**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 pseudocode** :

The solution to this problem includes three semaphore

-First is for the customer which counts the number of customers present in the waiting room .

- Second, the barber 0 or 1 is used to tell whether the barber is idle or is working

CONDITIONS:

1. The 'sleeping barber problem' consist of one barber, one barber's chair in a cutting room and a waiting room containing a number of chairs in it.

2. 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.

3. If the barber is busy cutting hair, the customer stays in the waiting room and waits for their turn.

4. 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 the them back to the chair and cuts their hair, if there are none, he returns to the chair and sleeps in it.

5. If a customer enters the Shop and all chairs are occupied, the customer leaves the shop.

**Deadlock:**

Deadlock is defined as a situation where set of processes are blocked because each process holding a resource and waiting to acquire a resource held by another process.

**Example:**

when two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone.

In simple words, we can say that if two or more processes are waiting for some events to happen, which never happens, then, that is called deadlock and those processes are in deadlock state.

**Example Scenario:**

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

**How to solve deadlock:**

The solution to these problem involves the use of three semaphores out of which one is a mutex(binary

semaphore).

3 Semaphores:

Customer(CustReady): Helps in counting the waiting customers.

Barber(BarberReady): To check the status of the barber either idle or not.

NumberOfFreeWRSeats: To keep the count of the available seats, so customer either wait if there is free seats or leave if there are none.

deadlock can be prevented by **eliminating any of the four necessary conditions, which are mutual exclusion, hold and wait, no preemption, and circular wait**. Mutual exclusion, hold and wait and no preemption cannot be violated practically. Circular wait can be feasibly eliminated by assigning a priority to each resource.

1. Mutual Exclusion

Mutual section from the resource point of view is the fact that a resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock. If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.

However, if we can be able to violate resources behaving in the mutually exclusive manner then the deadlock can be prevented.

## 2. Hold and Wait

Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

However, we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait. That means, a process must be assigned all the necessary resources before the execution starts. A process must not wait for any resource once the execution has been started.

## 3. No Preemption

Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.

## 4. Circular Wait

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

# **Deadlock avoidance**

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

Safe and Unsafe States