WIKIPEDIA
The Free Encyclopedia

Sleeping barber problem

In computer science, the **sleeping barber problem** is a classic inter-process communication and synchronization problem that illustrates the complexities that arise when there are multiple operating system processes.[1]

The problem was originally proposed in 1965 by computer science pioneer Edsger Dijkstra,[2] who used it to make the point that general semaphores are often superfluous.[3]

Problem statement

Imagine a hypothetical barbershop with one barber, one barber chair, and a waiting room with n chairs (n may be 0) for waiting customers. The following rules apply:[4]

- 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[3][5]

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.[6]

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.

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 with a first-in first-out (FIFO) queue. The semaphore would provide two functions: `wait()` and `signal()`, which in terms of C code would correspond to `P()` and `V()`, respectively.

```
# 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

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

References

1. John H. Reynolds (December 2002). "Linda arouses a Sleeping Barber" (http://charm.cs.uiuc.edu/cs498lvk/prog_models/linda/sleeping-barber.pdf) (PDF). *Proceedings of the Winter Simulation Conference*. Vol. 2. San Diego, CA: IEEE. pp. 1804–1808. doi:10.1109/WSC.2002.1166471 (https://doi.org/10.1109%2FWSC.2002.1166471). ISBN 0-7803-7614-5. S2CID 62584541 (https://api.semanticscholar.org/CorpusID:62584541). Retrieved 8 January 2022.
2. Allen B. Downey (2016). *The Little Book of Semaphores* (https://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf) (PDF) (2.2.1 ed.). Green Tea Press. p. 121. Retrieved 8 January 2022.
3. Edsger W. Dijkstra (1965). *Technical Report EWD-123: Cooperating Sequential Processes* (https://www.cs.utexas.edu/users/EWD/transcriptions/EWD01xx/EWD123.html). Eindhoven, The Netherlands: Eindhoven University of Technology. p. 38. Retrieved 8 January 2022.
4. Andrew S. Tanenbaum (2001). *Modern Operating Systems* (https://web.archive.org/web/20220108235906/http://www.microlinkcolleges.net/elib/files/undergraduate/Management/MODERN%20%20OPERATINGSYSTEMSSECOND%20EDITIONby%20Andrew%20S.%20Tanenbaum.pdf) (PDF) (2nd ed.). Upper Saddle River, NJ: Pearson. p. 129. ISBN 9780130313584. Archived from the original (http://www.microlinkcolleges.net/elib/files/undergraduate/Management/MODE

RN%20%20OPERATINGSYSTEMSSECOND%20EDITIONby%20Andrew%20S.%20Tanenbaum.pdf) (PDF) on 8 January 2022. Retrieved 8 January 2022.

5. *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.
6. Fukuda, Munehiro. "Program 2: The Sleeping-Barbers Problem" (http://courses.washington.edu/css503/prog/prog2.pdf) (PDF). *University of Washington*. Retrieved 8 January 2022.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Sleeping_barber_problem&oldid=1211195041"

-