**Operating System Mini Project**

Topic:

Sleeping Barber problem in Process Synchronization

Group members:

Name : Sakshi Patil **Sap ID : 60004180091**

Name : Samiksha Patil **Sap ID : 60004180092**

Computer SE-B

Batch : B2

**Problem Statement:**

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.

**Description:**

Description of Problem:

The sleeping barber problem is a classic inter-process communication and synchronization problem between multiple operating system processes. The problem is analogous to that of keeping a barber working when there are customers, resting when there are none, and doing so in an orderly manner.

The analogy is based upon a hypothetical barber shop with one barber. The barber has one barber chair and a waiting room with a number of chairs in it. When the barber finishes cutting a customer's hair, he dismisses the customer and then goes to the waiting room to see if there are other customers waiting. If there are, he brings one of them back to the chair and cuts his hair. If there are no other customers waiting, he returns to his chair and sleeps in it.

Each customer, when he arrives, looks to see what the barber is doing. If the barber is sleeping, then the customer wakes him up and sits in the chair. If the barber is cutting hair, then the customer goes to the waiting room. If there is a free chair in the waiting room, the customer sits in it and waits his turn. If there is no free chair, then the customer leaves.

Based on a naïve analysis, the above description should ensure that the shop functions correctly, with the barber cutting the hair of anyone who arrives until there are no more customers, and then sleeping until the next customer arrives.

**Solution to the problem :**

 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.

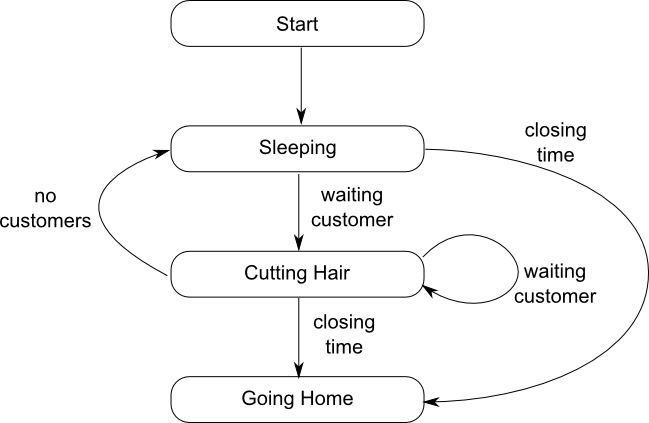
When the barber shows up in the morning, he executes the procedure barber, causing him to block on the semaphore customers because it is initially 0. Then the barber goes to sleep until the first customer comes up.

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.

**Flowchart:**



**Algorithm:**

Semaphore Customers = 0;

Semaphore Barber = 0;

Mutex accessSeats = 1;

int NumberOfFreeSeats = N;

Barber {

      while(1) {

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

            sem\_wait(Customers);

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

            sem\_wait(accessSeats);

            /\* a chair gets free.\*/

            NumberOfFreeSeats++;

            /\* bring customer for haircut.\*/

            sem\_post(Barber);

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

            sem\_post(accessSeats);

            /\* barber is cutting hair.\*/

      }

}

Customer {

      while(1) {

            /\* protects seats so only 1 thread tries to sit in a chair if that's the case.\*/

            sem\_wait(accessSeats);

            if(NumberOfFreeSeats > 0) {

                  /\* sitting down.\*/

                  NumberOfFreeSeats--;

                  /\* notify the barber. \*/

                  sem\_post(Customers);

                  /\* release the lock \*/

                  sem\_post(accessSeats);

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

                  sem\_wait(Barber);

                  // customer is having hair cut

            } else {

                  /\* release the lock \*/

                  sem\_post(accessSeats);

