DOCUMENTATION

Description	Problem statement Imagine a hypothetical barbershop with one barber, one barber chair, and a waiting room with n chairs (n) 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,

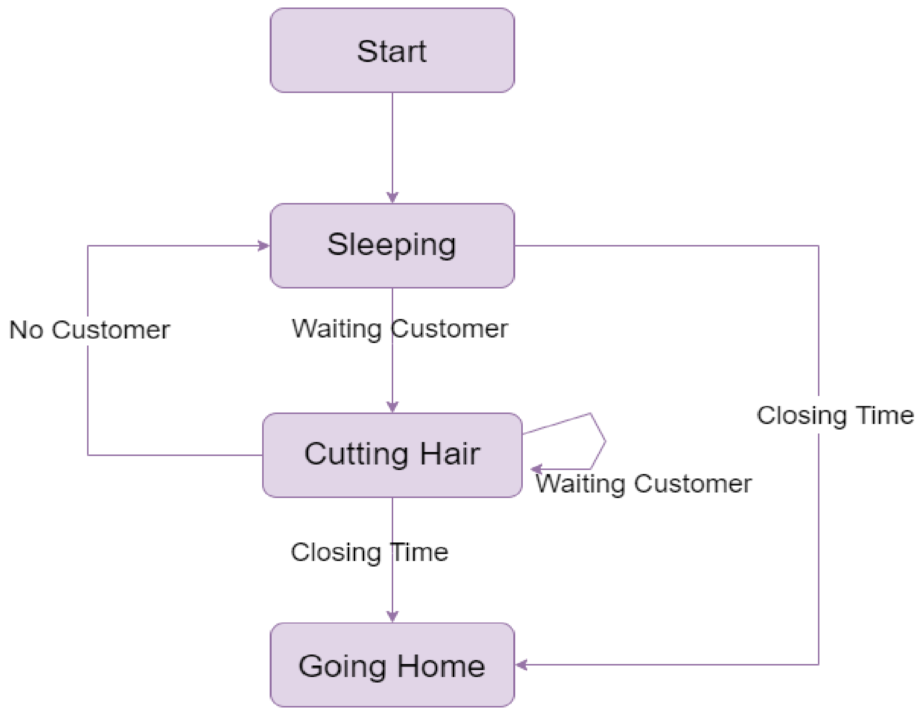
arises because all of the actions—checking the waiting room, entering the shop, taking a waiting room chair—take a certain amount of time. Specifically, a customer may arrive to find the barber cutting hair so they return to the waiting room to take a seat but while walking back to the waiting room the 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.

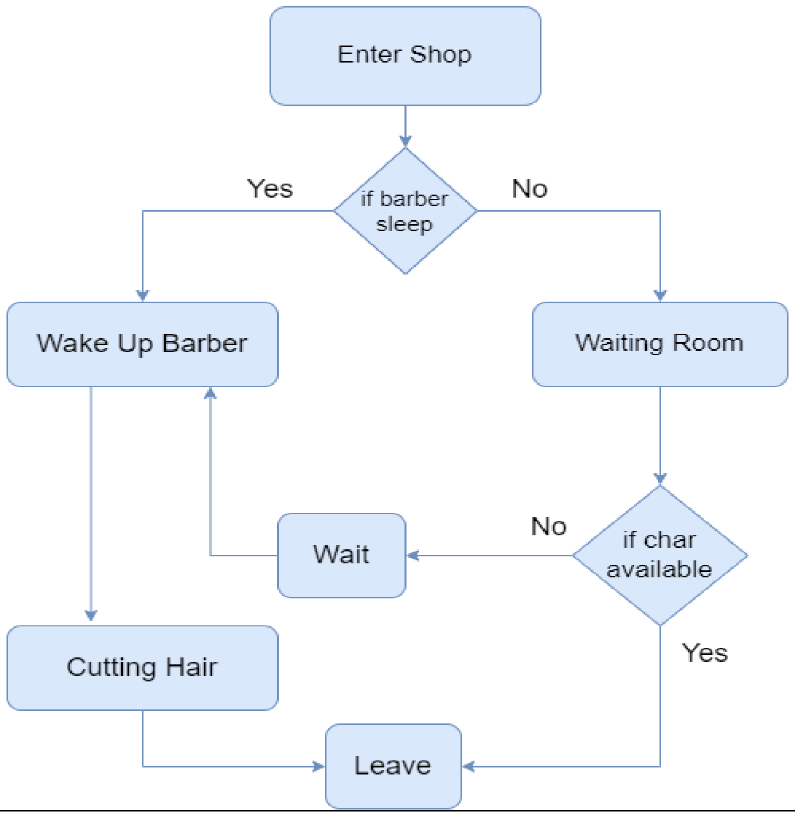
Solutions:

