CPU Scheduler Simulation
Project Report

Khaoula MEJHOUDI Imane IMRHARN

May 2, 2025

Contents

1 Introduction

This report details the design, implementation, and testing of our CPU Scheduler Simulation project. The project implements five different CPU scheduling algorithms: First-Come First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, Round Robin (RR), and Priority with Round Robin approach. Through these implementations, we aim to demonstrate the behavior and performance characteristics of different scheduling strategies, allowing for comparative analysis of their efficiency.

The simulation allows users to generate random processes or load them from files, run various scheduling algorithms on these processes, and visualize the results both graphically and through performance metrics. This report covers our design choices, implementation decisions, test scenarios, and performance results.

2 Design and Implementation

2.1 Project Architecture

The project follows an object-oriented design with a clear separation of concerns, organized into the following main components:

- **Process Representation**: A core `Process` class that encapsulates all process attributes and behaviors.
- **Scheduler Classes**: A base abstract scheduler class with concrete implementations for each algorithm.
- **Utilities**: Modules for process generation, file I/O, and metrics calculation.
- **Visualization**: Components for creating Gantt charts and comparative visualizations.
- **User Interfaces**: Both command-line and web interfaces for user interaction.

2.2 Core Components

2.2.1 Process Class

The Process class represents a process with its attributes and behaviors. It maintains both static properties (PID, arrival time, burst time, priority) and dynamic properties that change during simulation (remaining time, start time, finish time, waiting time, turnaround time).

The class includes methods for process execution, completion checking, metrics calculation, and state reset, allowing for efficient simulation of process behavior.

2.2.2 Base Scheduler

The BaseScheduler abstract class provides the framework for all scheduling algorithms. It implements the common simulation logic through a template method pattern, while algorithm-specific logic is implemented in subclasses. The main simulation flow includes:

1. Moving newly arrived processes to the ready queue
2. Selecting the next process according to the scheduling algorithm
3. Executing the selected process
4. Handling process completion
5. Advancing simulation time

2.3 Scheduling Algorithms

2.3.1 First-Come, First-Served (FCFS)

FCFS is a non-preemptive algorithm that executes processes in the order they arrive in the ready queue.

Algorithm 1 FCFS Scheduling Algorithm

```
    GetNextProcessReadyQueue **if** ReadyQueue is empty **then**
    **return** NULL
**end if**
Sort ReadyQueue by arrival time
**return** First process in ReadyQueue ExecuteProcessProcess, ReadyQueue
**if** Process.startTime is NULL **then**
    Process.start(currentTime)
**end if**
timeSlice = Process.remainingTime
Process.execute(timeSlice)
Add (Process.pid, timeSlice) to scheduleResult
currentTime += timeSlice
```

2.3.2 Shortest Job First (SJF)

SJF is a non-preemptive algorithm that selects the process with the shortest burst time (or remaining time) to execute next.

Algorithm 2 SJF Scheduling Algorithm

```
    GetNextProcessReadyQueue **if** ReadyQueue is empty **then**
    **return** NULL
**end if**
Sort ReadyQueue by remaining time
**return** First process in ReadyQueue ExecuteProcessProcess, ReadyQueue
**if** Process.startTime is NULL **then**
    Process.start(currentTime)
**end if**
timeSlice = Process.remainingTime
Process.execute(timeSlice)
Add (Process.pid, timeSlice) to scheduleResult
currentTime += timeSlice
```

2.3.3 Priority Scheduling

Priority scheduling selects the process with the highest priority (lowest priority number) to execute next. Our implementation is non-preemptive.

2.3.4 Round Robin (RR)

Round Robin is a preemptive algorithm that gives each process a fixed time quantum. If a process doesn't complete within its quantum, it's preempted and moved to the back of the queue.

