National University of Computer & Emerging Sciences

Karachi Campus

Project: System call for Sleeping Barber

GROUP MEMBERS:

Abdul Hadi (21K-3274)

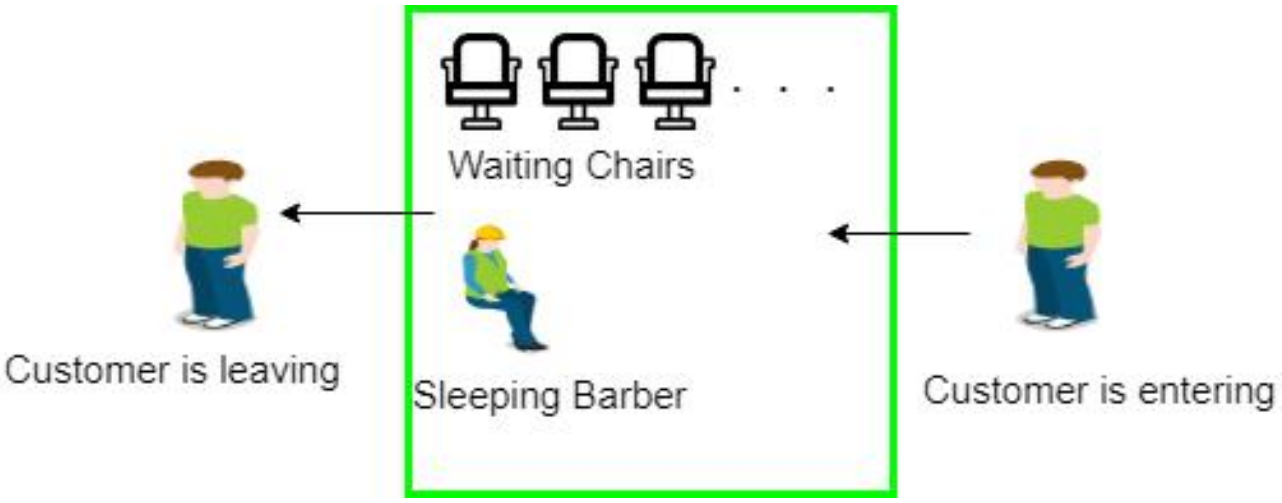
Anwer Saeed (21K-3303)

Rahim Khan (21K-4536)

Teacher: Dr. Nausheen Shoaib
Date of Submission: 12th May 2023

Introduction:

The Sleeping barber is a classic inter process-communication and synchronization problem that Illustrates the complexities that arise when there are multiple operating system process.



Background:

The Sleeping Barber Problem was first introduced by Edsger Dijkstra, a Dutch computer scientist, in 1965. Dijkstra is renowned for his work in computer science and is considered one of the pioneers of the field. He formulated the problem as part of his exploration of concurrent programming and synchronization primitives.

In his seminal paper titled "Solution of a Problem in Concurrent Programming Control," Dijkstra presented the Sleeping Barber Problem as an illustrative example of issues related to resource allocation and synchronization in concurrent systems. The problem has since become a widely studied and discussed topic in computer science and has been used as a basis for exploring various synchronization mechanisms and strategies.

Methodology:

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:

- 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 barber's 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 are also required to indicate the state of the system. For example, one might store the number of people in the waiting room.

```
# 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.)
```

Steps of implementation:

1. Go to the Directory arch/x86/entry/syscalls/ and open the file syscall_64.tbl and add the following line:

   335 64 sleepingbarber sys_sleepingbarber

```
327     64      preadv2                 __x64_sys_preadv2
328     64      pwritev2                __x64_sys_pwritev2
329     common  pkey_mprotect           __x64_sys_pkey_mprotect
330     common  pkey_alloc              __x64_sys_pkey_alloc
331     common  pkey_free               __x64_sys_pkey_free
332     common  statx                   __x64_sys_statx
333     common  io_pgetevents           __x64_sys_io_pgetevents
334     common  rseq                    __x64_sys_rseq
335     64      sleepingbarber          sys_sleepingbarber
#
```

2. Now go to the Directory include/linux/syscalls.h and write:

   asmlinkage long sys_sleepingbarber(void);

```
static inline long ksys_truncate(const char __user *pathname, loff_t length)
{
        return do_sys_truncate(pathname, length);
}

static inline unsigned int ksys_personality(unsigned int personality)
{
        unsigned int old = current->personality;

        if (personality != 0xffffffff)
                set_personality(personality);

        return old;
}
asmlinkage long sys_sleepingbarber(void);
#endif
```

3. Now go to kernel/Makefile and write:

   obj-y:=sleepingbarber.o

4. Finally go to kernel and create a file sleepingbarber.c and add the sleeping barber code there.

5. Finally compile the kernel and install modules and restart.
6. The system call is now successfully added.

Advantages:

1. Scalability: The solution should be able to handle a large number of customers efficiently without causing performance degradation. It should scale well with an increasing number of customers and barber resources.

2. Resource utilization: The solution should optimize the utilization of the barber's time and minimize customer waiting time. It should ensure that the barber is always occupied when there are customers, avoiding idle time.

3. Fairness: The solution should treat customers fairly and ensure that they are served in the order they arrived. It should prevent situations where some customers are consistently prioritized over others, leading to unfairness or starvation.

4. Deadlock and livelock avoidance: The solution should be designed to prevent deadlock, where the system becomes permanently blocked, or livelock, where the system keeps executing but makes no progress. It should guarantee progress and prevent situations where the barber and customers are unable to proceed.

5. Simplicity and maintainability: The solution should be straightforward and easy to understand, implement, and maintain. Complex solutions increase the likelihood of bugs and reduce the overall effectiveness of the solution.

Conclusion:

In conclusion, the Sleeping Barber Problem highlights the challenges of coordinating resources in concurrent systems. By employing appropriate synchronization mechanisms, such as semaphores or condition variables, it is possible to design a solution that ensures proper coordination between the barber and the customers, avoiding potential issues like deadlock or starvation.

Platforms:

This whole problem was solved using C language as it basis which was converted into a code for Kernal of Linux(Ubutu 16.04) where this was compiled

Final Output:

1. Go to Desktop.
2. Add a file test.c
3. Write the code and compile
   gcc -o t test.c
4. Now, run the file to see the output displayed on the terminal.
5. Also, write dmesg on console to see the value after sleepingbarber problem is used as

Output screenshot: