Project description:

In computer science, 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 imagine a hypothetical barbershop with k

barbers. When there are no customers, all barbers sleep in their

chairs. 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

Solution pseudocode:

\*The following pseudo-code guarantees synchronization between barber and customer and is deadlock free, but may lead to starvation of a customer. P and V are functions provided by the semaphores.

\*You need (as mentioned above): + Semaphore Customers = 0 + Semaphore Barber = 0 + Semaphore accessSeats (mutex) = 1 + int NumberOfFreeSeats = N //total number of seats

\*The Barber (Thread/Process): while(true) { //runs in an infinite loop P(Customers) //tries to acquire a customer - if none is available he goes to sleep P(accessSeats) //at this time he has been awakened - want to modify the number of available seats NumberOfFreeSeats++ //one chair gets free V(Barber) //the barber is ready to cut V(accessSeats) //we don't need the lock on the chairs anymore //here the barber is cutting hair }

\*The Customer (Thread/Process): P(accessSeats) //tries to get access to the chairs if ( NumberOfFreeSeats > 0 ) { //if there are any free seats NumberOfFreeSeats-- //sitting down on a chair V(Customers) //notify the barber, who's waiting that there is a customer V(accessSeats) //don't need to lock the chairs anymore P(Barber) //now it's this customers turn, but wait if the barber is busy //here the customer is having his hair cut } else { //there are no free seats //tough luck V(accessSeats) //but don't forget to release the lock on the seats //customer leaves without a haircut }

Examples of Deadlock :

Deadlock is where no process proceeds, and get blocked can prevented by Avoiding mutual exclusion, hold and wait, and circular wait and allowing preemption,Deadlock arises when four conditions Mutual exclusion, Hold and wait, No preemption, and Circular wait occurs simultaneously.

How did solve deadlock :

Absence of deadlock: For this scenario, the deadlock will occur if the customer ends up waiting for the barber and the barber ends up waiting for the customer to arrive. To handle this problem my code has used reentrant locks and after a thread acquires a lock it sleeps for few milliseconds and then release the lock.[4] The code ensures that each thread releases the lock after performing the critical section. I have also used try-catch blocks to handle exceptions. Using reentrant locks, the critical section which is inside the locks can only be accessed by one thread at a time.

Examples of starvation:

The problem of starvation generally occurs in priority scheduling algorithm. In priority scheduling algorithm, the process with higher priority is always allocated the resource, preventing the lower priority process from getting the requested resource

How did solve starvation :

Absence of Starvation: For this scenario problem of starvation will occur if the customers don’t follow any order for getting a haircut, as some won’t get a haircut even though even after waiting for a long time. To handle this problem in my code I have inserted the customers in a linked list which follows the first in first out property. So, every time a customer sits in a waiting room, they will be selected by the barber in first come first serve basis. We could have also used other data structures like a stack, but the linked list seems like the best choice for this scenario.

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

Real-world Example of this design:

This design is the best analogy for a customer care call centre. Initially when there is no customer on-call all call-executives just relax and wait for the call. The moment the first customer dials the number he/she is connected to any call-executive and in a scenario when all call-executives are busy the customer will have to wait in a queue till they are assigned to a call-executive. If all executives are busy and the waiting line is full, the customers are disconnected with a message that executives are busy and customers will be contacted later by the company. This best relates to this design as the customers are picked from the queue in a first come first serve basis and call-executives are utilized in such a way that everyone executive gets at least one call.

In this scenario we can have the following design similarities:

1. The critical section will be the call between executive and customer

2. The waiting room will be the waiting queue over a call, where customers will be held in a FIFO manner.

3. Locks can be acquired on the waiting queue so that no two executives pick the same customer.