Sleeping Barber Problem

Output:-

```
run:
Customer 1 just sat down.
The barber is cutting hair
Customer 1 is getting his hair cut
Customer 2 just sat down.
Customer 3 just sat down.
The barber is cutting hair
Customer 2 is getting his hair cut
Customer 4 just sat down.
Customer 5 just sat down.
Customer 6 just sat down.
Customer 3 is getting his hair cut
The barber is cutting hair
Customer 7 just sat down.
Customer 8 just sat down.
The barber is cutting hair
Customer 4 is getting his hair cut
Customer 9 just sat down.
There are no free seats. Customer 10 has left the barbershop.
There are no free seats. Customer 11 has left the barbershop.
The barber is cutting hair
Customer 5 is getting his hair cut
Customer 12 just sat down.
There are no free seats. Customer 13 has left the barbershop.
The barber is cutting hair
Customer 6 is getting his hair cut
Customer 14 just sat down.
There are no free seats. Customer 15 has left the barbershop.
The barber is cutting hair
Customer 7 is getting his hair cut
The barber is cutting hair
Customer 8 is getting his hair cut
The barber is cutting hair
Customer 9 is getting his hair cut
The barber is cutting hair
Customer 12 is getting his hair cut
The barber is cutting hair
Customer 14 is getting his hair cut
BUILD STOPPED (total time: 57 seconds)
```

Description:-

In computer science, the sleeping barber problem is a classic inter-process communication and synchronization problem between multiple operating system processes. The problem is analogous to that of keeping a barber working when there are customers, resting when there are none and doing so in an orderly manner.

The analogy is based upon a hypothetical barber shop with one barber. The barber has one barber chair and a waiting room with a 5 chairs in it. When the barber finishes cutting a customer's hair, he

dismisses the customer and then goes to the waiting room to see if there are other customers waiting. If there are, he brings one of them back to the chair and cuts his hair. If there are no other customers waiting, he returns to his chair and sleeps in it.

Each customer, when he arrives, looks to see what the barber is doing. If the barber is sleeping, then the customer wakes him up and sits in the chair. If the barber is cutting hair, then the customer goes to the waiting room. If there is a free chair in the waiting room, the customer sits in it and waits his turn. If there is no free chair, then the customer leaves.

This problem is modelled using two semaphores and a mutex. When the barber shows up for work in the morning, he executes the procedure barber, causing him to block on the semaphore customers because it is initially 0. The barber then goes to sleep. He stays asleep until the first customer shows up. When a customer arrives, he executes customer, starting by acquiring mutex to enter a critical region. If another customer enters shortly thereafter, the second one will no be able to do anything until the first one has released mutex. The customer then checks to see if the number of waiting customers is less than the number of chairs. If not, he releases mutex and leaves without a haircut. If there is an available chair, the customer increments the integer variable, waiting. Then he does an Up on the semaphore customers, thus waking up the barber. At this point, the customer and the barber are both awake. When the customer releases mutex, the barber grabs it, does some housekeeping, and begins the haircut. When the haircut is over, the customer exits the procedure and leaves the shop. Unlike our earlier examples, there is no loop for the customer because each one gets only one haircut. The barber loops, however, to try to get the next customer. If one is present, a haircut is given. If not, the barber goes to sleep.