                  // customer leaves

            }

      }

}

**Explanation of semaphores used in Code:**

**sem\_t waitingRoom :** waitingRoom Limits the number of customers allowed to enter the waiting room at one time.

**sem\_t barberChair :**  barberChair ensures mutually exclusive access to the barber chair.

**sem\_t barberPillow :** barberPillow is used to allow the barber to sleep until a customer arrives.

**Code:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.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];

int i, x, numCustomers, numChairs; int Number[MAX\_CUSTOMERS];

printf("Maximum number of customers can only be 25. Enter number of customers and chairs.\n");

scanf("%d",&x);

numCustomers = x;

scanf("%d",&x);

numChairs = x;

if (numCustomers > MAX\_CUSTOMERS) {

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

system("PAUSE");

return 0;

}

printf("A solution to the sleeping barber problem using semaphores.\n");

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]);

}

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

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

}

allDone = 1;

sem\_post(&barberPillow);

pthread\_join(btid,NULL);

system("PAUSE");

return 0;

}

void \*customer(void \*number) {

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

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

randwait(5);

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);

}

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(3);

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

sem\_post(&seatBelt);

}

else {

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

}

}

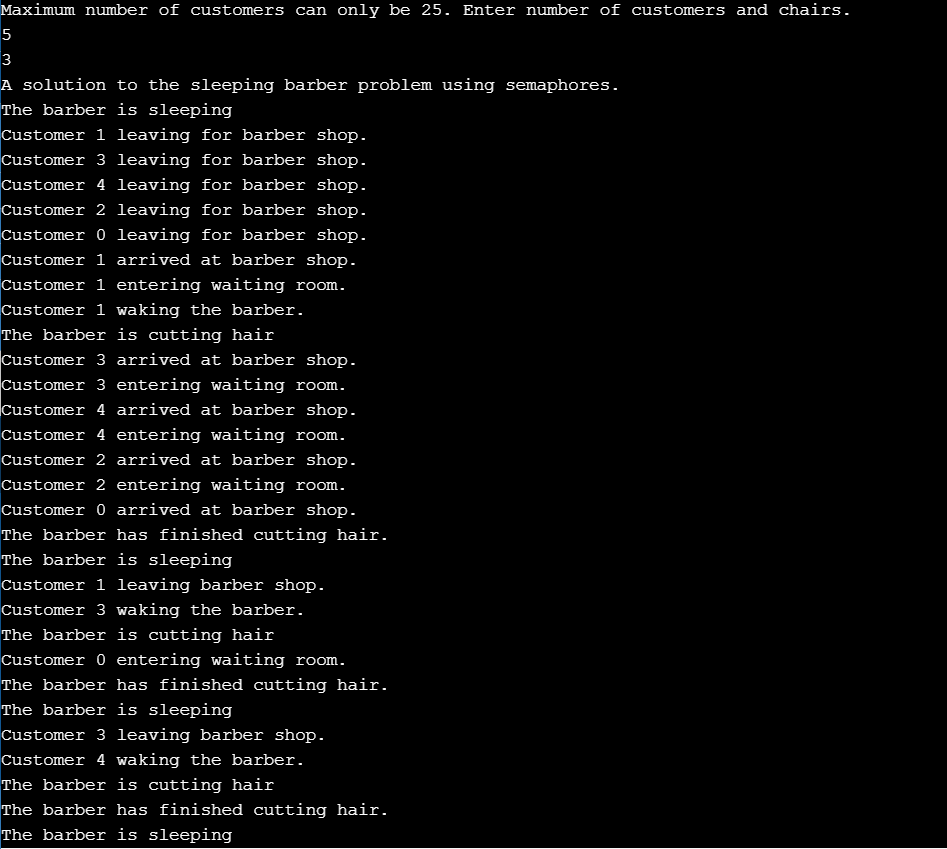
}

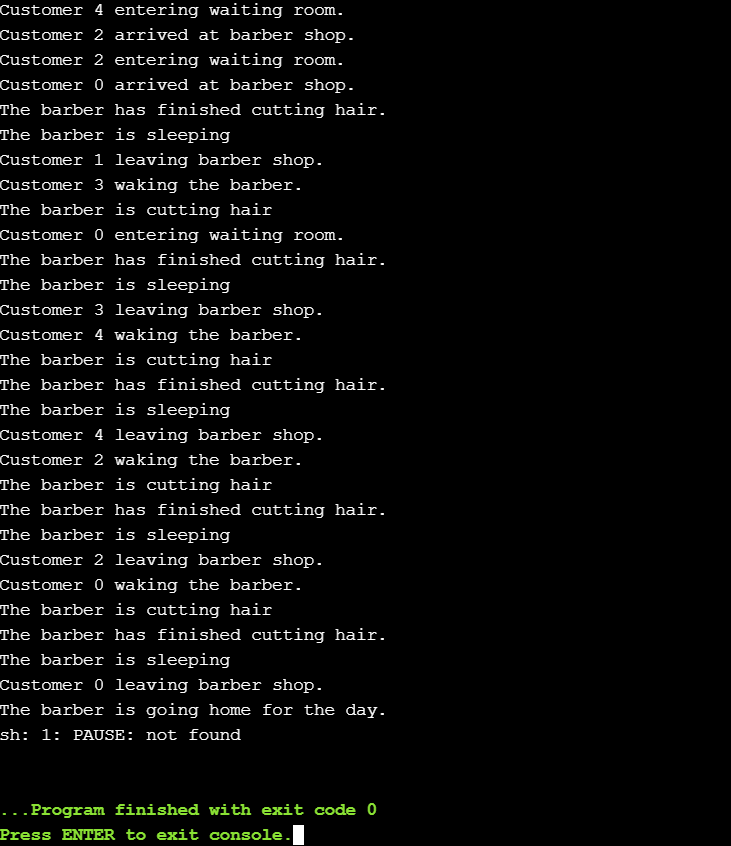
void randwait(int secs) {

int len = 1;

sleep(len);

**Output:**





**Comparison with Existing Technique:**

Sleeping Barber problem done be done using semaphores as well as monitors.

Semaphore and Monitor both allow processes to access the shared resources in mutual exclusion.