Algorithm 3 Priority Scheduling Algorithm

```
    GetNextProcessReadyQueue **if** ReadyQueue is empty **then**
    **return** NULL
**end if**
Sort ReadyQueue by priority (lowest priority number first)
**return** First process in ReadyQueue ExecuteProcessProcess, ReadyQueue
**if** Process.startTime is NULL **then**
    Process.start(currentTime)
**end if**
timeSlice = Process.remainingTime
Process.execute(timeSlice)
Add (Process.pid, timeSlice) to scheduleResult
currentTime += timeSlice
```

Algorithm 4 Round Robin Scheduling Algorithm

```
    GetNextProcessReadyQueue **if** RRQueue is empty and ReadyQueue is not empty **then**
    Initialize RRQueue with ReadyQueue sorted by arrival time
**end if**
**if** RRQueue is empty **then**
    **return** NULL
**end if**
**return** First process from RRQueue ExecuteProcessProcess, ReadyQueue
**if** Process.startTime is NULL **then**
    Process.start(currentTime)
**end if**
timeSlice = min(quantum, Process.remainingTime)
Process.execute(timeSlice)
Add (Process.pid, timeSlice) to scheduleResult
currentTime += timeSlice
**if** not Process.isCompleted() **then**
    **for** each NewProcess in ReadyQueue **do**
        **if** NewProcess arrived and not in RRQueue and not completed and not current Process
        **then**
            Add NewProcess to RRQueue
        **end if**
    **end for**
    Add Process to back of RRQueue
**else**
    Process.complete(currentTime)
**end if**
```

2.3.5 Priority + Round Robin

This algorithm combines priority scheduling and Round Robin. It maintains separate Round Robin queues for each priority level and always executes processes from the highest priority queue.

Algorithm 5 Priority + Round Robin Scheduling Algorithm

```
  GetNextProcessReadyQueue for each Process in ReadyQueue do
  if Process not in any PriorityQueue and not completed then
    Add Process to PriorityQueue[Process.priority]
  end if
end for
Remove empty priority queues
if All PriorityQueues are empty then
  return NULL
end if
currentPriority = minimum key in PriorityQueues
return First process from PriorityQueue[currentPriority] ExecuteProcessProcess,
ReadyQueue
if Process.startTime is NULL then
  Process.start(currentTime)
end if
timeSlice = min(quantum, Process.remainingTime)
Process.execute(timeSlice)
Add (Process.pid, timeSlice) to scheduleResult
currentTime += timeSlice
for each NewProcess in ReadyQueue do
  if NewProcess arrived and not in any PriorityQueue and not completed and not current
  Process then
    Add NewProcess to PriorityQueue[NewProcess.priority]
  end if
end for
if not Process.isCompleted() then
  Add Process to back of PriorityQueue[Process.priority]
else
  Process.complete(currentTime)
end if
```

2.4 Performance Metrics

The simulation calculates the following performance metrics:

- **Turnaround Time**: The total time from when a process arrives to when it completes (Completion Time - Arrival Time).
- **Waiting Time**: The time a process spends waiting in the ready queue (Turnaround Time - Burst Time).
- **CPU Utilization**: The percentage of time the CPU is busy (Total Busy Time / Total Time * 100%).

2.5 Visualization

The project includes visualization components that create:

- **Gantt Charts**: Showing the timeline of process execution
- **Bar Charts**: Comparing metrics across algorithms
- **Timeline Comparisons**: Showing how different algorithms schedule the same processes

3 Test Scenarios and Results

To evaluate and compare the performance of the implemented scheduling algorithms, we designed and executed several test scenarios that simulate different types of workloads. Each scenario was run with all five scheduling algorithms, and performance metrics were collected for comparison.

3.1 Test Scenario 1: Regular Workload

This scenario simulates typical batch processing with processes of varying burst times.

