

**PROJECT REPORT**

OPERATING SYSTEM

Department of Software Engineering

Sir Syed University of Engineering & Technology, Karachi, Pakistan

**INSTRUCTOR**

MISS SANA, MISS FALAK PROJECT TITLE:

**GROUP MEMBERS**  SLEEPING BARBER

Adnan Asad (BSE-2020F-249)

Rida Bhatti (BSE-2020F-217)

Maham Fatima (BSE-2020F-211)

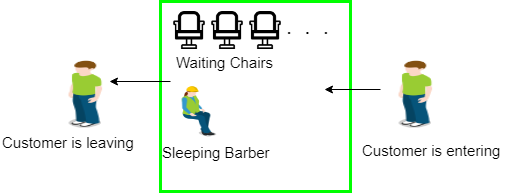
**COURSE ID**

SWE-204L

**Date: 16-JUNE-2022**

**INTRODUCTION:**

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.



**PROJECT SCOPE:**

The sleeping barber problem is a classic inter-process communication and synchronization problem that illustrates the complexities that arise when there are multiple operating system processes.

**SOLUTION:**

The solution to this problem includes three semaphores. 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 than 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.

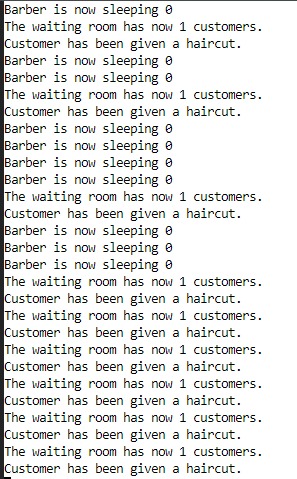
**FLOW CHART:**



**CODES:**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <sys/mman.h>  #include <sys/types.h>  #include <sys/wait.h>  #include <unistd.h>  static int \*waitingRoomCust;  int main(int argc, char const \*argv[]) {  waitingRoomCust = mmap(NULL, sizeof \*waitingRoomCust, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);  \*waitingRoomCust = 01;  srand(time(NULL));  int barber = fork();  printf("%d\n", barber);  fflush(stdout);  if (barber == 0) {  while (1) {  if (\*waitingRoomCust > 0) {  (\*waitingRoomCust)--;  printf("Customer has been given a haircut.\n");  fflush(stdout);  sleep((rand() % 12));  }  else {  printf("Barber is now sleeping %d\n", \*waitingRoomCust);  fflush(stdout);  sleep(1);  }  }  }  if (barber > 0) {  while (1) {  sleep(7);  if (\*waitingRoomCust <= 3) {  (\*waitingRoomCust)++;  printf("The waiting room has now %i customers.\n", \*waitingRoomCust);  fflush(stdout);  }  else {  printf("Waiting room is full, customer has left.\n");  fflush(stdout);  }  }  }  if (barber < 0) {  printf("failed %i ", barber);  fflush(stdout);  }  return 0;  } |

**OUTPUT:**

****

**CONCLUSION:**

To maintain the order of the customers, we should implement a data structure, known as Queue, which follows the First-in First-out principle. Instead of using a single Semaphore for customers, a list of Semaphore will be used. From now onward, we will refer a customer as a thread. So for threads, a list of Semaphore will be used, named as a queue. The intuition behind this approach is that: As each thread enters the barbershop, it creates a thread and puts it in the queue. Instead of waiting for the barber to be idle, each thread waits on its own semaphore. Whenever a barber is either idle or awaken, he removes the thread from the queue, and signals it to proceed to the chair.