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.

Barber



Customer

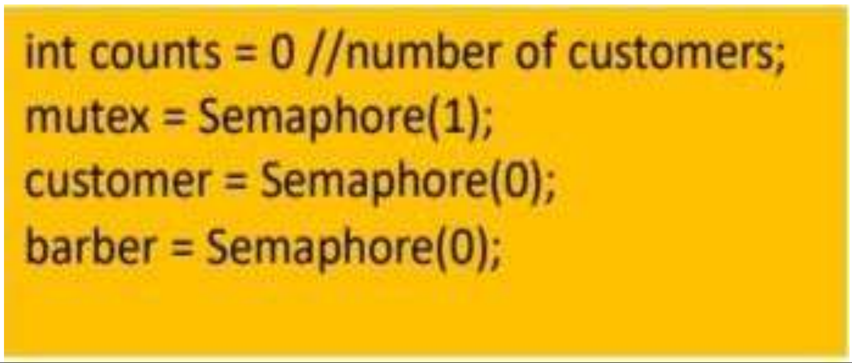


Solution Pseudocode	

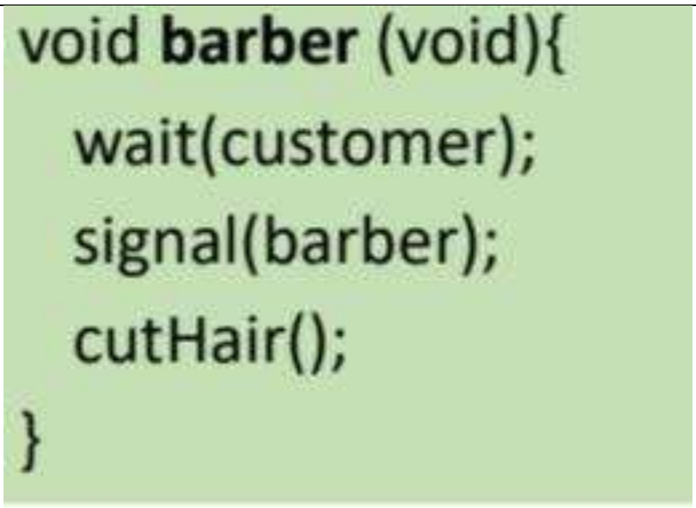
```
// number of patrons allowed in shop at a single instant
const int nChairs = N;

// customer semaphore
sCustomer = 0,

// barber's semaphore
barber = 0,

// for mutual exclusion to a shared area
mutex = 1;

//  Number of customers that are waiting
int nWaiting = 0;

Customer() {

    // Customer begins by trying to get into the shop
    acquire(mutex);

        // tack chair in the waiting area but must get into the
                                              critical section
    if (nWaiting < nChairs) {
        // If a chair is available.
        // waiting customer(s)
        nWaiting++;

        // Wake up Barber to start cust.'s hair cut
        release(customer);

        // Release my hold on mutual exclusion
        release(mutex);

        // Wait until the barber is ready for this cus.
        acquire(barber)

        Cutmyhair()

    }
    else {
  // Release my hold on mutual exclusion – no chairs   – I'm
```

```
        release(mutex)
    }
}

Barber() {
    While(TRUE) {
    // Catch up on my sleep if no customers are waiting.
        acquire(customer)

        // Mutual exclusion to grab the next customer
        acquire(mutex)

        // Who won't be waiting anymore
        nWaiting--

        //  As soon as I wake the poor soul up.
        release(barber)

        release(mutex)

        Clipaway()

    }
}
```

