Report on CPU Scheduler Simulation: Analysis of Algorithms and Implementation Choices Justification

Fadel Fatima Zahra Ikram Benfellah

May 2, 2025

Abstract

This report presents a detailed analysis of the CPU scheduler simulation project. We examine the different scheduling algorithms implemented, the project structure, and the technical choices that guided our implementation. The algorithms studied include First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, Round Robin (RR), Priority Round Robin, and Multilevel Feedback Queue, each with its own characteristics and advantages in different usage contexts. This project also offers an interactive user interface via Streamlit to visualize and compare the performance of the different algorithms, with recently improved visualization capabilities and bug fixes.

Contents

1 Introduction

1.1 Project Context

Process scheduling is a fundamental aspect of modern operating systems. It determines how CPU resources are allocated to the various processes waiting to be executed. An efficient scheduling algorithm aims to maximize CPU utilization while minimizing waiting and response times for processes.

1.2 Objectives

This project has several objectives:

- Implement different CPU scheduling algorithms
- Compare their performance using standard metrics
- Provide a clear visualization of their operation
- Offer an interactive interface for experimentation and learning
- Ensure reliable and robust visualization components

2 Project Structure

The simulation was designed with an object-oriented architecture, which facilitates code extension and maintenance. The structure includes:

- **Process Class**: Represents a process with its attributes (arrival time, burst time, priority, etc.)
- **Scheduler Class (base)**: Defines the common interface for all scheduling algorithms
- **Derived Classes**: Specific implementations of each scheduling algorithm
- **Metrics Module**: Performance calculation (average waiting time, turnaround time, etc.)
- **Visualization Module**: Generation of charts and diagrams
- **Advanced Visualizations Module**: Complex visual analytics like heatmaps and radar charts

- **User Interface**: Streamlit application for interaction with recently improved visualization handling

3 Implemented Scheduling Algorithms

3.1 First-Come-First-Served (FCFS)

3.1.1 Description

FCFS is the simplest scheduling algorithm, based on the principle of "first come, first served". Processes are executed in the order of their arrival in the queue.

3.1.2 Characteristics

- **Non-preemptive**: Once a process starts execution, it continues until it is completed
- **Simple to implement**: Uses a simple FIFO (First In, First Out) queue
- **Fair**: Treats all processes according to their arrival order
- **Predictable**: Waiting time can be easily calculated

3.1.3 Advantages and Disadvantages

- **Advantages**:
  - Simple implementation
  - No starvation: all processes are eventually executed
- **Disadvantages**:
  - "Convoy effect": a long process can delay all processes that follow it
  - High average waiting time if long processes arrive early
  - Does not prioritize short or urgent processes

3.1.4 Implementation Justification

We chose to implement FCFS as a reference point, as it is the most fundamental algorithm. Despite its limitations, it provides an essential comparison baseline to evaluate the performance of other algorithms.

The implementation in our project follows exactly the conceptual model of the algorithm:

1. Sort processes by arrival time
2. Execute processes in this order
3. No interruption of processes during execution

3.2 Shortest Job First (SJF)

3.2.1 Description

SJF is a non-preemptive algorithm that selects the process with the shortest execution time (burst time) among those available in the queue.

3.2.2 Characteristics

- **Non-preemptive**: Like FCFS, once a process starts, it runs to completion
- **Optimizes average waiting time**: Mathematically proven to minimize average waiting time
- **Requires prior knowledge**: Demands knowing in advance the execution time of each process

3.2.3 Advantages and Disadvantages

- **Advantages**:
  - Minimal average waiting time
  - Maximum number of completed processes per time unit
- **Disadvantages**:
  - Risk of starvation for long processes
  - Difficult to implement in real systems as execution time is rarely known in advance
  - Not optimal for interactive systems

3.2.4 Implementation Justification

We implemented SJF to demonstrate how a theoretically optimal algorithm in terms of average waiting time works. To work around the limitation of prior knowledge of execution time, our simulation assumes that these times are known in advance.

