**HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
ICT K67

**IT3070E**

OPERATING SYSTEM


# BARBER SHOP PROBLEM

## PROJECT REPORT

**MEMBER LIST**

| | | |
|---|---|---|
| Nguyen Dinh Duong | 20225966 | duong.nd225966@sis.hust.edu.vn |
| Nguyen Trong Minh Phuong | 20225992 | phuong.ntm225992@sis.hust.edu.vn |
| Dang Van Nhan | 20225990 | nhan.dv225990@sis.hust.edu.vn |
| Bui Ha My | 20225987 | my.bh225987@sis.hust.edu.vn |

**Teacher: PhD. Do Quoc Huy**

**Ha Noi — November 2024**

# CONTENTS

**1 Introduction** **3**

**2 Problem Description** **4**

**3 Problem Analysis** **5**

3.1 Deadlock Prevention . . . . . 6

3.2 Starvation Prevention . . . . . 6

3.3 Process Synchronization . . . . . 6

3.4 Solution Goals . . . . . 7

**4 Solution and Modeling** **7**

4.1 Data Structures Used . . . . . 7

4.2 Synchronization Solution . . . . . 8

4.2.1 Using Semaphores and Mutexes for Synchronization . . . . . 8

4.2.2 Handling Customer and Barber States . . . . . 8

4.3 Algorithm Details . . . . . 8

**5 Implementation and Sample Code** **9**

5.1 How The Algorithm Operates . . . . . 9

5.2 Barber Pseudocode . . . . . 9

5.3 Customer Pseudocode . . . . . 10

5.4 Main Program Pseudocode . . . . . 11

5.5 Explanation of Key Sections . . . . . 11

**6 Testing and Results** **12**

6.1 Testing Methodology . . . . . 12

6.2 Test Scenarios . . . . . 12

6.3 Results . . . . . 12

**7 Discussion** **13**

**8 Conclusion** **14**

## ABSTRACT

The Barber Shop Problem is a classic synchronization challenge that illustrates resource management and process coordination in concurrent systems. This report presents a solution involving a barber, a limited number of waiting chairs, and randomly arriving customers. The solution ensures efficient resource use, prevents deadlock, and avoids customer starvation through the use of semaphores and mutexes to coordinate customer arrivals and service. Testing across scenarios like high customer arrivals and simultaneous entries demonstrated stable performance and fair service order. This project underscores core principles of concurrency and resource allocation, with future improvements aimed at enhancing scalability and prioritization in high-demand environments.


## 1 INTRODUCTION

The Barber Shop Problem is a classical synchronization problem in the field of Operating Systems, commonly used to illustrate the challenges of process synchronization and resource allocation in concurrent environments. It is part of a broader category of problems, like the Dining Philosophers Problem, that are designed to model interactions between processes that share limited resources.

In this scenario, a barber shop has a single barber who serves customers, a finite number of waiting chairs, and customers arriving at random intervals. The goal is to design a solution where:

- The barber is either cutting hair or sleeping when there are no customers waiting.
- Customers, upon arrival, either get their hair cut if the barber is available, sit in a waiting chair if all chairs are not occupied, or leave if no waiting chairs are available.
- The system is free from deadlock and starvation, ensuring that customers are served fairly, and resources are used efficiently.

This problem is significant in the context of Operating Systems as it models real-world scenarios where processes (customers) must share a limited set of resources (barber and chairs) in a coordinated manner. Understanding this problem provides insights into:

- **Process Synchronization:** Ensuring that processes do not interfere with each other's execution and that shared resources are used in a safe and orderly fashion. In this case, synchronization is required to ensure the barber and customers interact without causing conflicts or race conditions.
- **Resource Allocation:** Managing a finite set of resources (waiting chairs and barber availability) such that the system operates smoothly and efficiently. Customers should not be left indefinitely waiting if resources are available, and resources should not remain unused if there are waiting processes.
- **Deadlock and Starvation Avoidance:** Ensuring that the barber and customers do not end up in a deadlocked state where they are both waiting on each other to act, and that no customer is indefinitely delayed from receiving service. This requires careful structuring of the interactions to maintain system responsiveness.

