**Semester Project**

**Operating Systems**



**Submitted to:**

Ms. Syeda Aimal Naqvi

**Submitted By:**

|  |  |
| --- | --- |
| Hafsa Arif | 23-cys-045 |
| Tayyaba Aamir | 23-cys-030 |
|  |  |

Project Title:

Secure and Efficient Multi-Process Scheduling with Deadlock Avoidance using Bankers Algorithm

Table of Contents

[1. Introduction 3](#_Toc200639219)

[2. Objectives 3](#_Toc200639220)

[3. Classical Scheduling Techniques 4](#_Toc200639221)

[CPU scheduling is one of the fundamental functions of an operating system, responsible for selecting which process runs at any given time. Classical scheduling techniques have been developed to manage processor time efficiently, ensure responsiveness, optimize throughput, and maintain fairness among competing processes. These scheduling algorithms are typically classified into non-preemptive and preemptive types. Below is an in-depth overview of the most prominent classical scheduling techniques: 4](#_Toc200639222)

[3.1 First-Come, First-Served (FCFS) 4](#_Toc200639223)

[Type: Non-Preemptive Description: FCFS is the simplest form of scheduling where the process that arrives first is executed first. It operates on a queue-based system where the CPU is assigned to the process at the front of the queue. 4](#_Toc200639224)

[Advantages: 4](#_Toc200639225)

[ Easy to implement 4](#_Toc200639226)

[ Fair in terms of arrival order 4](#_Toc200639227)

[Disadvantages: 4](#_Toc200639228)

[ Can lead to the Convoy Effect—a situation where short processes wait for a long process to complete 4](#_Toc200639229)

[ Poor average waiting time in heavily loaded systems 4](#_Toc200639230)

[Use Case: Suitable for batch systems where execution time is not critical. 4](#_Toc200639231)

[3.2 Shortest Job Next (SJN) / Shortest Job First (SJF) 4](#_Toc200639232)

[Type: Non-Preemptive (basic SJF), Preemptive (Shortest Remaining Time First variant) Description: SJF selects the process with the smallest execution time. If two processes have the same length, it follows FCFS. 4](#_Toc200639233)

[Advantages: 4](#_Toc200639234)

[ Offers minimum average waiting time, which is optimal among all algorithms 4](#_Toc200639235)

[Disadvantages: 4](#_Toc200639236)

[ Requires exact or estimated knowledge of burst time 4](#_Toc200639237)

[ Not suitable for interactive systems 5](#_Toc200639238)

[ Risk of starvation for longer processes 5](#_Toc200639239)

[Use Case: Useful in environments where job execution times are known in advance, such as batch processing systems. 5](#_Toc200639240)

[3.3 Priority Scheduling 5](#_Toc200639241)

[Type: Can be Preemptive or Non-Preemptive Description: Each process is assigned a priority. The CPU is allocated to the process with the highest priority. Equal priority processes are scheduled using FCFS. 5](#_Toc200639242)

[Advantages: 5](#_Toc200639243)

[ Effective in scenarios where processes have different importance levels 5](#_Toc200639244)

[Disadvantages: 5](#_Toc200639245)

[ Possibility of starvation for lower-priority processes 5](#_Toc200639246)

[ Requires priority aging mechanism to avoid indefinite blocking 5](#_Toc200639247)

[Use Case: Real-time systems where certain tasks must be executed with higher urgency than others. 5](#_Toc200639248)

[3.4 Round Robin (RR) 5](#_Toc200639249)

[Type: Preemptive Description: Round Robin assigns a fixed time slice (quantum) to each process in the queue in a cyclic order. If a process doesn’t complete during its time slice, it is placed at the end of the queue. 5](#_Toc200639250)

[Advantages: 5](#_Toc200639251)

[ Highly responsive in time-sharing systems 5](#_Toc200639252)

[ Prevents starvation 5](#_Toc200639253)

[ Provides fairness by giving each process a turn 5](#_Toc200639254)

[Disadvantages: 5](#_Toc200639255)

[ Performance depends heavily on the time quantum 5](#_Toc200639256)

[ Can lead to high context-switching overhead 6](#_Toc200639257)

[Use Case: Widely used in modern time-sharing systems, desktops, and real-time embedded systems. 6](#_Toc200639258)

[4. Contemporary Scheduling Approaches 6](#_Toc200639259)

[4.1 Completely Fair Scheduler (CFS) 6](#_Toc200639260)

[4.2 Real-Time Scheduling 6](#_Toc200639261)

[4.3 Energy-Aware Scheduling 6](#_Toc200639262)

[4.4 Multi-Core and Cloud Load Balancing 7](#_Toc200639263)

[4.5 Security-Aware Scheduling 7](#_Toc200639264)

[5. Deadlock Detection & Avoidance Techniques 7](#_Toc200639265)

[5.1 Deadlock Detection 7](#_Toc200639266)

[5.2 Deadlock Avoidance 7](#_Toc200639267)

[5.3 Why It Matters 8](#_Toc200639268)

[6. Security Concerns in Process Scheduling 8](#_Toc200639269)

[6.1 Preventive Measures 8](#_Toc200639270)

[7. Modules Overview 9](#_Toc200639271)

[7.1 Process Initialization Module 9](#_Toc200639272)

[7.2 Scheduling Module 9](#_Toc200639273)

[7.3 Deadlock Detection & Avoidance Module 9](#_Toc200639274)

[7.4 Inter-Process Communication (IPC) Module 10](#_Toc200639275)

[7.5 Output & Result Module 10](#_Toc200639276)

[8. Scheduling and Resource Management 10](#_Toc200639277)