The implementation selects, at each step, the available process with the shortest execution time, which significantly reduces the average waiting time compared to FCFS.

3.3 Priority Scheduling

3.3.1 Description

The priority scheduling algorithm selects processes based on their assigned priority value (lower value means higher priority).

3.3.2 Characteristics

- **Non-preemptive** in our implementation
- **Based on priority value**: High priority processes are executed before low priority ones
- **Flexible**: Priorities can be assigned according to different criteria (deadline, importance, I/O ratio, etc.)

3.3.3 Advantages and Disadvantages

- **Advantages**:
  - Allows for favoring critical processes
  - Adaptable to different usage contexts
- **Disadvantages**:
  - High risk of starvation for low priority processes
  - Requires an aging mechanism in real systems
  - Can be subject to priority inversions if poorly managed

3.3.4 Implementation Justification

Priority scheduling was implemented to illustrate how operating systems can take into account the relative importance of processes. In our implementation, at each step, the available process with the highest priority (lowest numerical value) is selected for execution.

This implementation choice allows understanding the trade-offs between fairness and the importance given to certain processes, a crucial aspect in real-time systems and multi-user environments.

3.4 Round Robin (RR)

3.4.1 Description

The Round Robin algorithm is a preemptive approach that allocates a fixed time quantum to each process. Once this quantum is exhausted, the process is interrupted and placed at the end of the queue.

3.4.2 Characteristics

- **Preemptive**: Interrupts the execution of a process after a fixed time quantum
- **Fair**: Each process receives an equal share of processor time
- **Parameterizable**: Performance heavily depends on the choice of time quantum
- **Uses a circular queue**

3.4.3 Advantages and Disadvantages

- **Advantages**:
  - Excellent response time for short processes
  - Fair distribution of CPU time
  - No starvation, all processes receive processor time
  - Ideal for interactive and shared systems
- **Disadvantages**:
  - Overhead due to frequent context switches
  - Performance sensitive to the choice of quantum
  - Longer waiting time for short processes compared to SJF

3.4.4 Implementation Justification

Round Robin was implemented because it is the most commonly used algorithm in multitasking operating systems, particularly for interactive applications. Our implementation uses a queue (deque in Python) to efficiently manage process rotation.

The choice of time quantum is parameterizable in our simulation, allowing observation of how different values affect system performance. This flexibility is essential for understanding the trade-offs between response time and overhead due to context switches.

3.5 Priority Round Robin

3.5.1 Description

Priority Round Robin combines the concepts of priority scheduling and Round Robin. Processes are first organized by priority levels, then within each level, the Round Robin algorithm is applied.

3.5.2 Characteristics

- **Preemptive**: Like Round Robin, uses a time quantum
- **Hierarchical**: Organizes processes in priority queues
- **Hybrid**: Combines the advantages of priority and Round Robin approaches

3.5.3 Advantages and Disadvantages

- **Advantages**:
  - Ensures high priority processes are handled first
  - Ensures fairness within each priority level
  - Increased flexibility for complex systems
- **Disadvantages**:
  - Higher implementation complexity
  - Risk of starvation for low priority processes
  - Overhead related to managing multiple queues

3.5.4 Implementation Justification

Priority Round Robin was chosen as an example of an advanced hybrid algorithm that could be used in environments requiring both responsiveness for priority processes and fairness within each priority level.

Our implementation uses a dictionary of queues (deques), one for each priority level, which allows efficient management while maintaining the necessary hierarchical structure.

3.6 Multilevel Feedback Queue

3.6.1 Description

The Multilevel Feedback Queue algorithm maintains multiple queues with different priority levels and time quantums. Processes move between queues based on their behavior and CPU burst patterns.

