Exp 11: Simulate Sleeping Barber Problems

The Sleeping Barber problem is a classic problem in process synchronization that is used to illustrate synchronization issues that can arise in a concurrent system. The problem is as follows:

There is a barber shop with one barber and a number of chairs for waiting customers. Customers arrive at random times and if there is an available chair, they take a seat and wait for the barber to become available. If there are no chairs available, the customer leaves. When the barber finishes with a customer, he checks if there are any waiting customers. If there are, he begins cutting the hair of the next customer in the queue. If there are no customers waiting, he goes to sleep.

The problem is to write a program that coordinates the actions of the customers and the barber in a way that avoids synchronization problems, such as deadlock or starvation.

One solution to the Sleeping Barber problem is to use semaphores to coordinate access to the waiting chairs and the barber chair. The solution involves the following steps:

Initialize two semaphores: one for the number of waiting chairs and one for the barber chair. The waiting chairs semaphore is initialized to the number of chairs, and the barber chair semaphore is initialized to zero.

Customers should acquire the waiting chairs semaphore before taking a seat in the waiting room. If there are no available chairs, they should leave.

When the barber finishes cutting a customer's hair, he releases the barber chair semaphore and checks if there are any waiting customers. If there are, he acquires the barber chair semaphore and begins cutting the hair of the next customer in the queue.

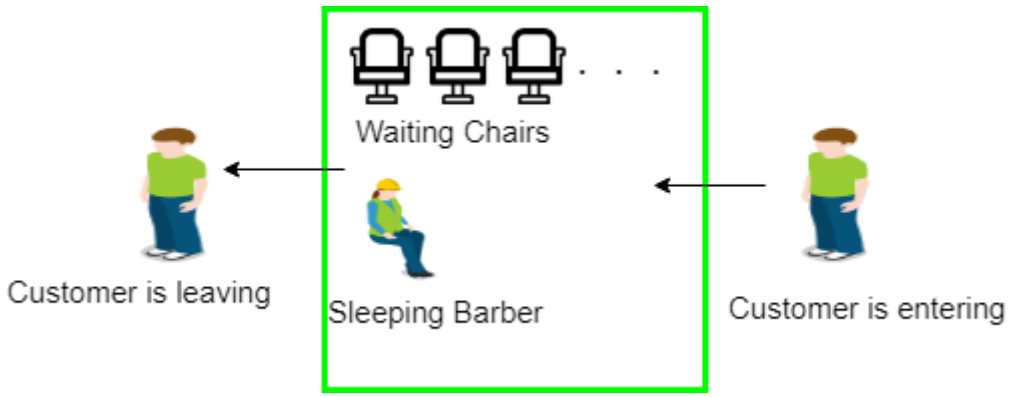
The barber should wait on the barber chair semaphore if there are no customers waiting.

The solution ensures that the barber never cuts the hair of more than one customer at a time, and that customers wait if the barber is busy. It also ensures that the barber goes to sleep if there are no customers waiting.

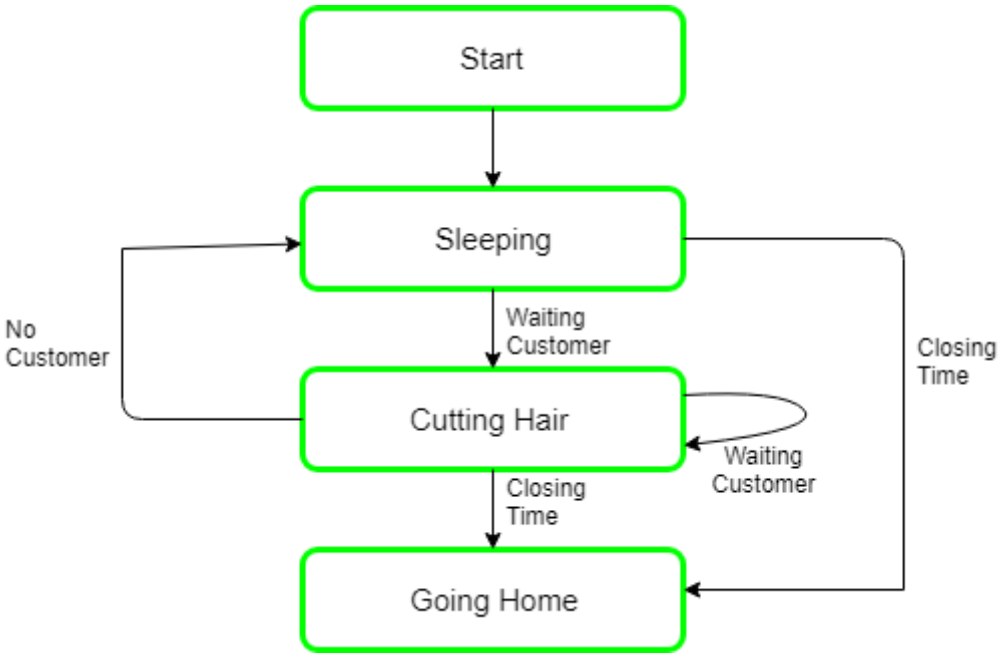
However, there are variations of the problem that can require more complex synchronization mechanisms to avoid synchronization issues. For example, if multiple barbers are employed, a more complex mechanism may be needed to ensure that they do not interfere with each other.

Prerequisite – 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.



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 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.



Program Code:

```
#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);
      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; // Set allDone to true to signal barber to finish up and exit waiting

    sem_post(&barberPillow); // Wake the barber so he will exit.
    pthread_join(btid, NULL);

    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); // Give up the chair.
    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; // Generate an arbit number... sleep(len);
}
```