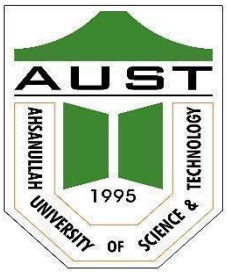
Ahsanullah University of Science & Technology



Department of Computer Science & Engineering

Report on

"Sleeping Barber"

Prepared for:

Mr. Mohammad Moinul Hoque
Associate Professor, CSE

Course Number & Name:

CSE-3213 : Operating System

Prepared by:

Parvez Ahammed
ID: 20200104129
Section: C

Date: 12.07.23

Problem Overview:

The Sleeping Barber Problem is a widely known synchronization problem in computer science as a barbershop has a limited number of waiting room chairs and several customers arrive. Either the barber sleeps or cuts hair. If the barber is sleeping when the customer arrives, they are awakened. When every seat in the waiting area has been filled, the client leaves. Developing a system that provides mutual exclusion, minimizes deadlocks or livelocks, and lets the barber and clients use shared resources is a challenge.

1. The 'sleeping barber problem' consists of one barber, one barber's chair in a cutting room and a waiting room containing a number of chairs in it.

2. 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.

3. If the barber is busy cutting hair, the customer stays in the waiting room and waits for their turn.

4. 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 the them back to the chair and cuts their hair, if there are none, he returns to the chair and sleeps in it.

5. If a customer enters the Shop and all chairs are occupied, the customer leaves the shop.

There might be a scenario in which the customer ends up waiting on the barber and a barber waiting on the customer, which would result in a deadlock. Starvation might happen to a customer who is waiting for a long time when the customers don't follow orders or when the barber calls out a customer randomly.

Solution:

Here are some essential points regarding the Sleeping Barber Problem solution:

- The solution requires the use of synchronization tools like condition variables and mutex locks.
- The barber sleeps while no customers are around and awakens when one does.

- If there are chairs available, customers can choose to wait or go on if all of them are taken.
- The barber attends to one customer at a time, checking to see whether any others are still waiting.
- Mutual exclusion is made sure to avoid disputes when using shared resources.
- Careful resource management and synchronization minimize deadlocks and livelocks.
- The system enables proper interaction between the barber and customers while utilizing the constrained shared resources.

Solution Explanation Using Scenarios:

Scenario-1:

In this scenario, the barbershop has multiple barbers and a waiting room with a limited number of seats. Initially, all the barbers are asleep in their chairs, and the waiting room is empty. When a customer arrives, they check if there are any available seats in the waiting room. If there are, the customer enters the waiting room and increments the counter to indicate their presence. The customer then proceeds to the comfort room. Meanwhile, one of the barbers finishes cutting a customer's hair and becomes available. The barber checks if there are any customers in the waiting room. If there are, the barber acquires the necessary semaphores and calls the next customer, who has been waiting the longest, to the barber's station. The barber and the customer leave the waiting room, and the barber cuts the customer's hair. Once the haircut is done, the customer leaves, and the barber goes back to sleep in their chair. This process continues as new customers arrive, ensuring that the barbers are continuously serving customers and that the waiting room is efficiently managed.

Scenario-2:

In this scenario, the barbershop also has multiple barbers and a waiting room with a limited number of seats. The barbers are initially asleep in their chairs, and the waiting room is empty. When a customer arrives, they check if there are any available seats in the waiting room. If there are, the customer enters the waiting room and waits

for their turn. At the same time, the customer goes to the comfort room. When a barber finishes cutting a customer's hair, they check if there are customers waiting in the queue. If there are, the barber dequeues the next customer in line and starts cutting their hair. Other customers who arrived earlier and are still waiting patiently in the queue will be served in the order of their arrival. Once the haircut is complete, the customer leaves the shop, creating an available seat in the waiting room. The barber checks the queue again and continues serving the next customer if there are any. If the queue is empty, the barber goes back to sleep in their chair until another customer arrives. This scenario ensures fair scheduling and prevents starvation by maintaining the order of customer arrivals and efficiently utilizing the barbers' time.