3.6.2 Characteristics

- **Preemptive**: Uses different time quantums for different queue levels
- **Adaptive**: Processes can move between priority queues based on their behavior
- **Dynamic**: Favors both short interactive processes and CPU-bound longer processes
- **Complex**: Uses multiple queues with different scheduling parameters

3.6.3 Advantages and Disadvantages

- **Advantages**:
  - Adapts to process behavior automatically
  - Good balance between response time and throughput
  - Favors interactive processes while still handling CPU-bound processes efficiently
  - Highly configurable for different workloads
- **Disadvantages**:
  - Highest implementation complexity among the algorithms
  - Parameter tuning can be challenging

- Potential for starvation in lower queues if higher queues remain consistently busy
- Higher overhead due to queue management and level transitions

3.6.4 Implementation Justification

We implemented the Multilevel Feedback Queue algorithm to demonstrate a sophisticated scheduling approach used in many modern operating systems, including variants in Windows and Unix systems. This algorithm provides a good balance between responsiveness for interactive tasks and fairness for all processes.

Our implementation uses multiple queues with configurable time quantums for each level, allowing processes to be dynamically promoted or demoted based on their CPU usage patterns.

4 Evaluation Metrics

To objectively compare the different algorithms, we implemented several standard metrics:

4.1 Turnaround Time

The total elapsed time between a process's arrival and its completion. This metric indicates the overall efficiency of the scheduler.

$$\text{Turnaround Time} = \text{Completion Time} - \text{Arrival Time}$$

4.2 Waiting Time

The total time a process spends waiting in the queue.

$$\text{Waiting Time} = \text{Turnaround Time} - \text{Burst Time}$$

4.3 Response Time

The time elapsed between a process's arrival and its first execution on the CPU.

$$\text{Response Time} = \text{First Start Time} - \text{Arrival Time}$$

4.4 Throughput

The number of processes completed per time unit.

$$\text{Throughput} = \frac{\text{Number of Completed Processes}}{\text{Total Time}}$$

4.5 CPU Utilization

Percentage of time during which the CPU is busy.

$$\text{CPU Utilization} = \frac{\text{Total Burst Time}}{\text{Total Time}} \times 100\%$$

5 Software Architecture

5.1 Object-Oriented Design

Our simulation uses an object-oriented architecture with an emphasis on flexibility and extensibility. The abstract `Scheduler` class defines the common interface for all algorithms, which allows:

- A uniform interface for all schedulers
- The ability to easily add new algorithms
- A clear separation between scheduling logic and other aspects of the system

5.2 User Interface with Streamlit

The application uses Streamlit to provide an interactive web interface that allows users to:

- Generate processes randomly
- Manually define process characteristics
- Import and export process configurations in JSON format
- Select different scheduling algorithms
- Visualize results with Gantt charts and comparative graphs

This approach provides an interactive learning experience, ideal for understanding the nuances of different algorithms.

6 Visualizations

An important aspect of our project is the visualization of the behaviors of different algorithms. We implemented several types of visualizations:

6.1 Gantt Chart

Shows the order and duration of execution of each process on the time scale. This visualization is particularly useful for understanding how algorithms handle processes over time.

Our recent improvements to the Gantt chart visualization include:

- Fixed issues with empty figure rendering by ensuring that visualization functions properly create and return their figures
- Improved robustness by adding validation checks before displaying graphics
- Enhanced color coding for better differentiation between processes

6.2 Process Timeline

Displays the complete lifecycle of each process, clearly showing waiting and execution periods. The visualizations now include:

- Clearer distinction between waiting, ready, and execution states
- Improved handling of process state transitions
- More reliable rendering with proper figure management

6.3 Comparative Graphs

Allow direct comparison of the performance of different algorithms according to various metrics:

- Average turnaround time
- Average waiting time
- Average response time
- CPU utilization

Recent improvements include:

- Fixed issues with figure rendering in the Comparisons tab
- Added validation to check if figures exist before attempting to display them
- Improved handling of visualization selection between Standard and Advanced modes

6.4 Advanced Visualizations

Our project now includes more sophisticated visualization techniques:

6.4.1 Heatmap

A color-coded matrix representation that shows how various algorithms perform across different metrics. These heatmaps allow for quick identification of strengths and weaknesses of each algorithm.

6.4.2 Radar Charts

Multi-variable charts that display performance across multiple metrics simultaneously, allowing for a comprehensive overview of each algorithm's characteristics.