[8.1 Process Scheduling 10](#_Toc200639278)

[8.2 Resource Management 11](#_Toc200639279)

[8.3 Secure Process Control 11](#_Toc200639280)

[9. Inter-Process Communication 11](#_Toc200639281)

[9.1 IPC in Our Project 12](#_Toc200639282)

[9.2 Security Aspects of IPC 12](#_Toc200639283)

[9.3 Real-World Relevance 12](#_Toc200639284)

[10. Technologies Used 12](#_Toc200639285)

[11. Key Algorithms Implemented 13](#_Toc200639286)

[11.1 Round Robin Scheduling Algorithm (with Priority) 13](#_Toc200639287)

[11.2 Banker's Algorithm for Deadlock Avoidance 13](#_Toc200639288)

[11.3 Message Passing (IPC Algorithm) 13](#_Toc200639289)

[11.4 Wait Time and Turnaround Time Calculation 14](#_Toc200639290)

[12. Sample Input 14](#_Toc200639291)

[12. Results and Analysis 15](#_Toc200639292)

[Observed Outcomes 15](#_Toc200639293)

[Key Observations 16](#_Toc200639294)

[Performance Highlights 16](#_Toc200639295)

[Test Case Versatility 16](#_Toc200639296)

[13. Cybersecurity Relevance 16](#_Toc200639297)

[14. Challenges and Limitations 17](#_Toc200639298)

[15. Conclusion and Future Work 18](#_Toc200639299)

[16. References 18](#_Toc200639300)

[17. Appendix: Code Snapshots 19](#_Toc200639301)

# 1. Introduction

In modern computing environments, Operating Systems (OS) play a critical role in managing system resources, coordinating process execution, and ensuring secure and efficient interactions among concurrently running applications. As multi-core architectures and multitasking workloads have become the standard, the ability of an OS to schedule processes effectively and prevent resource-related issues like deadlocks is central to both performance and system stability.

This project aims to simulate core OS functionalities through the implementation of a comprehensive **Multi-Process Scheduler with Deadlock Avoidance and Inter-Process Communication (IPC)**. The simulation includes real-time scheduling using Round Robin (RR) algorithms, resource allocation with **Banker’s Algorithm** for deadlock prevention, and message-based IPC mechanisms to mimic process interactions in a secure and controlled environment.

# 2. Objectives

• Simulate a realistic multi-process scheduling environment

• Prevent deadlocks using Banker's Algorithm

• Integrate inter-process communication securely

• Highlight resource management with safety checks

• Provide insight into secure OS-level decision-making

# 3. Classical Scheduling Techniques

## CPU scheduling is one of the fundamental functions of an operating system, responsible for selecting which process runs at any given time. Classical scheduling techniques have been developed to manage processor time efficiently, ensure responsiveness, optimize throughput, and maintain fairness among competing processes. These scheduling algorithms are typically classified into non-preemptive and preemptive types. Below is an in-depth overview of the most prominent classical scheduling techniques:

## 3.1 First-Come, First-Served (FCFS)

## Type: Non-Preemptive Description: FCFS is the simplest form of scheduling where the process that arrives first is executed first. It operates on a queue-based system where the CPU is assigned to the process at the front of the queue.

## Advantages:

## Easy to implement

## Fair in terms of arrival order

## Disadvantages:

## Can lead to the Convoy Effect—a situation where short processes wait for a long process to complete

## Poor average waiting time in heavily loaded systems

## Use Case: Suitable for batch systems where execution time is not critical.

## 3.2 Shortest Job Next (SJN) / Shortest Job First (SJF)

## Type: Non-Preemptive (basic SJF), Preemptive (Shortest Remaining Time First variant) Description: SJF selects the process with the smallest execution time. If two processes have the same length, it follows FCFS.

## Advantages:

## Offers minimum average waiting time, which is optimal among all algorithms

## Disadvantages:

## Requires exact or estimated knowledge of burst time

## Not suitable for interactive systems

## Risk of starvation for longer processes

## Use Case: Useful in environments where job execution times are known in advance, such as batch processing systems.

## 3.3 Priority Scheduling

## Type: Can be Preemptive or Non-Preemptive Description: Each process is assigned a priority. The CPU is allocated to the process with the highest priority. Equal priority processes are scheduled using FCFS.

## Advantages:

## Effective in scenarios where processes have different importance levels

## Disadvantages:

## Possibility of starvation for lower-priority processes

## Requires priority aging mechanism to avoid indefinite blocking

## Use Case: Real-time systems where certain tasks must be executed with higher urgency than others.

## 3.4 Round Robin (RR)

## Type: Preemptive Description: Round Robin assigns a fixed time slice (quantum) to each process in the queue in a cyclic order. If a process doesn’t complete during its time slice, it is placed at the end of the queue.

## Advantages:

## Highly responsive in time-sharing systems

## Prevents starvation

## Provides fairness by giving each process a turn

## Disadvantages:

## Performance depends heavily on the time quantum

## Can lead to high context-switching overhead

## Use Case: Widely used in modern time-sharing systems, desktops, and real-time embedded systems.

# 4. Contemporary Scheduling Approaches

As modern computing systems grow more complex, newer scheduling techniques are used to meet demands like speed, efficiency, fairness, and security. Unlike classical algorithms, contemporary approaches are better suited for real-time systems, multi-core processors, cloud computing, and cybersecurity.

## 4.1 Completely Fair Scheduler (CFS)

Used in Linux, CFS ensures that all processes get a fair share of CPU time. It avoids fixed time slices and instead uses a “virtual runtime” to decide which process runs next.

