CPU Scheduling Algorithms – Operating System Project Report

Name: Jesmin Anurag

December 2025

EXECUTIVE SUMMARY

This project implements a *CPU Scheduling Simulator* demonstrating fundamental Operating System concepts such as process scheduling, CPU time allocation, context switching, and performance measurement. The system allows users to simulate multiple CPU scheduling algorithms, including *First-Come, First-Served (FCFS), Shortest Job Next (SJN), Priority Scheduling, and Round Robin (RR).* Through the project, the impact of scheduling strategies on **waiting time, turnaround time, and CPU utilization** is explored, bridging theoretical OS concepts with practical implementation.

Project URL: https://github.com/yousefkotp/CPU-Scheduling-Algorithms

PROJECT OVERVIEW

Objective

To develop a CPU scheduling simulator that demonstrates core Operating System principles by modeling *process execution, waiting times, turnaround times, and Gantt chart visualization* through multiple scheduling algorithms.

OS Concepts Demonstrated

- *Process Scheduling:* Simulating CPU allocation to multiple processes based on different algorithms
- *Context Switching:* Tracking CPU execution across processes
- *Performance Metrics:* Calculating waiting time, turnaround time, and CPU utilization
- *Data Structures:* Using queues, arrays, and priority management for scheduling
- *Algorithm Implementation:* Handling preemptive and non-preemptive scheduling

Technology Stack

- *Language:* C++
- *Algorithms:* FCFS, SJN, Priority Scheduling, Round Robin
- *Data Structures:* Arrays, Queues, Linked Lists
- *I/O Interface:* Console-based input/output

TECHNICAL IMPLEMENTATION OS PERSPECTIVE

Process Scheduling Architecture

1. *Input Operations*
    - User inputs the number of processes and attributes: *arrival time, burst time, and priority*
    - Data stored in structured arrays or objects
    - Validation ensures correct values for scheduling
2. *CPU Scheduling Logic*
    - Each scheduling algorithm implemented as a *modular function*
    - *FCFS:* Non-preemptive, executed in order of arrival
    - *SJN:* Non-preemptive, selects the shortest burst time next
    - *Priority:* Can be preemptive or non-preemptive, selects process with highest priority
    - *Round Robin:* Preemptive, uses a fixed time quantum with circular queue logic
3. *Output Operations*
    - Computes *waiting time, turnaround time, and completion time* for all processes
    - Generates a *Gantt chart* to visualize CPU execution order
    - Displays average metrics for analysis

Data Flow Architecture

User Input → Process List → Scheduling Algorithm → Context Management → Waiting/Turnaround Calculation → Gantt Chart Visualization → Output Metrics

CHALLENGES FACED OS PERSPECTIVE (STAR FORMAT)

Challenge 1: Handling Multiple Scheduling Algorithms

Situation: Implementing multiple algorithms with *different execution rules* in one program caused complexity in code structure and data management.
Task: Ensure each algorithm works independently and accurately computes metrics while sharing a common input structure.
Action: Designed *modular functions* for each algorithm, reused common functions for input, calculation, and output. Implemented *clear data structures* for processes to maintain consistency across algorithms.
Result: All four algorithms were correctly simulated. Metrics are accurate, and users can switch algorithms seamlessly, demonstrating flexibility in CPU scheduling.

Challenge 2: Implementing Round Robin with Time Quantum

Situation: Round Robin requires *preemptive CPU allocation* with context switching at fixed intervals. Mismanagement led to incorrect waiting times and Gantt charts.
Task: Implement preemptive scheduling using a *circular queue*, ensuring correct CPU execution order and metric computation.
Action: Created a *ready queue* to manage processes. Implemented *time quantum logic* to decrement remaining burst time and requeue unfinished processes. Calculated waiting and turnaround times dynamically after each quantum.
Result: Round Robin scheduling became accurate and efficient, producing correct metrics and Gantt charts for all test cases. Learned practical preemptive scheduling management.

Challenge 3: Managing Priority Scheduling with Tie-Breakers

Situation: Multiple processes with the same priority or arrival time caused ambiguity in scheduling order.
Task: Develop a system to *resolve ties* consistently and correctly.
Action: Implemented *secondary criteria* such as arrival time or process ID to break ties deterministically. Tested multiple scenarios to verify correctness.
Result: Tie-breaking logic ensured predictable CPU scheduling, improving reliability and user confidence in simulation results.

