

**Submitted To :** Saima Siddique Tashfia

Lecturer, Dept. of CSE

UITS.

**Submitted By :**  Md. Mehadi Hasan, ID : 2125051003

Maheru Tabassum Ohana, ID : 2125051015

Sowrobh Bhuiyan, ID : 2125051026

Mofidul Haque Nibir, ID : 2125051020

**Department :** CSE, **Batch :** 50, **Section :** 7A

**Course Code :** CSE412

**Course Title :** Operating Systems LAB.

**“Project Report”**

**Project Title :** Process Scheduler Simulator.

**Date of Submission :** 9 December, 2024.



**Project Title :** Process Scheduler Simulator.

#### **Introduction :**

#### This project is an Operating System (OS) Lab Project focused on simulating Process Scheduling Algorithms using Cygwin, a Linux-like environment for Windows. By leveraging Cygwin, Linux commands and scripts were seamlessly executed on a Windows system, bridging the gap between two operating systems.

The core of the project is a Process Scheduler Simulator, designed to help users understand and simulate various CPU scheduling techniques. The simulator functions like a calculator for process scheduling, offering a user-friendly menu with five scheduling options. After running the program, users can select from the following algorithms:

1. First-Come, First-Served (FCFS)
2. Non-Preemptive Shortest Job First (SJF)
3. Preemptive Shortest Job First (Preemptive SJF)
4. Round Robin Scheduling
5. Priority Scheduling

Upon choosing an algorithm, the program computes and displays the scheduling outcomes based on the user's inputs, such as process arrival times, burst times, and other parameters specific to the chosen method.

This simulator not only demonstrates the implementation of these scheduling algorithms but also provides insights into their operational differences, making it a valuable educational tool for OS concepts.

**Background :**

The idea for this project originated from our Operating System Lab classes, where we were introduced to various CPU scheduling algorithms as part of our coursework. During these classes, we were taught how fundamental scheduling methods, including First-Come, First-Served (FCFS), Non-Preemptive Shortest Job First (SJF), Preemptive SJF, and Round Robin Scheduling, function and differ from one another. These algorithms form the backbone of CPU scheduling, playing a critical role in managing process execution in an operating system.

We chose this topic for our project because it addresses a practical challenge: integrating multiple scheduling techniques into a single tool for easier understanding and comparison. By creating a Process Scheduler Simulator, we aimed to consolidate these algorithms into one program, making them more accessible for students, educators, and anyone interested in OS concepts.

This project not only simplifies the learning process but also provides users with an interactive way to experiment with the algorithms. The inclusion of Priority Scheduling further enriches the scope, offering a broader perspective on CPU scheduling techniques. Through this project, we hope to create a tool that enhances understanding while showcasing our skills in scripting and problem-solving.

**Methodology :**

The implementation of this project involved several steps to ensure a functional and user-friendly Process Scheduler Simulator. The methodology outlines the approach to designing, coding, and executing the project.

#### **1. Development Environment -**

We used Cygwin, a Linux-like environment for Windows, to execute shell scripting and Linux commands seamlessly. This choice allowed us to utilize Linux-based concepts while working on a Windows platform, ensuring compatibility with the scripting requirements of scheduling algorithms.

#### **2. Design and Implementation -**

The simulator was designed as a menu-driven shell script, enabling users to select from the available scheduling algorithms. Each algorithm is implemented as a separate function within the script. The core logic of the simulator is encapsulated in the main() function:

# Main function to call scheduling algorithms

main() {

echo "Choose the scheduling algorithm:"

echo "1. FCFS"

echo "2. Non-Preemptive SJF"

echo "3. Preemptive SJF"

echo "4. Round Robin"

echo "5. Priority Scheduling"

read choice

case $choice in

1) fcfsScheduling ;;

2) sjfScheduling ;;

3) preemptiveSjfScheduling ;;

4) roundRobinScheduling ;;

5) priorityScheduling ;;

\*) echo "Invalid choice!" ;;

esac

}

main

* **Menu Selection:** The main() function displays a menu for users to choose one of the five algorithms.
* **Algorithm Execution:** Based on the user's choice, the corresponding function (e.g., fcfsScheduling) is invoked to execute the algorithm. Each function handles input parameters, computation logic, and output display for the respective algorithm.
* **Error Handling:** Invalid inputs are managed with an error message, ensuring a smooth user experience.