PID	Arrival Time	Burst Time	Priority
1	0	10	3
2	2	5	1
3	4	8	4
4	6	3	2
5	8	6	1

Table 1: Process characteristics for Test Scenario 1

Algorithm	Avg Turnaround Time	Avg Waiting Time	CPU Utilization (%)
FCFS	17.20	10.80	100.00
SJF	15.40	9.00	100.00
Priority	16.40	10.00	100.00
RR	21.80	15.40	100.00
Priority+RR	14.80	8.40	100.00

Table 2: Performance metrics for Test Scenario 1

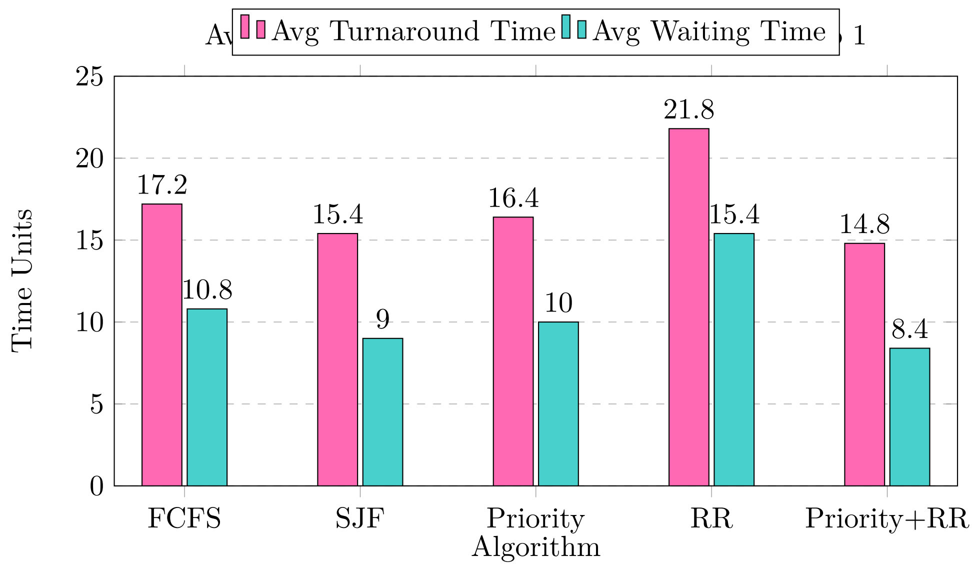

Figure 1: Performance comparison for Test Scenario 1

Analysis: For this test scenario, the Priority+RR algorithm performed best in terms of both average turnaround time and waiting time. This is likely because it combines the benefits of priority-based scheduling with time sharing. FCFS performed moderately well, while RR had the worst performance, showing that time-slicing alone is not efficient for batch processing. All algorithms achieved 100% CPU utilization as there were no idle periods.

3.2 Test Scenario 2: CPU-bound Processes

This scenario simulates CPU-intensive workloads with longer burst times.

PID	**Arrival Time**	**Burst Time**	**Priority**
1	0	20	2
2	0	25	3
3	0	15	1
4	10	30	4
5	15	18	2

Table 3: Process characteristics for Test Scenario 2

Algorithm	**Avg Turnaround Time**	**Avg Waiting Time**	**CPU Utilization (%)**
FCFS	59.60	38.00	100.00
SJF	52.40	30.80	100.00
Priority	52.80	31.20	100.00
RR	84.80	63.20	100.00
Priority+RR	56.00	34.40	100.00