Challenge 4: Gantt Chart Visualization and Metrics Accuracy

Situation: Simulating the Gantt chart correctly while maintaining accurate *waiting and turnaround times* required careful bookkeeping of CPU execution steps.
Task: Track each process's execution timeline and compute metrics dynamically during scheduling simulation.
Action: Stored process execution intervals in arrays and updated metrics after each CPU

allocation. Used iterative loops to simulate CPU cycles for preemptive scheduling.
Result: Generated accurate Gantt charts reflecting CPU execution, and metrics aligned perfectly with manual calculations. Users can visualize CPU scheduling intuitively.

OS CONCEPTS APPLIED

1. **Process Scheduling**
    - Implemented multiple algorithms to demonstrate *non-preemptive and preemptive scheduling*
    - Modeled CPU allocation and context switching
2. **Performance Metrics**
    - Calculated *waiting time, turnaround time, and completion time*
    - Demonstrated impact of different algorithms on CPU efficiency
3. **Queue Management**
    - Used arrays and circular queues for *ready queue handling*
    - Managed dynamic process addition for preemptive scheduling
4. **Simulation of OS Behavior**
    - Mimicked *CPU time allocation, process preemption, and priority handling*
    - Showed effects of scheduling decisions on overall system performance

PERFORMANCE METRICS (OS PERSPECTIVE)

Scheduling Accuracy

- All algorithms produce *correct waiting and turnaround times*
- Preemptive Round Robin correctly handles *context switches*

Efficiency

- Average waiting and turnaround times calculated accurately for multiple test cases
- Algorithms demonstrate performance differences clearly for educational purposes

Resource Usage

- CPU time tracking and arrays efficiently simulate process scheduling
- Minimal memory usage for storing process attributes and scheduling timeline

LEARNING OUTCOMES OS CONCEPTS

Core OS Concepts Mastered

1. *Process Scheduling Algorithms* – FCFS, SJN, Priority, RR
2. *Preemption & Context Switching* – Implemented dynamic CPU allocation
3. *Metric Computation* – Waiting time, turnaround time, average metrics
4. *Data Structure Management* – Arrays, queues for process handling
5. *Simulation of OS Scheduling* – Practical demonstration of theoretical concepts

Practical Skills Developed

- Implementing modular scheduling functions
- Handling preemptive and non-preemptive processes
- Resolving scheduling conflicts and tie-breaking
- Visualizing CPU execution through Gantt charts
- Bridging theory and practice in OS concepts

CONNECTION TO OS THEORY

Theoretical Concepts Applied

- *FCFS:* Non-preemptive scheduling based on arrival order
- *SJN:* Shortest process next prioritization
- *Priority Scheduling:* Importance-based CPU allocation
- *Round Robin:* Time-slice-based preemptive scheduling
- *Metrics Computation:* Practical demonstration of waiting time, turnaround time, and CPU utilization calculations
- *Simulation of CPU Behavior:* Connecting high-level programming to OS scheduling principles

FUTURE ENHANCEMENTS

1. Add *preemptive SJN* scheduling
2. Include *interactive GUI visualization* of process timelines
3. Extend to *multi-core CPU simulation*
4. Include *dynamic arrival and process creation during simulation*
5. Integrate *process I/O simulation* for mixed CPU-I/O workloads

CONCLUSION

This CPU Scheduling Simulator project effectively demonstrates *core Operating System scheduling concepts* through practical implementation. By implementing multiple algorithms and handling preemption, priorities, and performance metrics, the project bridges theory and practice. Users can observe how scheduling decisions impact CPU utilization, waiting times, and turnaround times, providing a hands-on understanding of OS process management.

REFERENCES

- Silberschatz, Galvin, Gagne. *Operating System Concepts* – Chapter on Process Scheduling
- Tanenbaum, Andrew S. *Modern Operating Systems* – CPU Scheduling and Process Management
- Project Repository: https://github.com/anuragjesmin/cse323/blob/main/.gitignore

Project Status: Complete
Course Focus: Operating System – Process Scheduling
License: MIT
Last Updated: December 2025