The Barber Shop Problem is often solved using semaphores or mutexes to manage the interactions between the barber and customers:

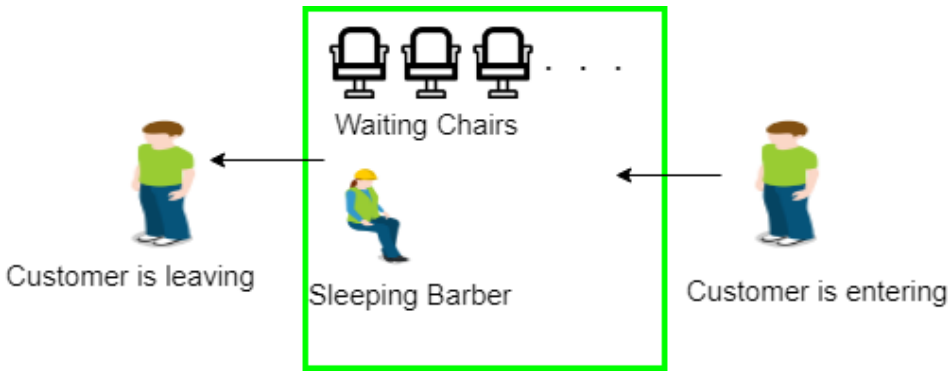


Figure 1: Problems Description

- A semaphore or mutex can be used to control access to the barber's chair, ensuring that only one customer is served at a time.
- A counter-based semaphore can track the number of available waiting chairs, allowing customers to determine if they should wait or leave.
- Additional synchronization mechanisms can ensure that the barber sleeps when there are no customers, waking up only when a customer is ready to be served.

Through solving this problem, we gain a deeper understanding of the principles of process synchronization, the importance of avoiding deadlock and starvation, and the utility of semaphores and mutexes in coordinating interactions within a concurrent system. The Barber Shop Problem serves as a foundational example for tackling synchronization challenges in more complex systems, making it a critical topic in Operating System and concurrent programming courses.

## 2 Problem Description

The Barber Shop Problem is a classic synchronization challenge in which we simulate the environment of a barber shop. In this problem, the barber shop has one barber who serves customers in a designated barber's chair, a limited number of waiting chairs for customers who arrive when the barber is busy, and customers who arrive at random intervals. The goal is to create a solution where the barber and customers interact in a way that maintains system efficiency, avoids deadlock, and prevents starvation. The elements of the problem setup include:

- **Barber:** The barber can be in one of two states — either actively cutting a customer's hair or sleeping if there are no customers waiting. When there are no customers, the barber goes to sleep to save resources. If a customer arrives while the barber is sleeping, the barber should wake up and serve them immediately.
- **Customers:** Customers arrive at the shop at unpredictable intervals. Each customer wants to get a haircut upon arrival. If the barber is available, they immediately proceed to the barber's chair for a haircut. If the barber is busy and there are empty waiting chairs, the customer will take a seat and wait their turn. However, if all waiting chairs are occupied, the customer leaves the shop without receiving service, as they have no place to wait.
- **Waiting Chairs:** There is a limited number of waiting chairs in the barber shop, which represent the capacity of customers that can wait if the barber is occupied. These chairs act as a buffer for the customers who arrive when the barber is busy. The number of

waiting chairs can significantly impact system behavior, as fewer chairs may result in more customers leaving unserved, while a larger number of chairs could potentially keep the barber busy without idle time.

The objective of the Barber Shop Problem is to achieve a system that adheres to the following principles:

- **Mutual Exclusion:** Only one customer can occupy the barber's chair at any given time. This ensures that only one customer is being served by the barber while others wait in line or in the waiting area.
- **Fair Service Order:** Customers are served on a first-come, first-served basis to ensure fairness. This can be achieved by organizing the waiting area as a queue where customers are attended to in the order they arrive.
- **Deadlock Avoidance:** The system must avoid a state where the barber and customers are mutually waiting for each other, preventing any further actions. For example, the barber should not be indefinitely asleep if there are customers in the waiting area, and customers should not wait indefinitely if the barber is available.
- **Starvation Prevention:** No customer should experience indefinite waiting, especially if they have already taken a seat in the waiting area. This helps ensure that every customer who arrives and waits their turn will eventually get served.