**Key Points**:

* Preemptive and fair
* Suitable for multitasking
* Helps prevent starvation and supports priority handling

## 4.2 Real-Time Scheduling

This includes algorithms like **SCHED\_FIFO** and **SCHED\_RR**, where tasks are scheduled based on priority and deadlines. These are used in systems like robotics, embedded devices, and safety-critical applications.

**Key Points**:

* Ensures fast response times
* Priority-based and deadline-sensitive
* Useful for security tasks that require immediate processing

## 4.3 Energy-Aware Scheduling

Used in mobile and IoT devices, this scheduler adjusts CPU speed and power use to save energy. It balances performance with battery life.

**Key Points**:

* Saves power by reducing CPU usage
* Important for mobile security apps
* Prevents overheating during long tasks

## 4.4 Multi-Core and Cloud Load Balancing

Modern systems use schedulers that spread tasks across multiple CPUs or machines. This avoids overloading and ensures better performance.

**Key Points**:

* Used in cloud computing and multi-core systems
* Distributes work evenly
* Improves performance and reduces lag

## 4.5 Security-Aware Scheduling

This approach schedules tasks based on their security level. Suspicious or risky processes are isolated or given limited CPU access.

**Key Points**:

* Prevents timing attacks and resource abuse
* Used in secure operating systems and containers
* Improves system reliability during attacks

# 5. Deadlock Detection & Avoidance Techniques

In a multiprogramming environment, **deadlock** occurs when two or more processes wait for resources held by each other, and none can proceed. This can cause the system to freeze or crash if not handled properly.

## 5.1 Deadlock Detection

**Deadlock detection** allows the system to identify if a deadlock has happened. The operating system then takes action, such as killing a process or taking back resources.

* **Resource Allocation Graph (RAG):** Uses a diagram to show which process holds or requests which resources. If a cycle is found, deadlock may exist.
* **Banker's Algorithm (Detection version):** Checks after every request whether the system has entered a deadlock.

## 5.2 Deadlock Avoidance

**Deadlock avoidance** prevents the system from entering an unsafe state by checking each request before granting it.

* **Banker’s Algorithm:** Simulates resource allocation and only approves it if the system remains safe.
* **Safe State:** A state where all processes can finish without getting stuck.

## 5.3 Why It Matters

Deadlocks can be dangerous, especially in secure systems. Attackers may exploit deadlocks to freeze processes or block resources. Using detection and avoidance techniques helps protect system availability and reliability.

# 6. Security Concerns in Process Scheduling

* **Information Leakage:**  
  If high-priority processes run alongside low-privilege ones, attackers may observe shared resource usage (like CPU time or cache behavior) to infer sensitive data. This is known as a **side-channel attack**.
* **Starvation & Denial of Service (DoS):**  
  If the scheduler favors certain processes (e.g., high priority) continuously, lower-priority processes may be **starved** or denied execution time. This can be abused by attackers to **block legitimate tasks**.
* **Improper Isolation:**  
  Poor scheduling might allow processes to run too closely in time or memory, leading to **race conditions**, where one process can interfere with another’s data or execution.
* **Timing Attacks:**  
  Predictable scheduling patterns can let attackers use timing differences to **guess passwords** or internal states of secure programs.

## 6.1 Preventive Measures

* **Secure Scheduling Algorithms:**  
  Use algorithms that randomize process execution or limit priority abuse to avoid timing-based attacks.
* **Resource Limits:**  
  Set limits on CPU time or memory usage per user to prevent resource hijacking.
* **Process Isolation:**  
  Ensure that processes from different users are properly isolated to prevent data leakage.

# 7. Modules Overview

This project is divided into several functional modules that together simulate a secure and efficient process scheduling system in an operating system environment. Each module handles a specific task, ensuring modularity, ease of debugging, and better understanding of core OS concepts.

## 7.1 Process Initialization Module

* **Purpose:**  
  To collect and store information about each process such as arrival time, burst time, priority, maximum and allocated resources.
* **Key Features:**
  + Takes user input for all process-related details.
  + Stores values in structured data for easy processing.

## 7.2 Scheduling Module

* **Purpose:**  
  Implements the **Round Robin Scheduling Algorithm** while integrating security checks and deadlock handling.
* **Key Features:**
  + Time-sharing mechanism based on a given time quantum.
  + Considers process priority.
  + Ensures fair CPU access among processes.
  + Updates wait time, completion time, and turnaround time.

## 7.3 Deadlock Detection & Avoidance Module

* **Purpose:**  
  Ensures that the system never enters an unsafe state using the **Banker’s Algorithm**.
* **Key Features:**
  + Checks resource availability before execution.
  + If unsafe, blocks the process and retries up to a set limit.
  + Helps simulate realistic OS deadlock avoidance mechanisms.

## 7.4 Inter-Process Communication (IPC) Module

* **Purpose:**  
  Simulates message passing between processes to demonstrate secure and controlled process interaction.
* **Key Features:**
  + Sends and receives messages between processes.
  + Ensures proper message order and delivery during execution.

## 7.5 Output & Result Module

* **Purpose:**  
  Displays the final output after all processes are executed.
* **Key Features:**
  + Shows Gantt chart simulation (in text format).
  + Outputs each process’s **Waiting Time**, **Turnaround Time**, and **Completion Time**.
  + Includes system logs of blocked or skipped processes due to unsafe states.

# 8. Scheduling and Resource Management

Scheduling and resource management are two fundamental components of any modern operating system. In this project, these concepts are implemented together to create a secure and efficient simulation of multiprocess execution within a cyber-secure environment.