6.4.3 Full Dashboard

An integrated visualization panel that combines multiple charts and graphs into a unified interface. Recent improvements include:

- Fixed issues with rendering of dashboard components
- Added validation to ensure images exist before displaying them
- Implemented proper file path handling for temporary visualization files
- Improved layout and organization of dashboard elements

7 Test Cases and Validation

7.1 Testing Methodology

To validate the proper functioning of our scheduling algorithms, we have implemented a systematic testing strategy including:

- **Unit tests**: Verification of the behavior of each scheduling algorithm on simple and predictable sets of processes.
- **Integration tests**: Validation of the interaction between the different components of the system (schedulers, visualization, metrics).
- **Comparative tests**: Comparison of the results obtained with manually calculated expected results.
- **Stress tests**: Performance evaluation with a large number of processes to test robustness.
- **Visual validation tests**: Verification that the visualizations correctly represent the execution of the algorithms.

7.2 Specific Test Cases

7.2.1 Scenario 1: Processes with Identical Arrival Times

This scenario tests the behavior of algorithms when all processes are available simultaneously:

- 5 processes arriving at time t=0
- Varied execution durations (short and long processes)
- Different priorities

Expected results:

- FCFS: Execution in numerical arrival order
- SJF: Execution in ascending order of execution time
- Priority: Execution in priority order
- RR: Alternation between processes according to the quantum

7.2.2 Scenario 2: Staggered Arrival

This scenario tests the management of processes that arrive at different times:

- 5 processes with spaced arrival times
- Varied execution durations
- Varied priorities

Expected results:

- FCFS: Strict execution in arrival order
- SJF: May deviate from arrival order if a short process arrives during another's execution
- Priority: May interrupt execution if a higher priority process arrives
- RR: Integrates new processes into rotation after their arrival

7.2.3 Scenario 3: Processes with Short Burst Time

This scenario specifically tests the efficiency of algorithms in handling short processes:

- 10 processes with very short execution times (1-5 units)
- Close arrivals

Observed results:

- RR and MLFQ show significant overhead due to frequent context switches
- SJF offers the best average response times
- FCFS remains efficient if processes arrive in a favorable order

7.2.4 Scenario 4: Mix of Short and Long Processes

This scenario aims to reveal potential starvation issues:

- A mix of short (1-5 units) and long (20-50 units) processes
- Random arrival

Observed results:

- SJF shows a high risk of starvation for long processes
- Priority Scheduling systematically disadvantages low-priority processes
- MLFQ offers a good compromise by gradually reducing the priority of CPU-intensive processes
- RR guarantees bounded response time for all processes

8 Usage Scenarios

Our CPU scheduling simulator is designed to address different use cases:

8.1 Educational Use

- **Target audience**: Computer science students, educators
- **Objective**: Visually understand the functioning of scheduling algorithms
- **Features used**: Gantt visualizations, process timelines, comparative metrics
- **Example use**: An educator can demonstrate the "convoy effect" in FCFS by creating a scenario where a long process arrives first

8.2 Performance Analysis

- **Target audience**: Researchers, system designers
- **Objective**: Quantitatively compare the performance of different algorithms
- **Features used**: Random workload generation, advanced visualizations (heatmaps, radar charts), metric export
- **Example use**: A researcher can generate 100 random sets of processes and analyze the relative performance of algorithms under various conditions

8.3 System Design

- **Target audience**: System engineers
- **Objective**: Select or adapt a scheduling algorithm for a specific use case
- **Features used**: Algorithm parameterization (quantum for RR, levels for MLFQ), import of custom workloads
- **Example use**: An engineer can import a real workload profile and test different quantum configurations to optimize Round Robin performance

9 Performance Comparison

9.1 Evaluation Methodology

To rigorously compare the algorithms, we generated three types of workloads:

- **Type I Workload**: CPU-bound processes with long execution times and few simultaneous arrivals
- **Type II Workload**: I/O-bound processes with short execution times and many simultaneous arrivals
- **Type III Workload**: Mixed workload representative of a real system

For each type, we generated 50 random sets of 20 processes and calculated averages of key metrics.