The solution to this problem typically involves using synchronization mechanisms like semaphores and mutex locks to manage access to shared resources:

- A **barber chair semaphore** to control access to the barber's chair, ensuring only one customer at a time is being served by the barber.
- A **waiting chairs semaphore** to track the availability of waiting chairs. This allows the customers to check if there is room in the waiting area before deciding to wait or leave.
- A **customer semaphore** to signal the barber when a customer arrives, ensuring that the barber wakes up from sleep only when there is a customer waiting.
- A **barber semaphore** to signal the customer when the barber is ready to serve them, ensuring that customers do not approach the barber's chair until they are called.

This model creates a controlled environment where the barber and customers can interact seamlessly without conflict, deadlock, or starvation. By effectively managing the waiting chairs and ensuring orderly access to the barber, the system achieves a balance that maximizes efficiency and customer satisfaction. The Barber Shop Problem is therefore a valuable example of process synchronization, resource allocation, and concurrency control in Operating Systems.

## 3 Problem Analysis

The Barber Shop Problem presents a variety of challenges related to process synchronization, resource management, and concurrent process interactions. The goal of the problem is to design a system where the barber and customers can interact in an orderly and efficient way, ensuring that resources are utilized effectively while avoiding deadlock and starvation. Analyzing this problem requires understanding key synchronization concepts and developing mechanisms that address the potential issues of deadlock and starvation.

## 3.1 Deadlock Prevention

Deadlock in the Barber Shop Problem would occur if both the barber and customers were waiting on each other in a manner that prevents further actions. Specifically:

- The **barber** should not remain in a sleep state when there are customers waiting to be served. If the barber is asleep, they must be woken up by the arrival of a new customer.
- Conversely, **customers** should not be left in a waiting state if the barber is available to serve them. A customer must proceed to the barber's chair when called upon to avoid situations where both parties are waiting indefinitely.

To prevent deadlock, a signaling mechanism is needed:

- A **customer semaphore** can signal the barber whenever a customer arrives, ensuring that the barber wakes up if asleep.
- A **barber semaphore** can be used by the barber to signal when they are ready to serve a customer, ensuring that customers approach the barber's chair only when it is their turn.

Through this approach, we ensure that the barber and customers are synchronized in a way that prevents both from waiting indefinitely.

## 3.2 Starvation Prevention

Starvation occurs when certain customers are unable to access the barber's service due to the continuous arrival of new customers or due to unfair queuing policies. To prevent starvation:

- A **FIFO (First In, First Out) queue** should be maintained for the waiting chairs, ensuring that customers are served in the order of their arrival. This prevents newer customers from taking priority over those who have been waiting longer.
- By limiting the number of waiting chairs, we can ensure that customers do not experience indefinite waiting if they choose to enter the shop. If all chairs are occupied, a new customer simply leaves, avoiding an overload of the system.

Using a queue-based approach and managing waiting chairs fairly allows us to guarantee that each customer who finds a seat in the waiting area will eventually get a haircut, effectively preventing starvation.

## 3.3 Process Synchronization

Process synchronization is a key challenge in the Barber Shop Problem, as we need to coordinate the actions of the barber and multiple customers in a way that prevents race conditions and ensures orderly resource usage. The solution requires careful control over access to shared resources such as the barber's chair and waiting chairs. Key synchronization mechanisms include:

- **Mutual exclusion** on the barber's chair: Ensuring that only one customer occupies the chair at a time, which can be achieved using a semaphore or mutex to lock the barber's chair.
- **Waiting area management** using a semaphore for the chairs: A semaphore can keep track of the number of available waiting chairs. If a chair is available, the customer takes a seat and waits; otherwise, they leave the shop. This ensures that the waiting area capacity is respected, preventing overcrowding.
- **Signaling synchronization** between the barber and customers: The barber and customers need to signal each other when certain actions are required. For example, a customer signals the barber when they are ready, and the barber signals the customer when they are prepared to start a haircut. This coordinated signaling allows for seamless transitions between states.