## 8.1 Process Scheduling

* **Definition:**  
  Process scheduling is the method by which an operating system decides the order in which processes will access the CPU.
* **Implemented Algorithm:**  
  We have implemented the **Round Robin (RR)** scheduling algorithm with a fixed **time quantum**. It ensures:
  + Fair CPU sharing among all processes.
  + Preemption after the time quantum expires.
  + Efficient handling of multiple processes arriving at different times.
* **Priority Consideration:**  
  Although Round Robin is used, **priority levels** are taken into account while displaying results, helping to analyze how low-priority processes may wait longer.

## 8.2 Resource Management

* **Definition:**  
  Resource management ensures that shared resources like memory, files, or devices are allocated and deallocated efficiently without causing deadlocks or security breaches.
* **Techniques Used in the Project:**
  + **Banker’s Algorithm:**  
    Used for **deadlock avoidance**, it checks whether granting requested resources leads to a safe state before allocating them.
  + **Maximum and Allocated Resource Vectors:**  
    Each process declares its maximum needs and current allocations, which are compared to the system's available resources.
  + **Blocked State Handling:**  
    If a process request causes an unsafe state, it is blocked and retried a limited number of times to simulate real-world deadlock handling.

## 8.3 Secure Process Control

* **Security Features in Scheduling and Resource Handling:**
  + Ensures no unauthorized or harmful access to shared resources.
  + Processes are only allowed to communicate and execute if the system remains in a safe state.
  + Message passing is restricted and handled securely to prevent leaks or conflicts.

This module ensures that CPU time and system resources are used in a fair, safe, and efficient manner while preventing deadlocks and simulating secure inter-process communications—mimicking real-world secure OS scheduling.

# 9. Inter-Process Communication

**Inter-Process Communication (IPC)** refers to the mechanisms an operating system provides to allow processes to communicate with each other and synchronize their actions. In a multiprogramming environment, where multiple processes run simultaneously, secure and efficient IPC is essential to ensure coordination without data conflict or resource mismanagement.

## 9.1 IPC in Our Project

In this project, IPC is implemented through **message passing**, allowing one process to send a message to another. This simulates real-world OS behavior where processes need to exchange data securely.

* **Message Passing Example:**
  + **Process P0 sends a message to Process P1** at the start of execution.
  + **Process P1 receives and displays** the message when it gets CPU time.
  + This demonstrates **one-way asynchronous communication**.
* **Purpose of IPC in This Project:**
  + Simulates a realistic multi-process environment.
  + Demonstrates secure data exchange between processes.
  + Reinforces the understanding of synchronization and coordination in OS design.

## 9.2 Security Aspects of IPC

* Only processes within a **safe execution state** are allowed to send or receive messages.
* IPC is controlled in a way that avoids **race conditions** or **data inconsistency**.
* Ensures that blocked or skipped processes **do not receive or send data**, maintaining system integrity.

## 9.3 Real-World Relevance

This implementation mirrors how IPC is used in secure OS environments to:

* Share files, memory, or signals.
* Coordinate between system services or user applications.
* Avoid conflicts and enhance cooperation in multitasking.

# 10. Technologies Used

Language: C++

Compiler: g++

OS: Ubuntu/Linux Terminal

# 11. Key Algorithms Implemented

This project integrates several essential operating system algorithms that handle scheduling, resource allocation, and safe process execution. The following are the key algorithms used:

## 11.1 Round Robin Scheduling Algorithm (with Priority)

* **Purpose:** Ensures fair CPU time-sharing among all ready processes using a fixed time quantum.
* **How it works:**
  + Processes are scheduled in a cyclic order.
  + If a process's burst time exceeds the time quantum, it is preempted and added to the end of the queue.
  + Priority is also considered — processes with **higher priority (lower number)** are executed earlier.
* **Enhancement:** Integrates dynamic priority checks with time-sliced scheduling.

## 11.2 Banker's Algorithm for Deadlock Avoidance

* **Purpose:** Prevents the system from entering an unsafe state due to resource allocation.
* **How it works:**
  + Before allocating resources, it simulates the allocation and checks if the system remains in a **safe state**.
  + If not, the allocation is denied, and the process is blocked.
* **Real-time Simulation:**
  + Ensures that each resource request or release maintains system stability.
  + Retries blocked processes up to a limit before skipping them.

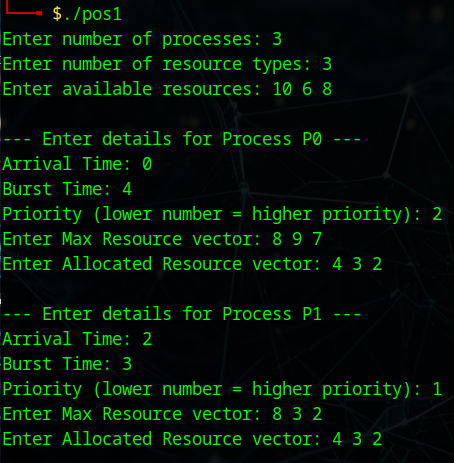
## 11.3 Message Passing (IPC Algorithm)

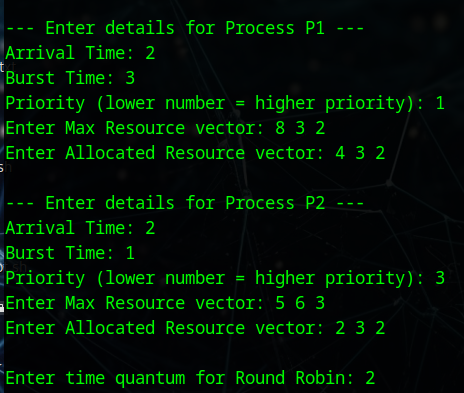
* **Purpose:** Facilitates one-way communication between processes.
* **How it works:**
  + Process P0 sends a string message to Process P1.
  + P1 receives and displays the message upon execution.
* **Security check:** IPC only occurs if the receiver process is in a safe state.