#### **3. Algorithm Implementations -**

Each scheduling algorithm is implemented with the appropriate logic:

* **FCFS:** Processes are executed in the order of arrival.
* **Non-Preemptive SJF:** Shortest jobs are executed first, without interruption.
* **Preemptive SJF**: Shorter jobs can preempt currently running processes.
* **Round Robin:** Processes are executed in a cyclic order based on a time quantum.
* **Priority Scheduling:** Processes are executed based on priority levels.

#### **4. Testing and Validation -**

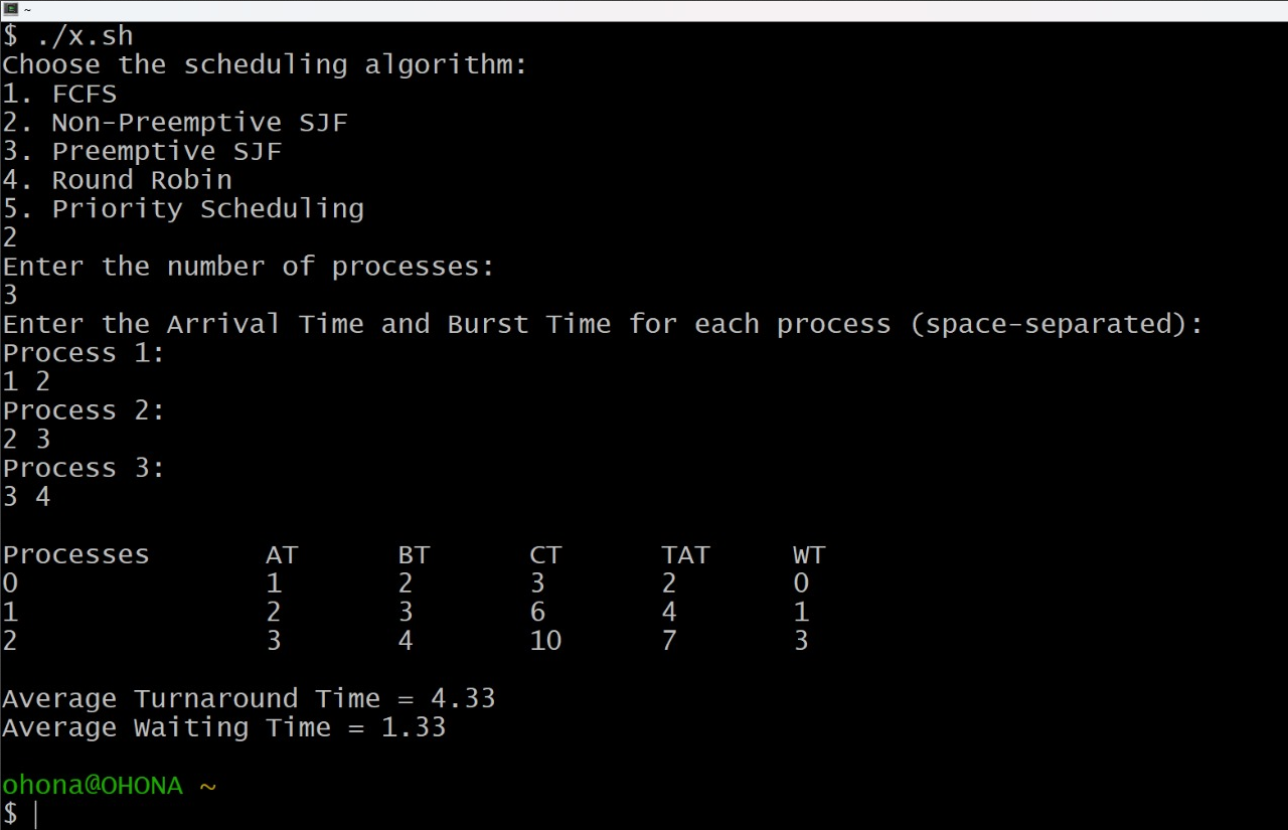
The script was tested with a variety of input cases to ensure correctness for all scheduling algorithms. Boundary conditions, such as invalid inputs and zero processes, were also considered during testing.