Scenario-3:

Let's consider a scenario where there are two customers in the barbershop, and initially, the barbers are asleep in their chairs. The first customer enters the waiting room, and one of the barbers wakes up and starts cutting the customer's hair. Barber 1 completes the haircut, and the customer leaves the shop.

After some time, no customers arrive, and the barbers remain idle. Suddenly, two customers arrive in quick succession. Both customers head to the barber's chair. However, if any additional customer arrives while the barbers are busy cutting hair, that customer will be directed to the waiting room. The waiting customers in the waiting room maintain the order in which they arrived, ensuring fairness.

Once the waiting room becomes empty again, the barbers return to their chairs and go back to sleep, waiting for the next customer to arrive. This cycle continues, with customers being served in the order they arrive, and the barbers going back to sleep when there are no customers in the shop.

Appendix:

```
package sleepingbarber;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;
import java.util.concurrent.Semaphore;



public class SleepingBarber {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int numberOfSeatsWR = 5;
        BarberShop barberShop = new BarberShop(numberOfSeatsWR);

        Thread barberThread = new Thread(barberShop::barber);
        barberThread.start();

        // Continuously create and start customer threads
        int customerId = 1;
        while (true) {
            final int finalCustomerId = customerId;
            Thread customerThread = new Thread(() ->
barberShop.customer(finalCustomerId));
            customerThread.start();
            customerId++;

            try {
                // Delay between customer arrivals
                Thread.sleep(2000); // Simulating new customer arrival every 2
seconds
            } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
        }
    }
}


class BarberShop {
        private int numberOfSeatsWR;
        private final Semaphore barberReady;
        private final Semaphore accessWRSeats;
        private final Semaphore custReady;
        private final Queue<Integer> waitingCustomers;
        private final Queue<Integer> orderOfCustomers;
        private final Random random;

        public BarberShop(int numberOfSeatsWR) {
            this.numberOfSeatsWR = numberOfSeatsWR;
            this.barberReady = new Semaphore(0);
            this.accessWRSeats = new Semaphore(1);
            this.custReady = new Semaphore(0);
            this.waitingCustomers = new LinkedList<>();
            this.orderOfCustomers = new LinkedList<>();
            this.random = new Random();
        }

        public void barber() {
        while (true) {
            try {
                accessWRSeats.acquire();
                if (orderOfCustomers.isEmpty()) {
                    System.out.println("No customers waiting. Barber is
sleeping.");
                    accessWRSeats.release();
                    Thread.sleep(2000); // Barber is sleeping, check again after
a delay
                    continue;
                }
                int customerId = orderOfCustomers.peek(); // Get the first
customer in the order
```

```
                if (waitingCustomers.peek() == customerId) {
                    waitingCustomers.poll(); // Remove the customer from the
waiting queue
                    orderOfCustomers.poll(); // Remove the customer from the
order queue
                    numberOfSeatsWR++;
                    barberReady.release();
                    accessWRSeats.release();

                    System.out.println("Customer " + customerId + " arrived and
entered the waiting room.");

                    Thread.sleep(1000); // Simulating hair cutting process
                    System.out.println("Barber is cutting hair for Customer " +
customerId);
                    Thread.sleep(1000); // Simulating hair cutting process
                    System.out.println("Customer " + customerId + " has finished
the haircut and left the shop.");
                } else {
                    accessWRSeats.release();
                    Thread.sleep(500); // Give time for other threads to make
progress
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }


        public void customer(int customerId) {
        try {
            Thread.sleep(random.nextInt(5000)); // Random delay before customer
arrival

            accessWRSeats.acquire();
            if (numberOfSeatsWR > 0) {
                numberOfSeatsWR--;
                waitingCustomers.add(customerId);
                orderOfCustomers.add(customerId); // Maintain the order of
customers
```

```
                custReady.release();
                accessWRSeats.release();
                barberReady.acquire();


                System.out.println("Customer " + customerId + " is getting a
haircut");

                Thread.sleep(2000); // Simulating time taken for haircut
            } else {

                accessWRSeats.release();
                System.out.println("Customer " + customerId + " arrived but found
no seats in the waiting room. Leaving the shop.");
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

}
```