Pesudocode:

```
Public class OSproject extends Thread {
    Customers:Semaphore(0)
    barber:Semaphore(0)
    chairs:Semaphore(1)
    class Customer extends Thread {
        customer:int
        needscut:Boolean
        public Customer(int i){
            customer = i
        }
        Run(){
            While true
                Try to
                    Call chairs.acquire()
                    If freeseats>0
                        Display “Customer” + this.customer +”sat down”
                        Decrement freeseats by one
                        Call customer.release()
```

```
                    Call chairs.release()
                    Try to
                        Call barber.acquire()
                        Set needscut = false
                        Call getHaircut()
                     Catch
                Else
                    Display “NO free seats Customer”+ this.customer+”has left the barbershop”
                    Call chairs.release()
                    Set needsCut = false
            Catch
        }
    ENDWHILE
        Function getcutHair()
            Display “The barber is cutting hair”
            Try
               Sleep(5050)
            Catch
```

```
Class Barber extends Thread{
        Run(){
            While true
                Try to
                    Call customers.acquire()
                    Call chairs.release()
                    Increment freeseats by one
                    Call barber.release()
                    Call chairs.release()
                    This.cutHair()
                Catch
    }
}
        Function cutHair()
            Display “The barber is cutting hair”
            Try to
                Sleep(5000)
            Catch
```

In the main function

Input

Number of customers

Number of chairs

Create instance of barber shop

Start()

Run ()

Create instance of barber

Start()

For 1 to number of customers

Create instance of customer

Start()

Try to

Sleep()

Catch

Class numbers {

X:public static int

Y:public static int

}

Explanation for real world application and how did apply the problem

sleeping barber algorithm in real world : The bank customer service Every customer enters the bank and looks for a customer service available to help him. If the currency service is not available, the customer will wait on the chair, and in the event that there are no chairs, the customer will wait outside the bank.

Starvation: is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time. In heavily loaded computer system, a steady stream of higherpriority processes can prevent a low-priority process from ever getting the CPU, A starvation generally occurs as a result of a deadlock, livelock, or caused by a greedy process. A process which requires completing a task that is unable to gain regular access to the shared resources.

Example1: “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, business class has priority over 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.

Example2: Let's say that you work in a big company. Each day there is rush hour in lunch time - everyone wants to get on the food line first. Your office is on the top floor and the only way to get to the lobby is to use a lift. So, you call the lift and wait... and wait. Your waiting time could be infinite because everyone on the bottom floors is loading the lift, so it never reaches the top! And when it finally does, you decided to go back to your office and eat some leftovers from yesterday

Soultion: Starvation can occur if it is possible that one thread can be delayed infinitely by other threads' requests, A process may never be removed from the semaphore queue in which it is suspended. 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 after waiting for a long time., customers may not decide to approach the barber in an orderly manner, leading to process starvation as some customers never get the chance for a haircut. To handle this problem, we use a first-in-first-out (FIFO) queue So, every time a customer sits in a waiting room, they will be selected by the barber in first come first serve basis

Deadlock is a situation that occurs in OS when any process enters a waiting state because another waiting process is holding the demanded resource. Deadlock is a common problem in multi-processing where several processes share a specific type of mutually exclusive resource known as a soft lock or software

Example 1: this example of a deadlock scenario between 2 processes (Process A and Process B) where each process needs to access the resources File A, File B, and File C. Present a scenario that allows the 2 processes to become deadlocked?

The question provides the scenario - you have 2 processes that both want to lock the same 3 files. Imagine what happens if the one process tries to lock file A then lock file B, and the other process tries to lock file B and then lock file A. In this case; one process can lock file A and the other can lock file B; and neither process can lock their second file (because the other process locked it) so you get deadlock

Soultion: Take two processes A and B. Also, take the general guideline when a process is trying to access a file system, It needs to acquire a resource lock, and then access the file system. When the resource lock is acquired by a process, no other process can acquire the same resource lock, and hence, cannot access the file system. Also, the locks are handled by semaphores with wait forever property, so when a process tries to acquire a lock and it's not available, it's going to wait forever at that point, without proceeding(in a pending state).

Process A is going to access filesystem A. It acquires lock for it, and is working on it. Due to context switching, it goes out of context.

Now process B is going to access filesystem B. It acquires lock for it, and is working on it. Due to context switching, it goes out of context.

Now process A, without releasing the lock for file system A, tries to acquire lock for file system B. The lock is not available, so it is going to enter pending state, and does not proceed further

Now process B is scheduled, which again, without releasing the lock for file system B, tries to acquire lock for file system A. The lock is not available, so it is going to enter pending state, and does not proceed further.

Example 2:

Imagine a rarely used route which has a pedestrian crossing. You are required to stop before the crossing if there is a pedestrian waiting. So, you drive and see that there is a pedestrian waiting in the horizon. You know that you should stop but you hesitate because you don't know if the pedestrian wants to use the crossing or not. Finally, you stop after all but the pedestrian does not walk over the crossing! He doesn't walk because he doesn't know if you're letting him or not. This causes a deadlock situation where you both wait each other to cross the crossing. Also, here a faulty situation is shown: if both sides would respect their rules, there wouldn't be obscurity