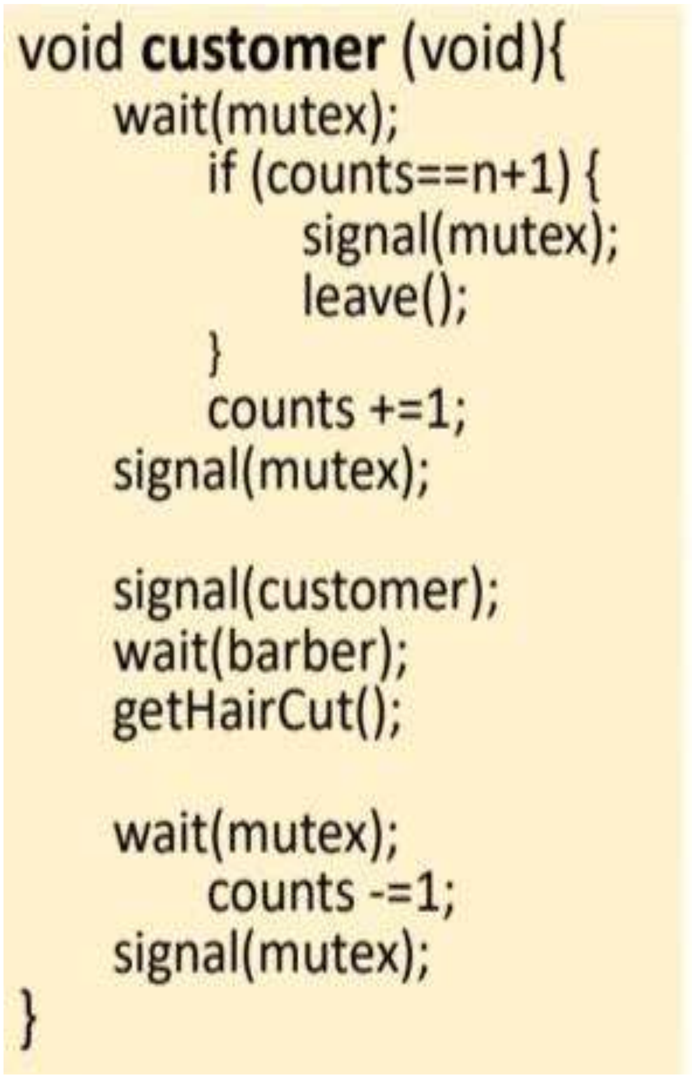
Reference :-



```
int counts = 0 //number of customers;
mutex = Semaphore(1);
customer = Semaphore(0);
barber = Semaphore(0);
```