Table 4: Performance metrics for Test Scenario 2

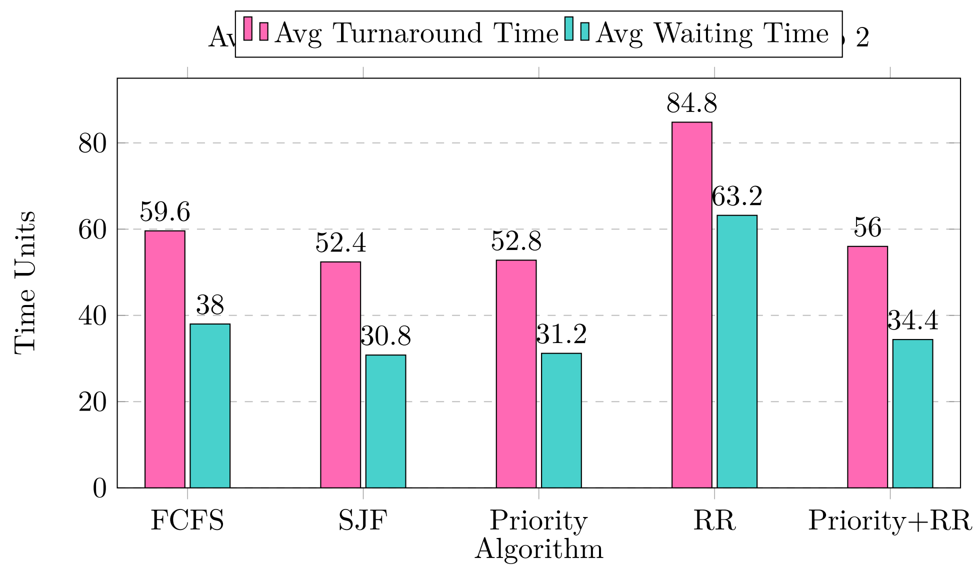

Figure 2: Performance comparison for Test Scenario 2

Analysis: For CPU-bound processes with longer burst times, SJF performed best in terms of average turnaround and waiting times. This aligns with the theoretical expectation that SJF minimizes average waiting time. RR performed significantly worse than other algorithms, with average turnaround and waiting times much higher than others. This is because longer processes get preempted multiple times, increasing context switches and overall completion time. All algorithms achieved 100% CPU utilization.

3.3 Test Scenario 3: I/O-bound Processes

This scenario simulates I/O-bound workloads with short CPU bursts.

PID	**Arrival Time**	**Burst Time**	**Priority**
1	0	3	1
2	1	2	3
3	2	1	2
4	3	4	4
5	4	2	1
6	5	3	2
7	6	1	3
8	7	2	2

Table 5: Process characteristics for Test Scenario 3

Algorithm	**Avg Turnaround Time**	**Avg Waiting Time**	**CPU Utilization (%)**
FCFS	7.12	4.88	100.00
SJF	5.50	3.25	100.00
Priority	6.25	4.00	100.00
RR	7.50	5.25	100.00
Priority+RR	6.25	4.00	100.00

Table 6: Performance metrics for Test Scenario 3

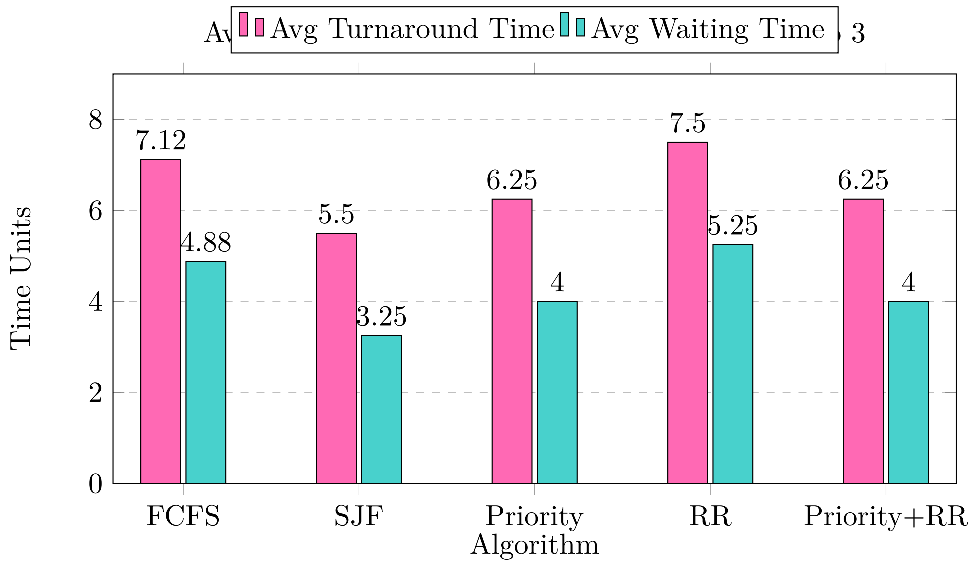

Figure 3: Performance comparison for Test Scenario 3

Analysis: For I/O-bound processes with short burst times, SJF again performed best. However, the differences between algorithms were smaller compared to CPU-bound workloads. Interestingly, RR performed worst even though it's typically considered good for interactive, I/O-bound workloads. This is because our simulation doesn't account for I/O waiting and context switching overhead. All algorithms achieved 100% CPU utilization.