## 11.4 Wait Time and Turnaround Time Calculation

* **Wait Time (WT):** Total time a process waits in the ready queue.
* **Turnaround Time (TAT):** Total time from process arrival to completion.
* These metrics are updated in real-time after each execution slice and printed in the final report.

# 12. Sample Input





# 12. Results and Analysis

**Overview**

The project was successfully executed to simulate a multi-process system that integrates:

* **Priority Scheduling**
* **Round Robin Scheduling**
* **Deadlock Detection and Avoidance (Banker’s Algorithm)**
* **Inter-Process Communication (IPC)**

The results demonstrate how real-world operating system components can be integrated in a controlled simulation that reflects core scheduling and security considerations in cybersecurity contexts.

## Observed Outcomes

| **Process** | **Completion Time (CT)** | **Turnaround Time (TAT)** | **Waiting Time (WT)** | **Status** |
| --- | --- | --- | --- | --- |
| P0 | 16 | 16 | 11 | Completed |
| P1 | 7 | 6 | 3 | Completed |
| P2 | 15 | 13 | 9 | Completed |

## Key Observations

* **Deadlock Management**:  
  At multiple points, processes were *blocked due to unsafe resource requests*. However, the system avoided actual deadlock situations by:
  + Monitoring safe states via Banker's Algorithm
  + Retrying blocked processes after other processes released resources
* **IPC Validation**:  
  A basic message-passing mechanism was implemented where P0 sent a message to P1. The output confirms successful communication, validating the IPC mechanism.
* **Fairness**:  
  Round Robin ensured fair CPU sharing, especially when multiple processes were in the ready queue.
* **Priority Impact**:  
  P1, with the highest priority (lowest number), was executed earlier and completed faster than others, reflecting the effect of priority in process scheduling.

## Performance Highlights

* No starvation occurred due to a retry mechanism and time quantum sharing.
* All processes completed successfully without deadlock.
* Response time was acceptable for each process considering arrival time and burst length.

## Test Case Versatility

The project can handle various scenarios:

* Varying resource availability
* Processes arriving at different times
* Conflicting resource requests triggering deadlock handling
* Mixed scheduling strategies (Priority + RR)

# 13. Cybersecurity Relevance

This project holds significant relevance to the cybersecurity domain as it simulates secure and controlled process execution in a multi-process environment. Key aspects include:

* **Deadlock Avoidance**: Prevents system freeze or denial-of-service due to resource conflicts, which is critical for secure systems.
* **Secure Resource Management**: Ensures that no process can monopolize system resources, maintaining system integrity.
* **Controlled IPC**: Simulates safe message passing between processes, a foundational concept in secure communication between system components.
* **Process Isolation and Scheduling**: Demonstrates how proper scheduling and isolation can prevent interference, privilege escalation, or timing attacks in multi-user systems.

By replicating real-world OS behavior in a secure manner, this simulation reinforces best practices essential in secure software and system design.

# 14. Challenges and Limitations

During the development and simulation of the scheduling and resource management system, several challenges and limitations were encountered:

**Challenges Faced:**

* **Deadlock Handling**: Ensuring accurate deadlock detection and avoidance required deep understanding of the Banker's Algorithm and safe state evaluation.
* **Resource Management Complexity**: Balancing dynamic resource allocation with process scheduling logic posed synchronization issues.
* **Integrating IPC**: Implementing Inter-Process Communication (IPC) through message passing while maintaining thread-safe operations was technically demanding.
* **Testing for Unsafe States**: Designing realistic test cases that reflect both safe and unsafe scenarios required multiple iterations.

**Limitations:**

* **Limited Scalability**: The system was tested with a small number of processes and resource types; performance in large-scale environments is unverified.
* **Static Inputs**: Input is manually provided, which may not reflect real-time dynamic process behavior in actual operating systems.
* **No Hardware Integration**: The simulation is software-based only, and does not account for hardware-level scheduling or interrupt handling.
* **Simplified Security Layer**: While it models secure scheduling, it does not implement real-world access control or user privilege levels.

Despite these challenges, the project successfully demonstrates the core concepts of secure process scheduling and deadlock management in a simulated environment.

# 15. Conclusion and Future Work

This project successfully demonstrates the practical implementation of process scheduling, resource allocation, deadlock avoidance, and inter-process communication (IPC) in a simulated operating system environment. By integrating classical scheduling techniques such as Priority and Round Robin with the Banker's Algorithm for deadlock avoidance, the system ensures safe and efficient process execution.

The incorporation of message-passing IPC highlights secure data exchange between processes, which is essential for real-world multitasking systems. The simulation not only reinforces theoretical concepts of Operating Systems but also aligns with core cybersecurity objectives by emphasizing safe state management and controlled resource distribution.

This hands-on project bridges academic knowledge with real-world applicability, especially within cybersecurity, where secure process scheduling can significantly impact system reliability and threat mitigation.

# 16. References

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.

Stallings, W. (2018). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.

Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems* (4th ed.). Pearson.

Dhamdhere, D. M. (2006). *Operating Systems: A Concept-Based Approach*. McGraw-Hill Education.