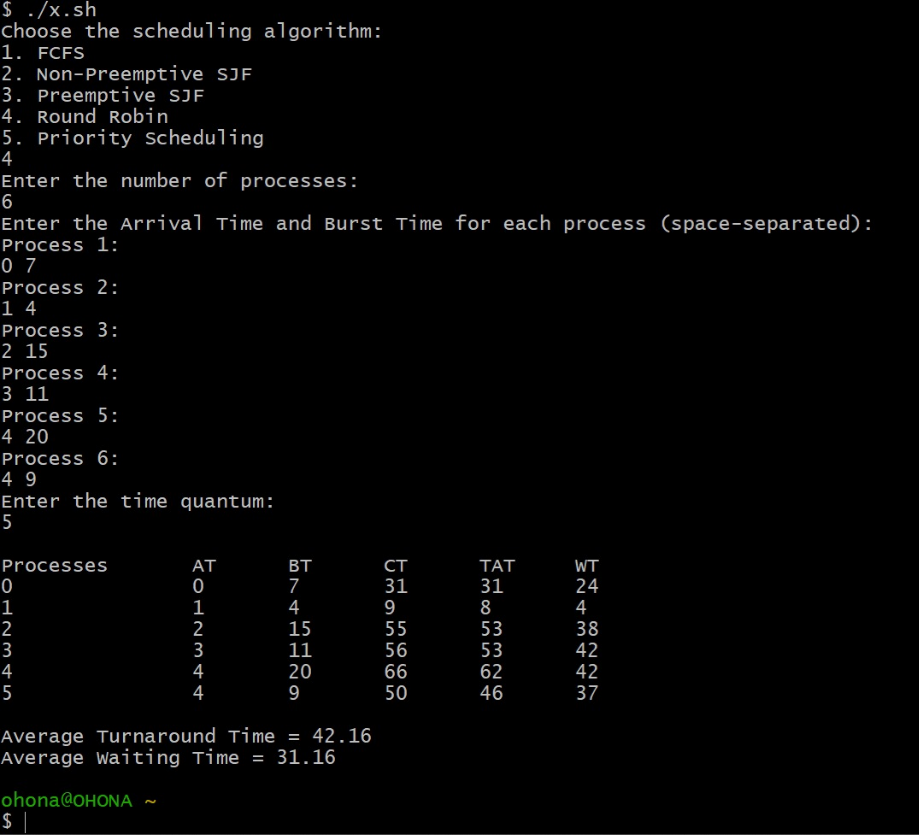
#### **5. Repository and Documentation -**

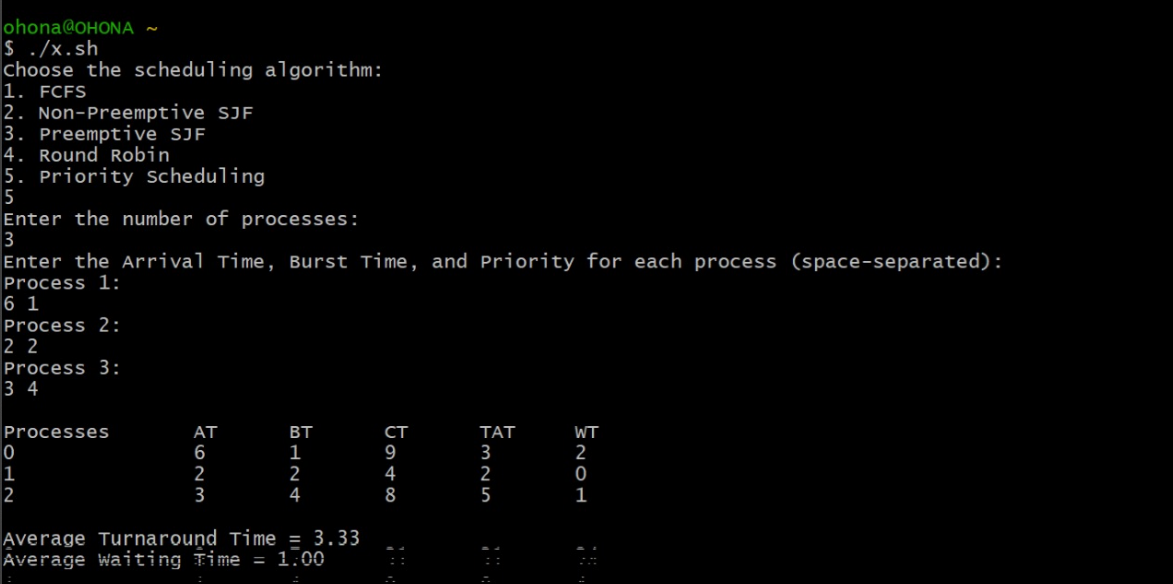
The completed project, along with detailed documentation and examples, is hosted on GitHub for accessibility and collaboration. Users can find the full project at:

[**GitHub Link**](https://github.com/Ahana-tabassum/OS_Project) **- “**https://github.com/Ahana-tabassum/OS\_Project**”**

By following this methodology, we ensured that the simulator is robust, accurate, and user-friendly, fulfilling its purpose as an educational tool.

**Output :   
  
**

****

****

**Limitations :**

While developing the Process Scheduler Simulator, we faced a few limitations due to using Cygwin to simulate Linux commands on Windows:

1. **Compatibility Issues:**  
    Some Linux commands and utilities were not fully supported in Cygwin, requiring workarounds and affecting certain parts of the project.
2. **Installation Challenges:**  
    Setting up Cygwin was complex, with issues in selecting the right packages and ensuring compatibility, which delayed the project’s start.
3. **Performance Limitations:**  
    Cygwin's performance was slower than a native Linux system, which impacted the efficiency of script execution, especially for more complex simulations.
4. **Error Handling Difficulties:**  
    Debugging errors was harder in Cygwin compared to a native Linux environment, particularly when dealing with system calls and I/O handling.

Despite these challenges, the project was completed successfully, although it highlighted the need for better cross-platform compatibility and optimization.

**Future Work :**

While we have successfully implemented the core functionality, there are several areas where the project could be expanded or improved in the future:

1. **Graphical User Interface (GUI):**  
   Although the current version is command-line based, a GUI could be developed to enhance user experience, making it more visually intuitive and interactive.
2. **Real-Time Process Simulation:**  
   In the future, it would be useful to simulate processes in real-time, showing the execution flow step-by-step and visualizing the Gantt charts for each scheduling algorithm.
3. **Additional Scheduling Algorithms:**Although we focused on the basic algorithms, adding more advanced algorithms like Multilevel Queue Scheduling or Multilevel Feedback Queue could make the simulator more comprehensive.
4. **Error Handling Improvements:**  
   We attempted to incorporate advanced error handling, such as validating process burst times, handling negative or zero values, and ensuring input consistency, but we were unable to fully implement this feature due to time constraints.
5. **Performance Metrics:**  
   Integrating performance metrics such as turnaround time, waiting time, and throughput would provide users with more insights into the effectiveness of each scheduling method.

Despite the challenges and incomplete features, the project lays a strong foundation for further exploration and development in process scheduling simulation. The future improvements mentioned would enhance its functionality, making it a more powerful tool for both educational purposes and practical understanding of operating system concepts.

**Conclusion :**

In this project, we successfully developed a Process Scheduler Simulator that integrates five key CPU scheduling algorithms: FCFS, Non-Preemptive SJF, Preemptive SJF, Round Robin, and Priority Scheduling. The project was implemented using Cygwin, allowing us to simulate Linux-based scheduling techniques in a Windows environment. The simulator provides an interactive menu, enabling users to easily select an algorithm, input process details, and observe how each scheduling method works in practice.

Through this project, we gained valuable experience in shell scripting, algorithm implementation, and problem-solving. Additionally, the tool serves as an educational resource that simplifies the understanding of complex scheduling algorithms by presenting them in a clear, accessible manner.

**References :  
  
1.** A. Silberschatz, P. B. Galvin, and G. Gagne, Operating System Concepts, 9th ed. Wiley, 2018.

**2.** A. S. Tanenbaum and H. Bos, Modern Operating Systems, 4th ed. Pearson Education, 2014.

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

**4.** B. Jacob and P. Johnson, "CPU Scheduling Algorithms: A Survey of Existing Techniques and New Proposals," Int. J. Comput. Appl., vol. 167, no. 4, pp. 21-27, 2017.