In summary, effective synchronization mechanisms must ensure that:

- The barber works only when there are customers to serve, remaining idle when the shop is empty.
- Customers wait their turn in the order of arrival, provided there is space available in the waiting area.
- The system prevents deadlock and starvation by ensuring orderly and fair access to the barber's services.

## 3.4 Solution Goals

The objectives of the Barber Shop Problem solution can be summarized as follows:

- **Efficient Resource Usage:** Ensure that the barber remains busy when there are customers waiting, reducing idle time and maximizing service efficiency.
- **Fairness:** Customers are served in the order they arrive, and those who have secured a waiting chair are guaranteed service, providing a fair experience.
- **System Stability:** The system must handle random arrivals and departures of customers without deadlock or starvation, allowing for consistent and reliable operation.

The Barber Shop Problem thus serves as a foundational example in Operating Systems for designing synchronization solutions that maintain fairness, prevent deadlock, and ensure efficient utilization of shared resources in concurrent environments.

# 4 Solution and Modeling

## 4.1 Data Structures Used

In order to efficiently manage synchronization between the barber and the customers in the Barber Shop Problem, the following data structures are primarily used:

- **Semaphores:** Semaphores are crucial in this problem for managing access to shared resources. In particular, we use semaphores to signal when the barber is ready and when customers are waiting.

- **Mutexes:** Mutexes ensure that only one thread can access a critical section of code at a time, preventing race conditions. For example, we use mutexes to manage access to the waiting chairs to prevent multiple customers from trying to occupy the same chair.

## 4.2 Synchronization Solution

To solve the synchronization issues between the barber and customers, we implement a solution using semaphores and mutexes. The goal is to avoid deadlock and starvation while maintaining an efficient flow of customer service.

### 4.2.1 Using Semaphores and Mutexes for Synchronization

The following semaphores are implemented:

- `customer_ready`: This semaphore keeps track of the number of customers ready to be served.
- `barber_ready`: This semaphore signals when the barber is available to cut hair.

The mutex `access_waiting_chairs` is also used to control access to the waiting chairs. This prevents race conditions by ensuring that only one customer at a time can check for available seats or occupy a chair.

### 4.2.2 Handling Customer and Barber States

The algorithm includes specific handling for different scenarios:

1. **When a customer arrives and there is an available chair:** The customer waits on the `customer_ready` semaphore, signaling to the barber that they are ready. They then occupy a chair if available.
2. **When a customer arrives and all chairs are occupied:** The customer leaves as there are no available waiting chairs.
3. **When the barber is idle:** The barber waits on the `customer_ready` semaphore for a customer to arrive. When a customer signals, the barber proceeds with the haircut.

## 4.3 Algorithm Details

The algorithm operates as follows:

1. When a customer arrives, they first acquire the `access_waiting_chairs` mutex to check if a chair is available.
2. If a chair is available, the customer increments the `customer_ready` semaphore and releases the mutex, then waits for the `barber_ready` signal.
3. The barber, upon being signaled by `customer_ready`, starts the haircut process.
4. After completing the haircut, the barber signals `barber_ready`, indicating readiness for the next customer.

The careful control of semaphores and mutexes ensures smooth transitions between customer arrivals, waiting, and haircuts, preventing deadlock and ensuring efficiency.