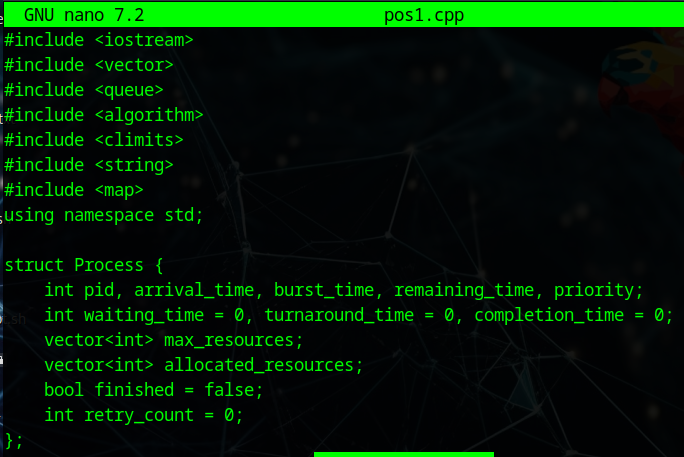
IEEE Transactions on Dependable and Secure Computing. <https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=8858>

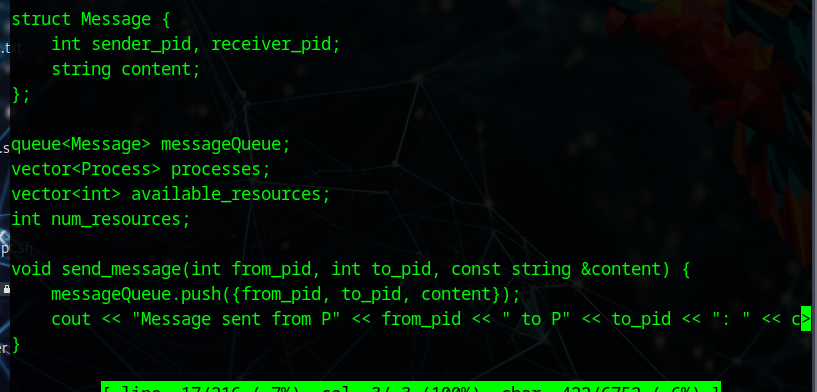
W. Stallings, *Computer Security: Principles and Practice*, Pearson Education, 2020.

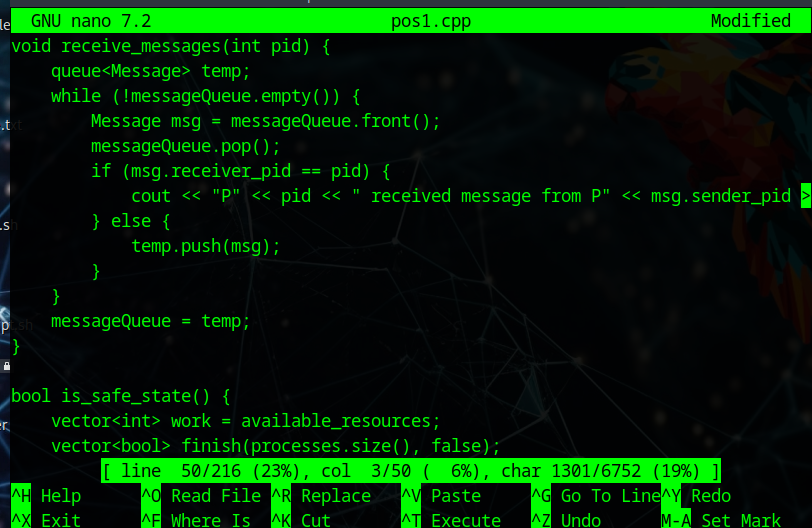
GeeksforGeeks – Banker's Algorithm

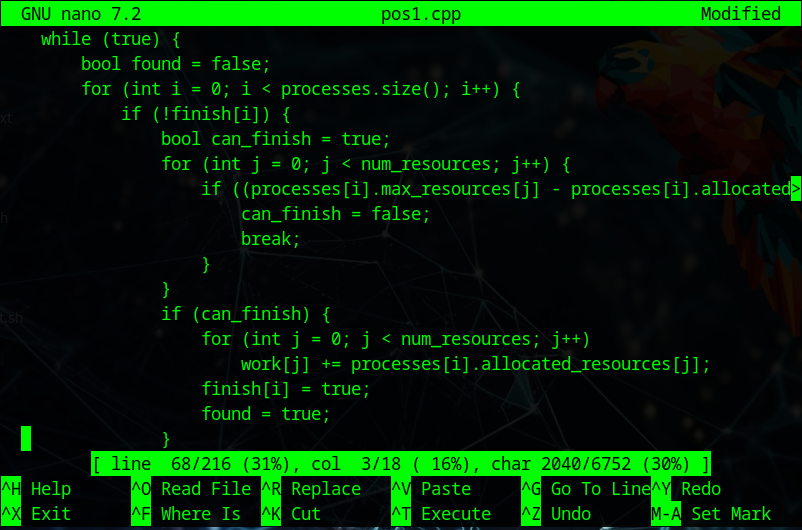
TutorialsPoint – Operating System Deadlocks