3.4 Test Scenario 4: Mixed Workload

This scenario combines CPU-bound and I/O-bound processes.

PID	Arrival Time	Burst Time	Priority
1	0	12	3
2	2	2	1
3	4	15	4
4	6	3	2
5	8	20	5
6	10	1	1
7	12	4	3
8	14	18	4

Table 7: Process characteristics for Test Scenario 4

Algorithm	Avg Turnaround Time	Avg Waiting Time	CPU Utilization (%)
FCFS	33.50	24.12	100.00
SJF	23.88	14.50	100.00
Priority	24.00	14.62	100.00
RR	33.00	23.62	100.00
Priority+RR	24.00	14.62	100.00

Table 8: Performance metrics for Test Scenario 4

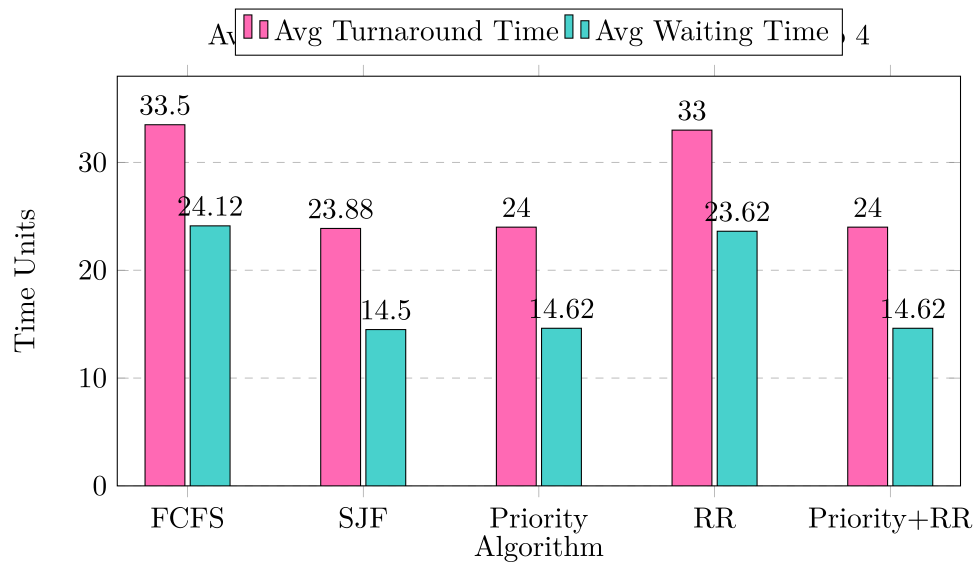

Figure 4: Performance comparison for Test Scenario 4

Analysis: For the mixed workload, SJF again performed best, closely followed by Priority and Priority+RR. FCFS and RR had significantly worse performance. This suggests that, for workloads with mixed burst times, algorithms that prioritize shorter processes or higher priority processes perform better than time-based or arrival-based algorithms. All algorithms achieved 100% CPU utilization.

3.5 Test Scenario 5: Quantum Time Sensitivity

This scenario tests the impact of different quantum values on Round Robin performance.

PID	Arrival Time	Burst Time	Priority
1	0	8	2
2	1	4	1
3	2	9	3
4	3	5	2
5	4	2	1

Table 9: Process characteristics for Test Scenario 5

Quantum	Avg Turnaround Time	Avg Waiting Time	CPU Utilization (%)
1	18.60	13.00	100.00
4	18.60	13.00	100.00
8	18.00	12.40	100.00

