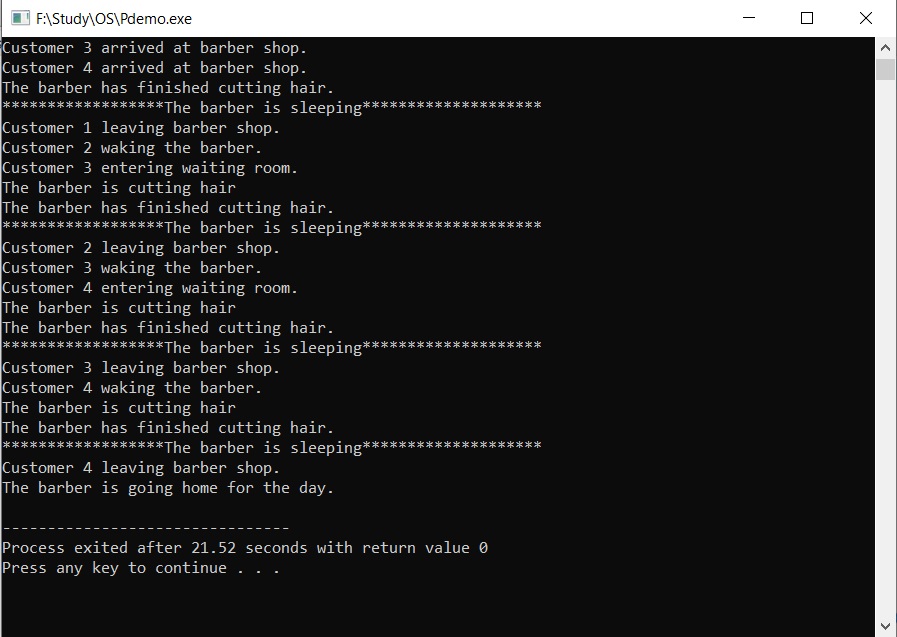


## 7.2 Output 2

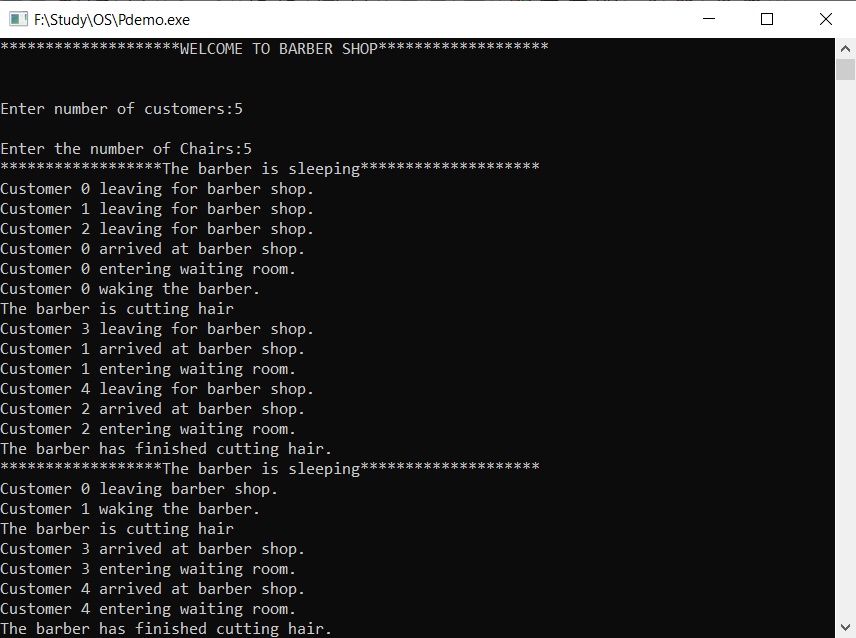


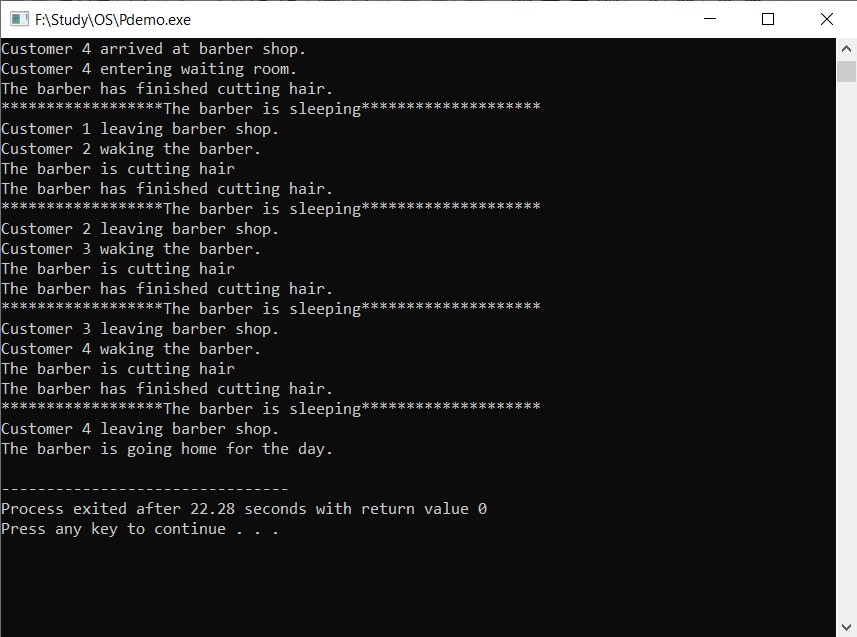
## 7.3 Output 3



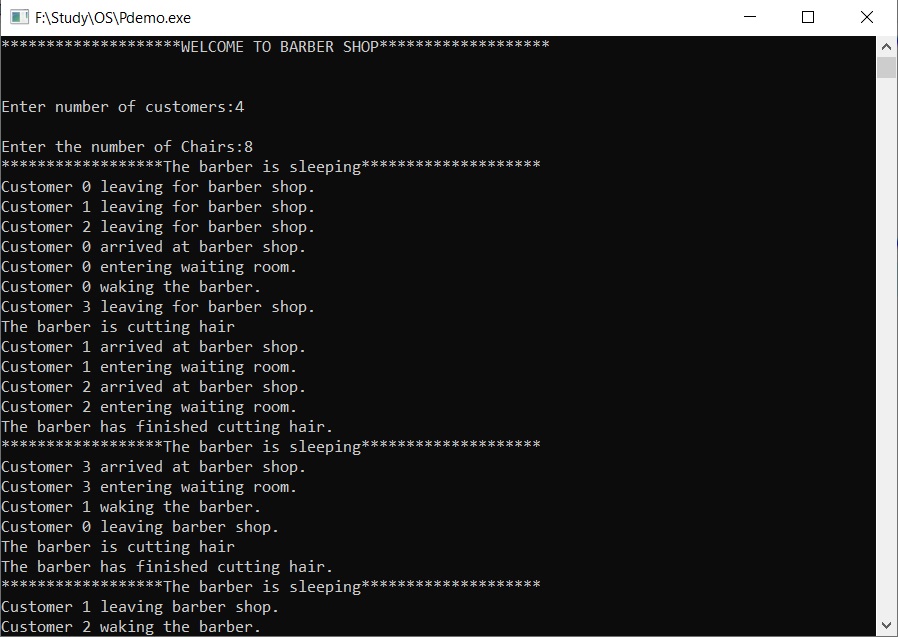


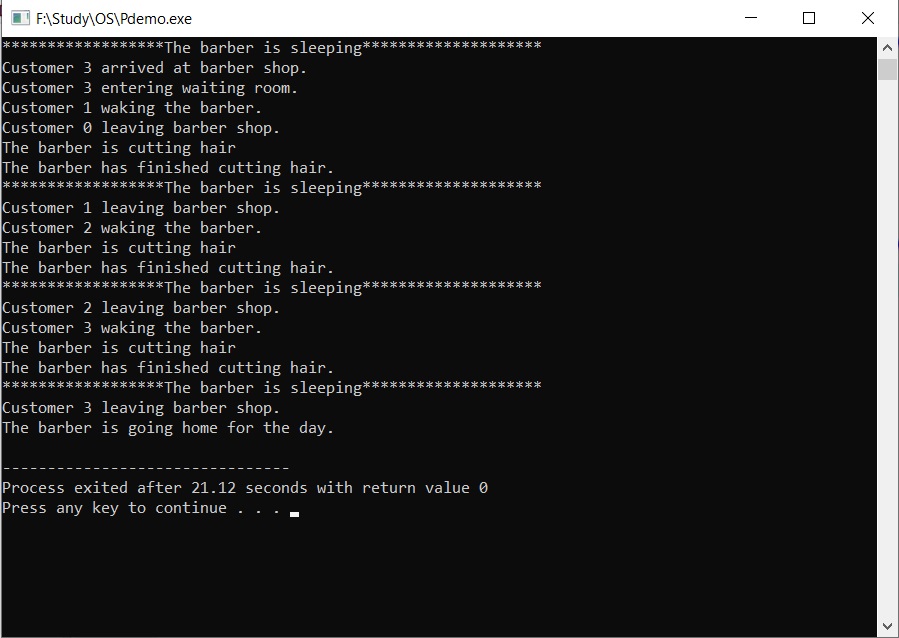
## 7.4 Output 4



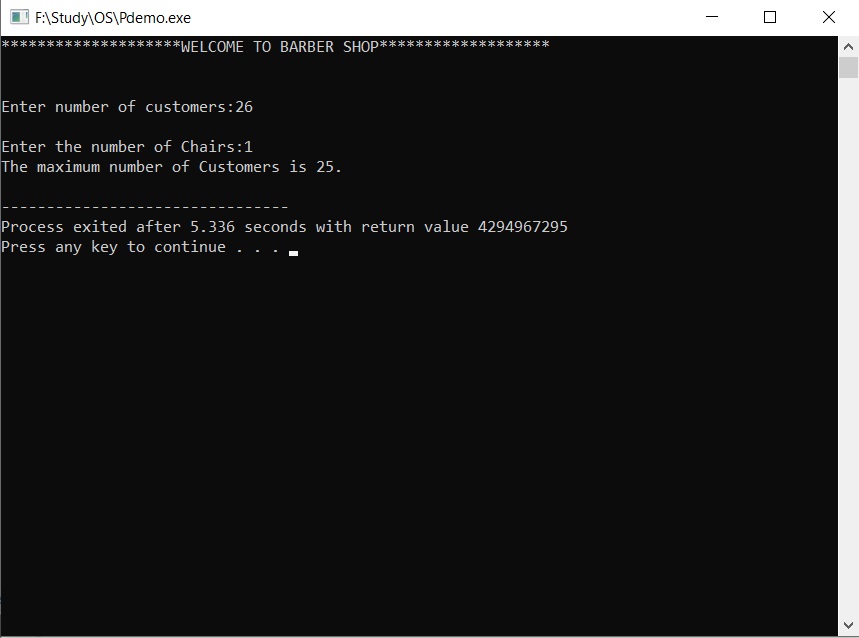


## 7.5 Output 5





## 7.6 Output 6



# 

# 8. REFERENCES

* Modern Operating Systems (2nd Edition) by [Andrew S. Tanenbaum](https://en.wikipedia.org/wiki/Andrew_S._Tanenbaum) ([ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [0-13-031358-0](https://en.wikipedia.org/wiki/Special:BookSources/0-13-031358-0))
* The Little Book of Semaphores by Allen B. Downey, <http://greenteapress.com/semaphores>
* [Cooperating sequential processes](http://www.cs.utexas.edu/users/EWD/transcriptions/EWD01xx/EWD123.html) by E.W. Dijkstra. Technical Report EWD-123, 1965, Eindhoven University of Technology, The Netherlands.
* <http://www.geeksforgeeks/inter> process communication.com