Sleeping barber problem

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's chair in a cutting room and a waiting room containing a number of chairs in it. 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 them back to the chairs and cut their hair. If there are none, he returns to the chair and sleeps in it.

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. If the barber is cutting hair, the customer stays in the waiting room. If there is a free chair in the waiting room, the customer sits in it and waits their turn. If there is no free chair, the customer leaves.

Based on a naïve analysis, the above decisions should ensure that the shop functions correctly, with the barber cutting the hair of anyone who arrives until there are no more customers, and then sleeping until the next customer arrives. In practice, there are a number of problems that can occur that are illustrative of general scheduling problems.

The problems are all related to the fact that the actions by both the barber and the customer (checking the waiting room, entering the shop, taking a waiting room chair, etc.) all take an unknown amount of time. For example, a customer may arrive and observe that the barber is cutting hair, so he goes to the waiting room. While they're on their way, the barber finishes their current haircut and goes to check the waiting room. Since there is no one there (the customer not having arrived yet), he goes back to their chair and sleeps. The barber is now waiting for a customer, but the customer is waiting for the barber. In another example, two customers may arrive at the same time when there happens to be a single seat in the waiting room. They observe that the barber is cutting hair, go to the waiting room, and both attempt to occupy the single chair.

The Sleeping Barber Problem is often attributed to Edsger Dijkstra (1965), one of the pioneers in computer science.

Many possible solutions are available. The key element of each is a mutex, which ensures that only one of the participants can change state at once. The barber must acquire/enforce this mutual exclusion (of room status) 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 either a waiting room chair or the barber chair, and also when they leave the shop because no seats were available. This eliminates both of the problems mentioned in the previous section. 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.

A *multiple sleeping barbers problem* has the additional complexity of coordinating several barbers among the waiting customers.

Implementation

- The following pseudocode guarantees synchronization between barber and customer and is deadlock free, but may lead to starvation of a customer. The problem of starvation can be solved by utilizing a queue where customers are added as they arrive, so that barber can serve them on a first come first served basis (FIFO =>

First In, First Out) The functions wait() and signal() are functions provided by the semaphores. In c-code notation, a wait() is a P() and a signal() is a V().

```
# The first two are mutexes (only 0 or 1 possible)
Semaphore barberReady = 0
Semaphore accessWRSeats = 1      # if 1, the number of seats in the waiting room can be incremented or
decremented
Semaphore custReady = 0          # the number of customers currently in the waiting room, ready to be
served
int numberOfFreeWRSeats = N      # total number of seats in the waiting room

def Barber():
  while true:                    # Run in an infinite loop.
    wait(custReady)              # Try to acquire a customer - if none is available, go to sleep.
    wait(accessWRSeats)          # Awake - try to get access to modify # of available seats, otherwise
sleep.
    numberOfFreeWRSeats += 1     # One waiting room chair becomes free.
    signal(barberReady)          # I am ready to cut.
    signal(accessWRSeats)        # Don't need the lock on the chairs anymore.
    # (Cut hair here.)

def Customer():
  while true:                    # Run in an infinite loop to simulate multiple customers.
    wait(accessWRSeats)          # Try to get access to the waiting room chairs.
    if numberOfFreeWRSeats > 0: # If there are any free seats:
      numberOfFreeWRSeats -= 1   #   sit down in a chair
      signal(custReady)          #   notify the barber, who's waiting until there is a customer
      signal(accessWRSeats)      #   don't need to lock the chairs anymore
      wait(barberReady)          #   wait until the barber is ready
      # (Have hair cut here.)
    else:                        # otherwise, there are no free seats; tough luck --
      signal(accessWRSeats)      #   but don't forget to release the lock on the seats!
      # (Leave without a haircut.)
```

See also

- Producers-consumers problem
- Dining philosophers problem
- Cigarette smokers problem
- Readers–writers problem

References

- *Modern Operating Systems (2nd Edition)* by Andrew S. Tanenbaum (ISBN 0-13-031358-0)
- *The Little Book of Semaphores* by Allen B. Downey, http://greenteapress.com/semaphores
- *Cooperating sequential processes* (http://www.cs.utexas.edu/users/EWD/transcriptions/EWD01xx/EWD123.html) by E.W. Dijkstra. Technical Report EWD-123, 1965, Eindhoven University of Technology, The Netherlands.