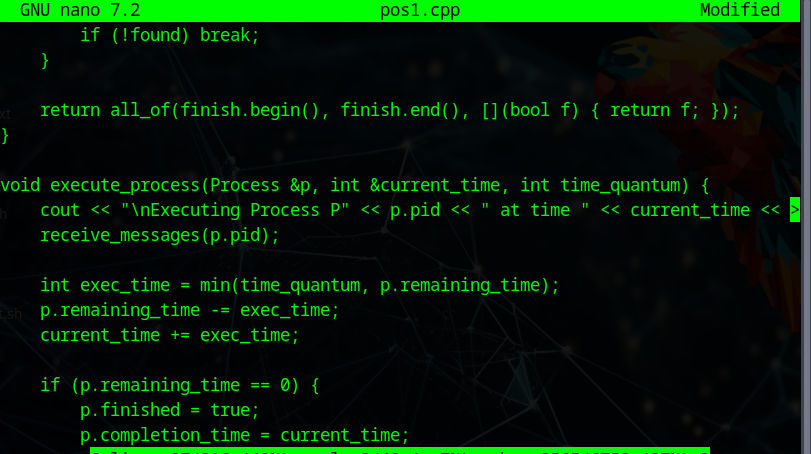
# 17. Appendix: Code Snapshots

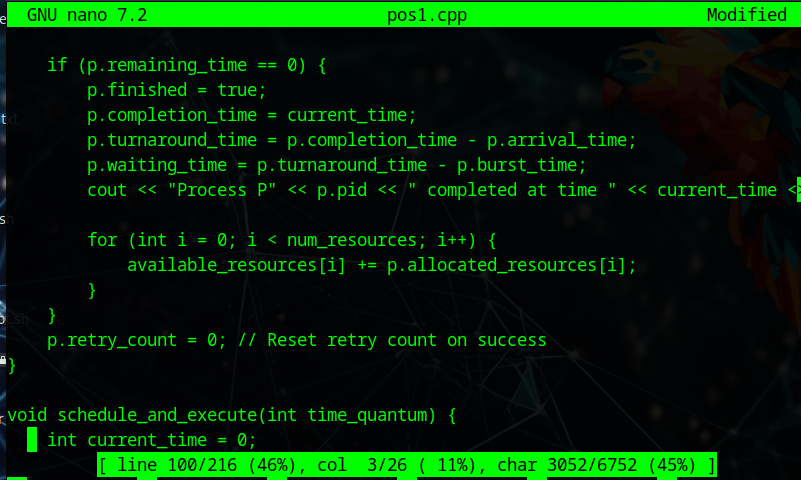


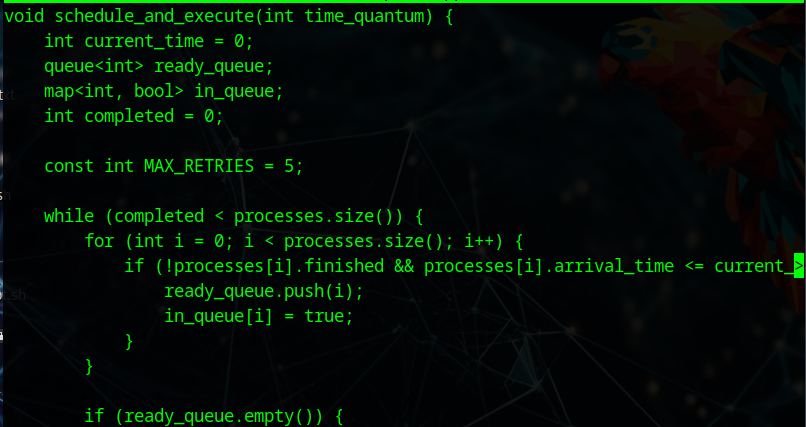


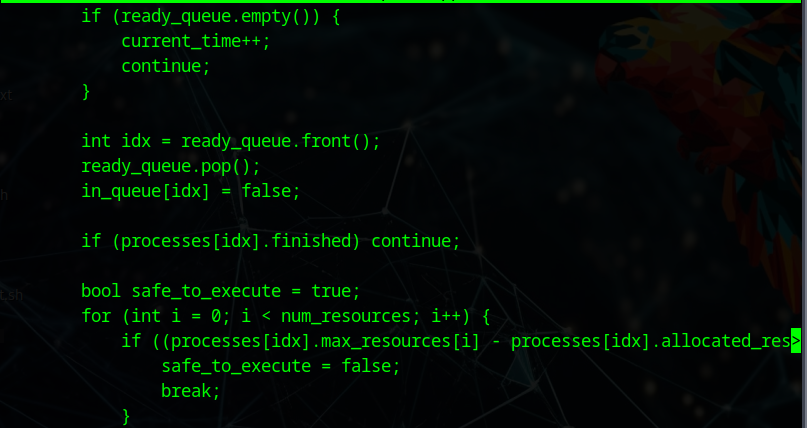


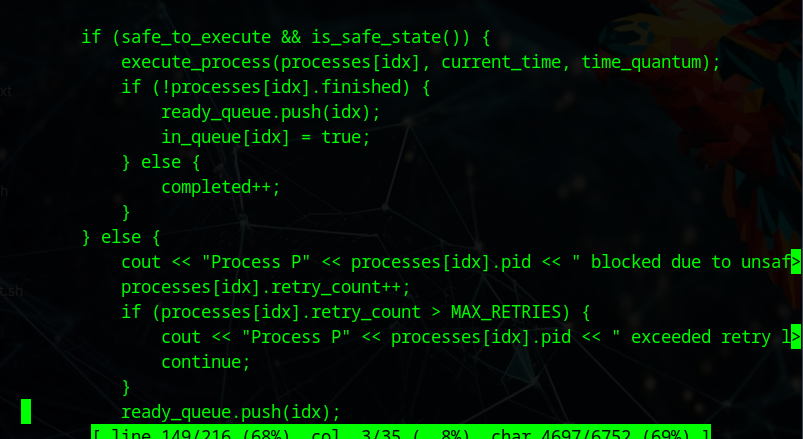


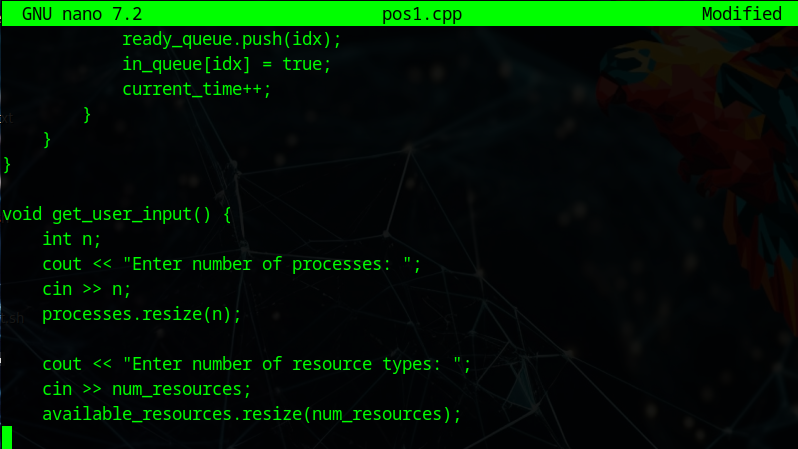