The basic difference between semaphore and monitor is that the semaphore is an integer variable S which indicate the number of resources available in the system whereas, the monitor is the abstract data type which allows only one process to execute in critical section at a time.

We have explained and implemented sleeping barber problem using semaphore earlier . Following is the explanation of the same problem using Monitors.

 monitor is a construct such as only one process is active at a time within the monitor. If other process tries to access the shared variable in monitor, it gets blocked and is lined up in the queue to get the access to shared data when previously accessing process releases it.

The **conditional variable** can invoke only two operation **wait**() and **signal**(). Where if a process **P invokes a wait()** operation it gets suspended in the monitor till other process **Q invoke signal()** operation i.e. a signal() operation invoked by a process resumes the suspended process.

• For the Barbershop, the monitor provides an environment

for the customers and barber to rendezvous

• There are four synchronisation conditions:

– Customers have to wait for barber to become available to get a

haircut

– Customers have to wait for barber to open door for them

– Barber needs to wait for customers to arrive

– Barber needs to wait for customer to leave

• Processes

– wait on conditions using wait()s in loops

– Signal() at points when conditions are true

**Algorithm using monitors:**

monitor sleepingbarber;

int waiting;

const int CHAIRS;

condn barber,customer;

void get\_haircut(){

if (waiting < CHAIRS) {

waiting++;

cSiginal(customer);

cWait(barber);

//get hair cut

waiting--;

}

}

void cut\_hair(){

while (waiting == 0) cWait(customer)

cSignal(barber);

//cut hair

}

void customer() { gethaircut(); }

void barber() {

while true{ cut hair(); }

}

main()

parbegin(barber,customer)

**Conclusion:**

Hence we have implemented sleeping barber problem which is analogous to inter-process communication and synchronization problem between multiple operating system processes using semaphore and Mutual Exclusion.