Multiple sleeping barber problem

1-pseudocode:

Semaphore barberReady=0

Semaphore accessWRSeats = 1

🡪if 1, the number of seats in the waiting room can be incremented or decremented.

Semaphore custReady = 0

🡪the number of customers currently in the waiting room, ready to be served.

int numberOfFreewRSeats = N

🡪total number of seats in the waiting room

def Barber():

while true:

🡪Run in an infinite loop.

wait(custReady)

🡪 Try to acquire a customer - if none is available, go to sleep.

wait(accessWRSeats)

🡪Awake - try to get access to modify # of available seats, otherwise sleep.

number0fFreeWRSeats += 1

🡪One waiting room chair becomes free.

signal (barberReady)

🡪I am ready to cut.

signal (accessWRSeats)

🡪Don't need the lock on the chairs anymore.

# (Cut hair here.)

def Customer ():

while true:

🡪 Run in an infinite loop to simulate multiple customers.

wait(accessWRSeats)

🡪Try to get access to the waiting room chairs.

if numberOfFreeWRSeats >0:

🡪 If there are any free seats:

numberofFreeWRSeats-=1

🡪sit down in a chair

signal (custReady)

🡪notify the barber, who's waiting until there is a customer

signal (accessWRSeats)

🡪don't need to lock the chairs anymore

wait (barberReady)

🡪wait until the barber is ready

# (Have hair cut here.)

else: 🡪otherwise, there are no free seats.

signal(accessWRSeats)

🡪but don't forget to release the lock on the seats

# (Leave without a haircut.)

Example of deadlock:

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 to a deadlock.

Solution:

The solution to this problem involves the use of three semaphores out of which one is a mutex(binary semaphore).

🡪3 semaphores:

1. Customer (custReady) : helps in counting the waiting customers.
2. Barber (barberReady) : to check the status of barber either idle or not.
3. NumberOfFreeWRSeats : to keep the count of the available seats , so customer either wait if there is free seats or leave if there are none.

#1 mutex:  
 AccessWRSeats : a mutex which allows the customer to get exclusive access to the number of free seats and allows them to increase or decrease the number.

Example of starvation :

Starvation might happen to a customer who is waiting for a long time when the customer do not follow order or when the barber calls out a customer randomly.

Solution:

The problem of starvation can be solved by utilizing a queue 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)

Explanation for real world application and how did apply the problem:

Real world application: barbershop, carwash center, clinic.

How did we apply the problem🡪Imagine a hypothetical barbershop with one barber, one barber chair, and a waiting room with n chairs (n may be 0) for waiting customers. The following rules apply:

#If there are no customers, the barber falls asleep in the chair.

#A customer must wake the barber if he is asleep.

#If a customer arrives while the barber is working, the customer leaves if all chairs are occupied and sits in an empty chair if it's available.

#When the barber finishes a haircut, he inspects the waiting room to see if there are any waiting customers and falls asleep if there are none.

There are two main complications. First, there is a risk that a race condition, where the barber sleeps while a customer waits for the barber to get them for a haircut, arises because all of the actions(checking the waiting room, entering the shop, taking a waiting room chair)take a certain amount of time. Specifically, a customer may arrive to find the barber cutting hair so they return to the waiting room to take a seat but while walking back to the waiting room the barber finishes the haircut and goes to the waiting room, which he finds empty (because the customer walks slowly or went to the restroom) and thus goes to sleep in the barber chair. Second, another problem may occur when two customers arrive at the same time when there is only one empty seat in the waiting room and both try to sit in the single chair; only the first person to get to the chair will be able to sit.

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

There are several possible solutions, but all solutions require a mutex, which ensures that only one of the participants can change state at once. The barber must acquire the room status mutex 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 a waiting room or barber chair, and also when they leave the shop because no seats were available. This would take care of both of the problems mentioned above. A number of semaphores is also required to indicate the state of the system. For example, one might store the number of people in the waiting room.