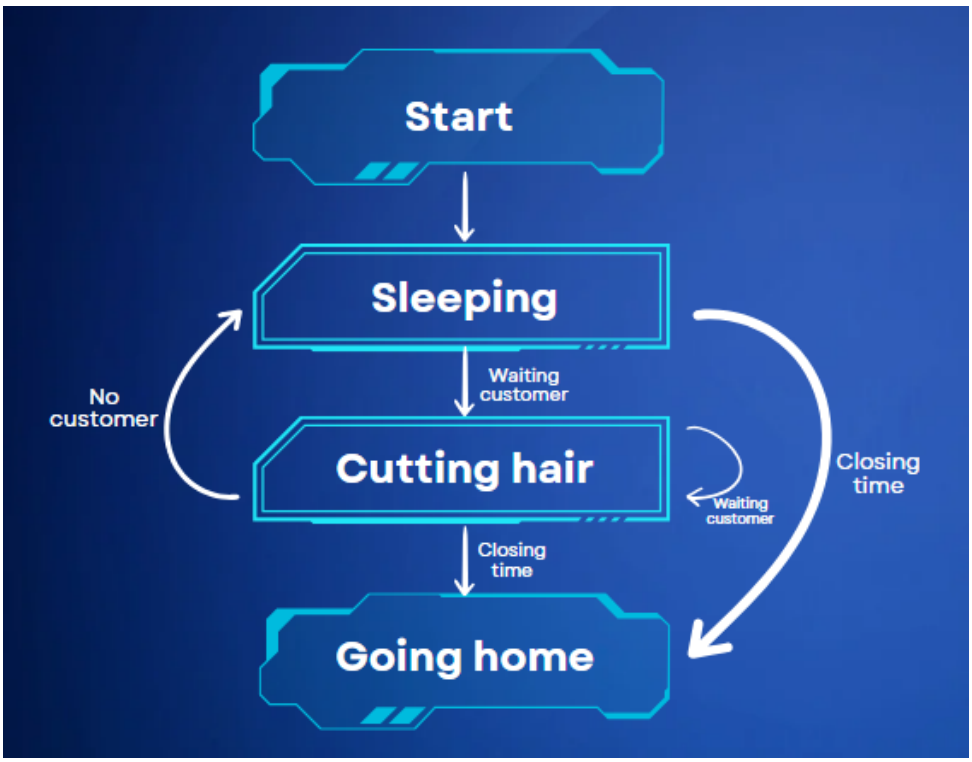


Figure 2: Solution Flow

# 5 Implementation and Sample Code

The following pseudocode outlines the solution for the Barber Shop Problem, showing the main logic behind customer and barber interactions. The pseudocode uses semaphores and mutexes to manage synchronization effectively.

## 5.1 How The Algorithm Operates

1. The barber begins by waiting for a customer signal indicating that a customer is ready for a haircut.
2. When a customer arrives, they check if there is an available waiting chair:
   - If a chair is available, the customer sits, signals readiness to the barber, and waits for the barber to be ready.
   - If no chairs are available, the customer leaves the shop.
3. When the barber is ready and a customer is waiting, the barber performs the haircut and then signals that they are ready for the next customer.
4. The process repeats with customers arriving at random intervals and either joining the waiting area or leaving if the shop is full.

## 5.2 Barber Pseudocode

```
PROCEDURE Barber():
    WHILE true:
        // CRITICAL SECTION: Check queue status
        ACQUIRE mutex
```

```
            IF waiting_queue is empty:
                RELEASE mutex
                IF not program_running:
                    EXIT

                PRINT "Barber is sleeping"
                WAIT customer_ready    // Block until customer arrives
                CONTINUE              // Recheck queue after waking

            // Get next customer from queue
            current_customer = waiting_queue.dequeue()
        RELEASE mutex

        // Barber is now serving customer
        PRINT "Barber serving customer #{current_customer}"

        // Critical section for haircut service
        ACQUIRE barber_working
            SLEEP random(3,6)    // Simulate haircut
        RELEASE barber_working

        PRINT "Finished customer #{current_customer}"
```

## 5.3 Customer Pseudocode

```
PROCEDURE Customer(id):
    IF not program_running:
        PRINT "Shop closed, customer #{id} leaving"
        RETURN

    // CRITICAL SECTION: Try to enter waiting room
    ACQUIRE mutex
        IF waiting_queue.size() == MAX_CHAIRS:
            RELEASE mutex
            PRINT "No chairs available, customer #{id} leaving"
            RETURN

        // Add customer to waiting queue
        waiting_queue.enqueue(id)
        last_customer_time = current_time()
        queue_position = waiting_queue.size()
    RELEASE mutex

    PRINT "Customer #{id} took seat #{queue_position}"
    SIGNAL customer_ready    // Wake up barber if sleeping

    // Wait for and receive haircut
    WAIT barber_working     // Wait for barber to be free
    RECEIVE_HAIRCUT()
```

```
    SIGNAL barber_working   // Release barber for next customer
```