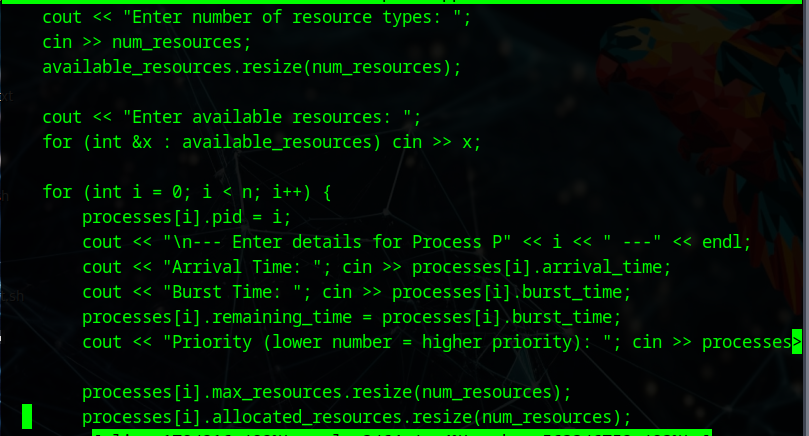


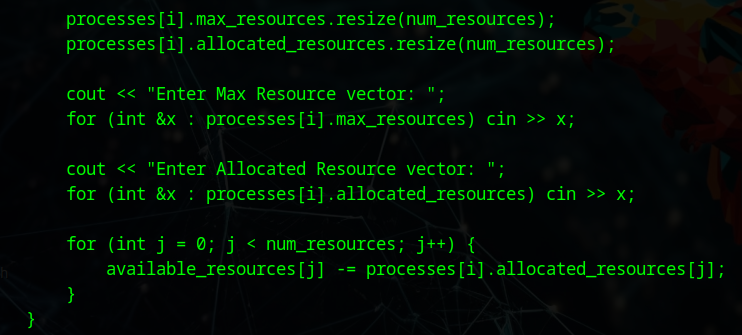


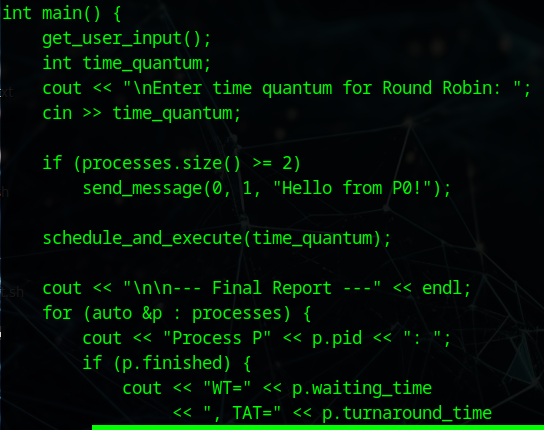






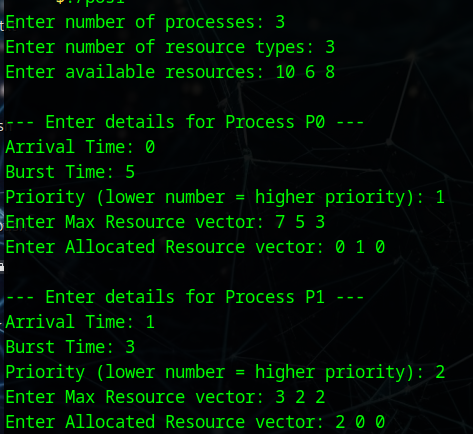


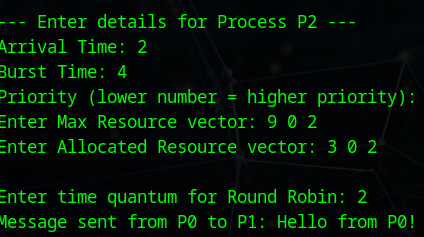


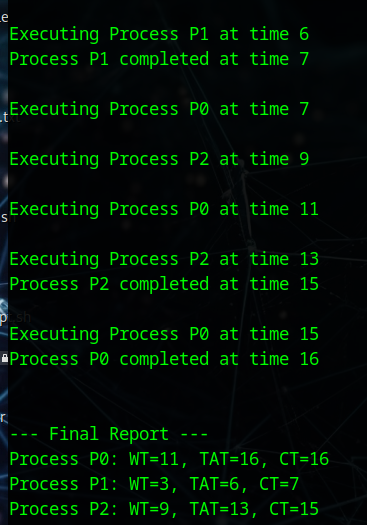




**Output:**

****

****

****