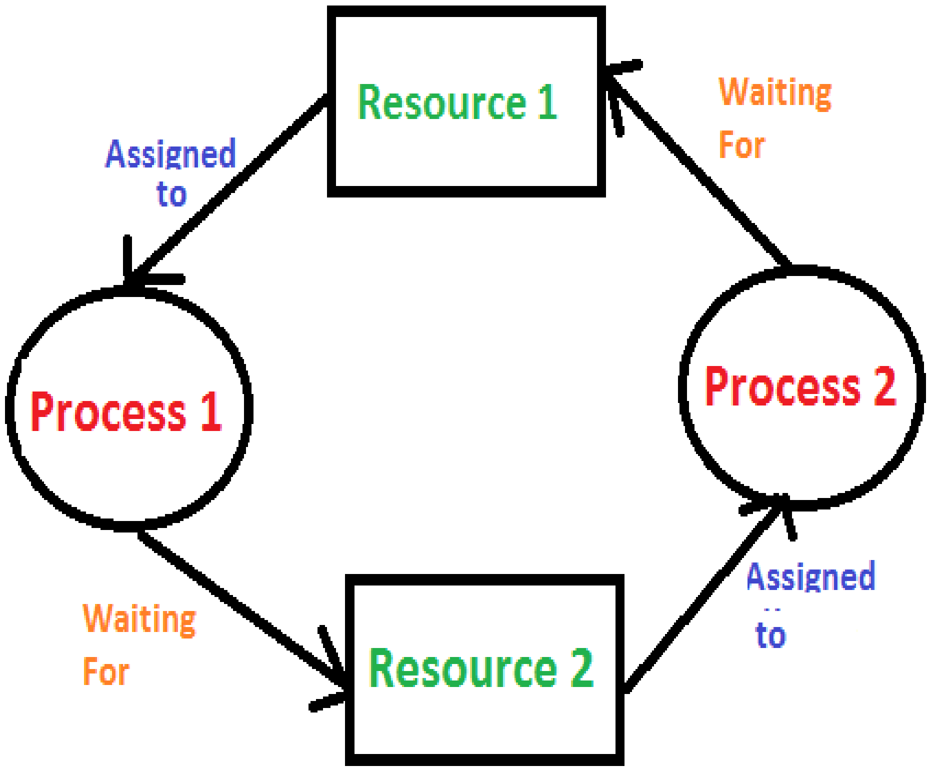
```
void barber (void){
   wait(customer);
   signal(barber);
   cutHair();
}
```



```
void customer (void){
      wait(mutex);
            if (counts==n+1) {
                  signal(mutex);
                  leave();
            }
            counts +=1;
      signal(mutex);

      signal(customer);
      wait(barber);
      getHairCut();

      wait(mutex);
            counts -=1;
      signal(mutex);
}
```

Problem & Examples Deadlock	Deadlock: The deadlock situation occurs when one of the processes got blocked. Deadlock is an infinite process. Every Deadlock always has starvation.

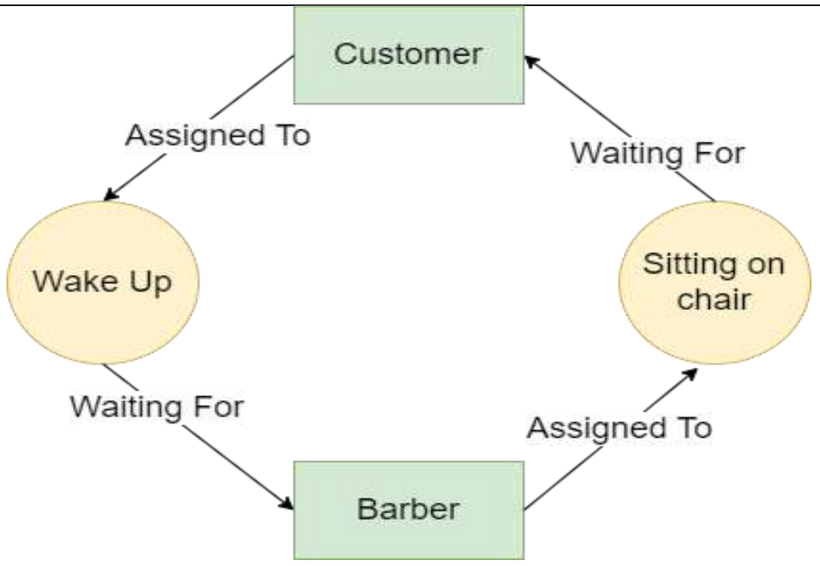
Deadlock happens then Mutual exclusion, hold and wait. Here, preemption and circular wait do not occur simultaneously. For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