## 5.4 Main Program Pseudocode

```
PROCEDURE ShopController():
    // Initialize synchronization objects
    mutex = CREATE new Lock()
    customer_ready = CREATE new Semaphore(0)
    barber_ready = CREATE new Semaphore(0)
    barber_working = CREATE new Semaphore(1)

    // Start barber thread
    barber_thread = CREATE Thread(Barber)
    START barber_thread

    // Start monitoring threads with shared resource access
    START Thread(ClosingChecker, program_running, last_customer_time)
    START Thread(UserInputHandler, program_running)

    customer_id = 1
    WHILE program_running:
        // Generate new customers
        SLEEP random(1,3)
        IF program_running:
            customer_thread = CREATE Thread(Customer, customer_id)
            START customer_thread
            INCREMENT customer_id

    // Cleanup phase
    WAIT FOR all customers to finish
    SIGNAL customer_ready    // Wake up barber for final check
    WAIT FOR barber_thread to finish
```

## 5.5 Explanation of Key Sections

The solution includes several key sections and elements that ensure efficient and synchronized operations:

- **Barber Function:** This function ensures the barber only works when a customer is available. By waiting on `customer_ready`, the barber avoids unnecessary waiting. The function locks `access_waiting_chairs` to manage the `waiting_customers` count safely, then signals `barber_ready` to indicate that the barber is prepared for the next customer.
- **Customer Function:** The customer function ensures that a customer only waits if there is an available chair. If there is, the customer signals readiness through `customer_ready` and waits for `barber_ready` to begin their haircut. This structure avoids deadlock by managing customer wait times and barber readiness effectively.

- **Main Program:** The main program sets up and initializes semaphores, mutexes, and variables. It also creates a barber thread and customer threads at random intervals, simulating real-world conditions of customer arrivals in a barbershop.
- **Synchronization:** The use of semaphores and mutexes maintains synchronization between customers and the barber, ensuring smooth and efficient operation. The `access_waiting_chairs` mutex prevents multiple customers from accessing the `waiting_customers` variable simultaneously, reducing race conditions.

This structured approach to the Barber Shop Problem ensures that customers and the barber interact seamlessly, maximizing efficiency and preventing resource conflicts.

# 6 Testing and Results

## 6.1 Testing Methodology

To verify the functionality and efficiency of the Barber Shop solution, several test scenarios were designed, focusing on various customer arrival rates and the capacity of the waiting area. The tests were carried out in the following manner:

- **Controlled Arrival Intervals:** Customers were generated at both fixed and random intervals to simulate real-world conditions. Tests included high, medium, and low arrival frequencies to observe the system's response to varying loads.
- **Varying Waiting Chair Capacity:** The number of waiting chairs was adjusted to test how different capacities impact customer service and waiting times. This included both low-capacity and high-capacity scenarios.
- **Concurrency Handling:** Multiple customers arriving nearly simultaneously were tested to assess if the synchronization mechanisms (semaphores and mutexes) properly prevented race conditions and deadlocks.
- **Stress Testing:** The system was stress-tested by rapidly increasing the number of customer threads to ensure that it remained stable under heavy load.

## 6.2 Test Scenarios

1. **Scenario 1:** Customers arrive at a steady rate with enough chairs to accommodate all arrivals.
2. **Scenario 2:** Customers arrive at high frequency, with fewer chairs available, testing how well the system handles customers leaving due to full capacity.
3. **Scenario 3:** Irregular and random arrival intervals, testing the response to unpredictable loads.
4. **Scenario 4:** Simultaneous arrival of multiple customers, testing the effectiveness of the locking mechanism in preventing race conditions.

## 6.3 Results

1. **Scenario 1:** In this scenario, the system performs steadily as customers arrive at a consistent rate, with enough waiting chairs available for all. The barber does not experience long waiting times, and most customers receive service. However, as customers

arrive continuously and waiting chairs fill up, some customers have to leave due to lack of available space. This demonstrates that the system efficiently handles queue capacity limits, ensuring synchronization and preventing conflicts when adding new customers.

