MULTIPLE SLEEPING BARBER PROBLEM

Problem definition

Sleeping barber problem is a synchronization problem between multiple operating system processes.

The problem is similar to that of keeping the barber working when there are customers, resting “sleeping” when there are none, and doing so in an orderly manner.

The sleeping barber problem consists of one barber, one barber's chair in a cutting room, and a waiting room containing a number of chairs in it. Each customer, when they arrive, looks to see what the barber is doing, if the barber is sleeping, the customer wakes him up and sits in the cutting room chair. If the barber is busy cutting hair, the customer stays in the waiting room and waits for their turn. When the barber finishes cutting a customer's hair, he dismisses the customer and goes to the waiting room to see if there are others waiting, if there are, he brings one of them back to the chair and cuts his hair, if there are none, he returns to the chair and sleeps in it.

Solution pseudocode

```
#define CHAIRS 5                    /* number of chairs for waiting customers */
typedef int semaphore;

semaphore customers = 0;            /* number of waiting customers */
semaphore barbers = 0;              /* number of barbers waiting for customers */
semaphore mutex = 1;                /* for mutual exclusion */
int waiting = 0;                    /* customers are waiting not being haircut */
```

```
void Barber(void)

{

        while (TRUE)

        {       down(customers);      /* go to sleep if number of customers is 0 */
        down(mutex);          /* acquire access to 'waiting' */                waiting
= waiting – 1;  /* decrement count of waiting customers */
                up(barbers);          /* one barber is now ready to cut hair */

                up(mutex);            /* release 'waiting' */

                cut_hair();           /* cut hair, non-CS */

        }

 }

void customer(void)

{       down(mutex);                  /* enter CS */
        if (waiting < CHAIRS)

        {

                waiting = waiting + 1;  /* increment count of waiting customers */
                up(customers);        /* wake up barber if necessary */

                up(mutex);            /* release access to 'waiting' */

                down(barbers);        /* wait if no free barbers */
                get_haircut();        /* non-CS */

        }

        else{

                up(mutex);             /* shop is full, do not wait */
        }

}
```

What is deadlock?

A deadlock occurs when two or more threads wait forever for a lock or resource held by another of the threads.

Sleeping Barber Deadlock

Deadlock is a state in which each member of a group is waiting for another member, including itself, to take action.

Deadlock Problem

There might be a scenario in which the customer ends up waiting on the barber and a barber waiting on the customer, which would result in a deadlock.

Deadlock Example

A customer walks in and sees the barber still cutting hair, while waiting, he went to the comfort room. Simultaneously, the barber finishes cutting the hair of the first customer and checks if a customer is waiting at the waiting room.

Deadlock Solution

The solution to these problems involves the use of three semaphores out of which one is a mutex “binary semaphore”.

3 Semaphores:

Is simply a variable. This variable is used to solve critical section problems and to achieve process synchronization in the multi-processing environment.

- Customer (CustReady): Helps in counting the waiting customers.
- Barber (BarberReady): To check the status of the barber either idle ”sleeping” or not ”cutting customer’s hair”.
- NumberOfFreeChairs: To keep the count of the available chairs, so customer either wait if there is free chairs or leave if there are none.

2 Mutex:

Or a lock is a synchronization mechanism for enforcing limits on access to resources in an environment where there are many threads of execution.

- accessChairs: A mutex allows the customer to get exclusive access to the number of free chairs and allows them to increase or decrease the number.

- accessBarber: A mutex allows the barber to get exclusive access to customer.

What is starvation?

Starvation might happen to a customer who is waiting for a long time when the customers don't follow order or when the barber calls out a customer randomly.

Starvation Solution

The problem of starvation can be solved by linked list where customers are added as they arrive, so that the barber may serve them on a first come first served basis “FIFO: First IN, First OUT”.

A Real-World Example

The Airport With A Single Check-In Counter Problem.

Imagine an airport with a single check-in counter and two queues, one for business class and one for economy class. As you may know, the business class has priority over the economy class.

However, there is a business convention period and the business queue is always full. The economy queue doesn’t move at all and the people in that queue will miss the plane as they cannot access the check-in counter.