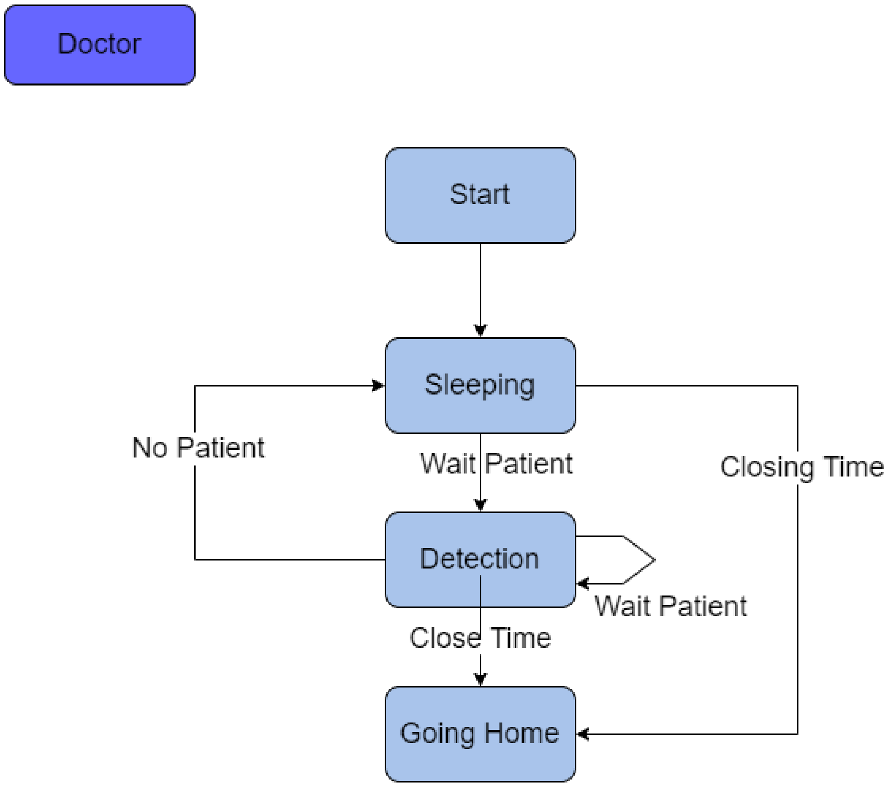
The deadlock occurs in sleeping barber shop:

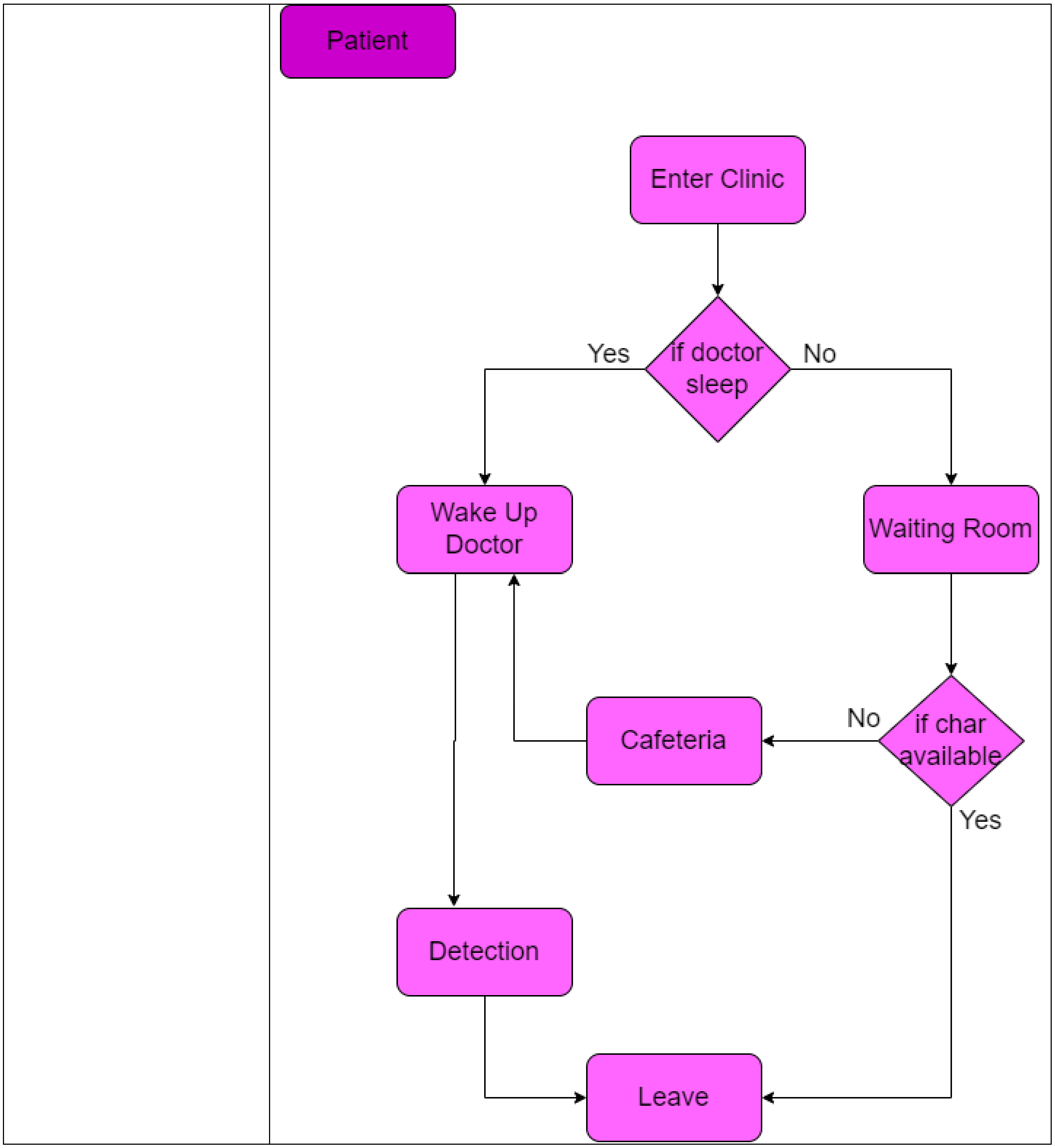
1-The barber sees no one sitting in a chair and thinks the waiting room is empty so goes to sleep.