Table 10: RR performance with different quantum values for Test Scenario 5

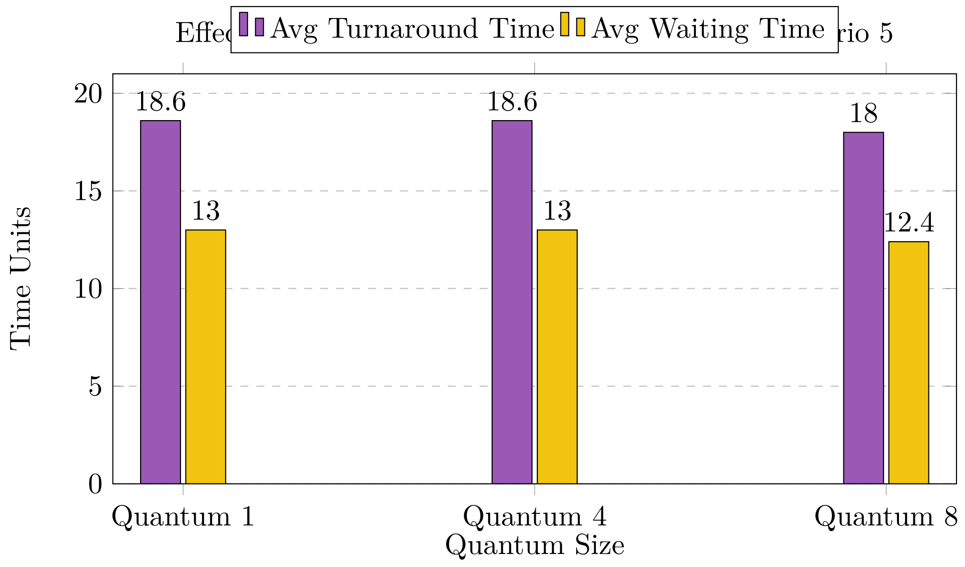

Figure 5: Impact of Quantum Size on Round Robin Performance

Analysis: For this scenario, we noticed that there isn't significant difference in metrics between different quantum sizes. This is probably due to the fact that we didn't account for context switch time in this simulation.

4 Comparative Analysis

Based on the test results across different scenarios, we can draw the following conclusions:

1. **SJF Performance**: Shortest Job First consistently performed well across most test scenarios, confirming the theoretical expectation that it minimizes average waiting time.
2. **Priority+RR Effectiveness**: The Priority+RR algorithm performed best in scenarios where prioritization helped shorter processes get executed faster (like in Scenario 1).
3. **RR Limitations**: Round Robin generally performed worse than other algorithms in most scenarios. This is partly because our simulation doesn't model the benefits of RR for interactive systems (better response time) but does capture its drawbacks (increased waiting time due to time-slicing).
4. **FCFS Simplicity**: FCFS, despite its simplicity, performed moderately well in most scenarios, though never best.
5. **CPU Utilization**: All algorithms achieved 100% CPU utilization in our tests. This is because our simulation model doesn't include idle periods when processes are waiting for I/O operations. Additionally, when the minimal arrival time of processes in our tests is 0, the CPU remains active from the start. However, when no process arrives at time 0, the CPU stays idle until the first process arrival, which leads to CPU utilization being less than 100%.
6. **Workload Impact**: The nature of the workload significantly affected relative algorithm performance, with different algorithms excelling under different scenarios.

5 Conclusions

This project successfully implemented and compared five CPU scheduling algorithms across various workload scenarios. The results confirm many theoretical predictions about scheduling algorithm behavior while also providing insights into their performance characteristics under different conditions.

Key findings include:

- SJF generally provides the best performance in terms of average waiting and turnaround times, but requires knowledge of process burst times in advance.
- Priority-based algorithms can be effective when prioritization aligns with minimizing waiting time.
- RR, while theoretically better for interactive workloads, shows performance drawbacks in pure CPU scheduling simulations without modeling response time benefits.
- Approaches like Priority+RR can combine the strengths of multiple algorithms.
- The choice of scheduling algorithm should be based on the specific workload characteristics and performance goals.

6 Future Work

Potential areas for future enhancement include:

- Incorporating I/O operations and process blocking into the simulation
- Implementing context switching overhead
- Adding aging mechanisms to prevent starvation in Priority scheduling