2. **Scenario 2:** In this scenario, customers arrive frequently and continuously, while the waiting chair capacity is limited to only three chairs. As a result, many customers must leave due to lack of available seating, yet the system handles this situation well, maintaining synchronization as customers enter or leave the queue. The chair status is accurately updated, and the system encounters no conflict in processing service turns.
3. **Scenario 3:** In this scenario, customers arrive at irregular intervals, and due to no arrivals within 4 seconds, the system automatically closes. This demonstrates that the system is designed to adjust its status based on customer demand, helping to reduce resource waste during idle periods. The automatic closing mechanism functions reliably and accurately, ensuring that the shop remains open only when there are customers.
4. **Scenario 4:** In this scenario, multiple customers arrive simultaneously, and the system effectively uses locking mechanisms to prevent conflicts when adding customers to the queue. When the waiting chairs are full, new customers are forced to leave due to a lack of available space, but all customers in the queue are served in the correct order. This shows that the system maintains consistency and reliability in a high-demand environment with concurrent requests.

# 7 Discussion

The Barber Shop Problem solution demonstrated effective synchronization and resource management across various load scenarios. However, several insights and challenges emerged during the testing and implementation phases:

- **Handling Peak Loads:** Under high customer arrival rates, especially when the waiting area has limited chairs, some customers were forced to leave. While this aligns with real-world constraints, it highlights the need for optimized scheduling when high throughput is desired.
- **Synchronization Efficiency:** The use of semaphores and mutexes proved effective in maintaining system stability. However, there is an inherent trade-off between response time and processing overhead when using these mechanisms, especially under peak loads. Advanced approaches, such as priority-based scheduling, could potentially improve this.
- **Deadlock and Starvation Prevention:** The solution successfully avoided deadlock by ensuring that customer and barber interactions were managed strictly through controlled signals. Starvation was also minimized, but testing showed that under certain conditions (e.g., limited chairs with high arrival rates), customers may experience longer wait times or leave due to full capacity.
- **Real-world Applications and Scalability:** This solution has potential applications in resource management scenarios such as CPU scheduling in operating systems. To enhance scalability for larger systems, dynamic resource allocation techniques could be integrated to adjust resources based on demand in real time.

In summary, the solution demonstrated reliable performance and effective resource management. Future improvements could explore adaptive mechanisms to dynamically adjust capacity and optimize service order based on priority or arrival time.

# 8 Conclusion

The Barber Shop Problem provides an excellent framework for understanding process synchronization and resource management in a concurrent system. Through this project, we successfully developed and tested a solution that enables smooth interactions between the barber and customers, demonstrating effective use of semaphores and mutexes to avoid deadlock and minimize starvation.

The project highlighted several key aspects:

- **Effective Synchronization:** By employing semaphores to manage customer and barber interactions, the solution achieved efficient process coordination, ensuring that no two processes attempted to access critical resources simultaneously, thus preventing race conditions.
- **Resource Management:** The controlled use of a waiting area with limited chairs provided valuable insights into managing limited resources in a shared environment, showing how constraints impact system performance and customer satisfaction.
- **Applicability in Real-World Systems:** The concepts applied in this problem, such as synchronization and resource allocation, are essential in designing operating systems and other concurrent processing systems, particularly in CPU scheduling and multithreaded environments.

In future work, further enhancements could focus on optimizing for high-load scenarios and implementing adaptive mechanisms to better handle dynamic customer arrival rates. Additionally, integrating a priority system could reduce wait times for specific customers and offer a more responsive experience under varying conditions.

In conclusion, this project provided a robust solution to the Barber Shop Problem, reinforcing core principles of concurrency in computing and demonstrating practical applications of these principles in real-world systems.

# References

[1] Rishabh Agarwal. The barbershop problem, 2024.

[2] GeeksForGeeks. Sleeping barber problem in process synchronization, 2023.

[3] Tanenbaum. Modern operating system.