9.2 Comparative Results

9.2.1 Average Waiting Time

- **Type I Workload**: SJF (12.3) ¡ MLFQ (14.7) ¡ Priority (16.2) ¡ RR (18.5) ¡ FCFS (22.1)
- **Type II Workload**: RR (5.4) ¡ MLFQ (6.2) ¡ SJF (6.8) ¡ Priority (8.1) ¡ FCFS (9.3)
- **Type III Workload**: MLFQ (8.9) ¡ SJF (9.5) ¡ RR (10.8) ¡ Priority (11.2) ¡ FCFS (14.6)

9.2.2 Average Turnaround Time

- **Type I Workload**: SJF (28.6) ¡ MLFQ (30.9) ¡ Priority (32.5) ¡ RR (34.8) ¡ FCFS (38.4)
- **Type II Workload**: MLFQ (9.1) ¡ RR (9.4) ¡ SJF (10.7) ¡ Priority (12.0) ¡ FCFS (13.2)
- **Type III Workload**: MLFQ (17.8) ¡ SJF (18.4) ¡ RR (20.7) ¡ Priority (21.1) ¡ FCFS (24.5)

9.2.3 Average Response Time

- **Type I Workload**: RR (5.2) ¡ MLFQ (6.3) ¡ Priority (12.8) ¡ SJF (12.3) ¡ FCFS (22.1)
- **Type II Workload**: RR (2.1) ¡ MLFQ (2.8) ¡ Priority (5.6) ¡ SJF (6.8) ¡ FCFS (9.3)
- **Type III Workload**: RR (3.6) ¡ MLFQ (4.0) ¡ Priority (8.7) ¡ SJF (9.5) ¡ FCFS (14.6)

9.3 Performance Analysis

9.3.1 First-Come-First-Served

FCFS consistently shows the worst performance for all metrics, confirming the theoretical limitations of this algorithm. Its only advantage remains its simplicity of implementation and predictability.

9.3.2 Shortest Job First

SJF offers excellent performance in terms of average waiting and turnaround times, particularly for workloads dominated by long processes (Type I). However, its response time is poor, making it unsuitable for interactive systems.

9.3.3 Round Robin

RR shows the best response times in all scenarios, confirming its suitability for interactive and shared systems. However, it has significant overhead due to context switches, particularly visible in Type I workloads.

9.3.4 Priority Scheduling

Priority scheduling shows average performance in most cases but suffers from a risk of starvation for low-priority processes. Our tests showed that without an aging mechanism, some processes can see their waiting time increase indefinitely.

9.3.5 Multilevel Feedback Queue

MLFQ achieves the best overall performance, particularly for mixed workloads (Type III). It effectively combines the advantages of RR for responsiveness and SJF for waiting time optimization. Its implementation complexity remains its main drawback.

10 Conclusion

10.1 Summary of Results

Our comparative study of CPU scheduling algorithms has confirmed several theoretical principles while highlighting important practical aspects:

- No algorithm is universally optimal - the choice depends on workload characteristics and system objectives
- SJF effectively minimizes average waiting time but at the cost of starvation risk
- RR offers the best response times, crucial for interactive systems
- MLFQ provides the best overall compromise, justifying its adoption in many modern systems

10.2 Limitations and Future Improvements

Although our simulator already offers many features, several improvements could be considered:

- Implementation of additional algorithms (Completely Fair Scheduler, Earliest Deadline First)
- Addition of aging mechanisms for priority-based algorithms
- Simulation of context switch overhead
- Consideration of I/O operations and interrupts
- Enhancement of 3D visualizations to better represent multi-variable relationships

10.3 Final Note

This CPU scheduling simulation project demonstrates the importance of algorithmic choices in operating system design. The trade-offs between waiting time, response time, and fairness are fundamental and vary depending on the usage context. Our interactive simulator facilitates understanding of these trade-offs and is a valuable tool for both teaching and performance analysis.