2-The customer thinks the barber is busy so doesn't try to wake up the barber and just patiently waits for the barber.

	 Customer Assigned To Waiting For Wake Up Sitting on chair Waiting For Assigned To Barber
Solve Problem Deadlock	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. 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 [This solution participates in solving the starvation problem as well]. When the haircut is over, the customer exits the procedure and if there are no customers in waiting room barber sleeps.
Problem & Example Starvation	Starvation: is a situation where all the low priority processes got blocked, and the high priority processes execute. Starvation is a long waiting but not an infinite process. Every starvation doesn't necessarily have a deadlock. It happens due to uncontrolled priority and resource management. The Starvation occurs in sleeping barber shop: Two customers arrive at the same time, see that the barber is already busy, then both try to sit in the same chair, which causes: 1- One customer sits down and the other leaves 2- Both clients leave

Solve Problem Starvation	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.
Explanation for real world application and how did apply the problem	Doctor Night Duty Problem with one doctor, one detection room, one cafeteria and a waiting room with n chairs (n) for waiting patients. The following rules apply: Patient threads will alternate between having coffee in the cafeteria and being treated by the doctor. If the doctor is available, they will be treated immediately. Otherwise, they will either sit in a chair in the waiting room or, if no chairs are available, will go back to the cafeteria for a random amount of time and will return to see the doctor at a later time. If a patient arrives and notices that the doctor is taking a nap, the patient must notify the doctor using a semaphore. When the doctor finishes treating a patient, she must check to see if there are patients waiting for help in the waiting room. If so, the doctor must help each of these patients in turn. If no patients are present, the doctor may return to napping. *Solutions:* 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 doctor must acquire the room status mutex before checking for patients and release it when they begin either to sleep or detection; a patient must acquire it before entering the hospital and release it once they are sitting in a waiting room or detection room, and also when they go to cafeteria 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.  Doctor Start Sleeping No Patient Wait Patient Closing Time Detection Wait Patient Close Time Going Home


Patient
Enter Clinic
if doctor sleep
Yes
No
Wake Up Doctor
Waiting Room
if char available
No
Yes
Cafeteria
Detection
Leave