

**“Quad-Sched: A Short Term Schedular”**

**by**

**Aleeza Asghar**

Table of Contents

1. **Introduction3**
2. **Key Aspects of the projects3**
3. **Methodology4**

First Approach……………………………………………………………………………………………………………………………………4

Second Approach……………………………………………………………………………………………………………………………….6

Third Approach………………………………………………………………………………………………………………………………….7

1. **Challenges9**
2. **Optimization9**
3. **Results9**
4. **Conclusion11**

1. INTRODUCTION:  
In operating systems, the short-term scheduler is responsible for selecting a ready process to run on the CPU. A commonly used approach for short-term scheduling is the **Multi-Level Feedback Queue (MLFQ).** MLFQ uses multiple queues with different scheduling algorithms to handle various types of processes. Processes are promoted or demoted between queues based on their behavior favoring short or interactive tasks while preventing starvation of longer ones.

This report analyzes three different implementations of an MLFQ scheduler, each using a unique combination of algorithms such as **Round Robin, SJF, Priority, and SRTF**. The goal is to compare their effectiveness in managing CPU time, minimizing waiting, and improving overall system efficiency.

**2. Key Aspects of the Project**

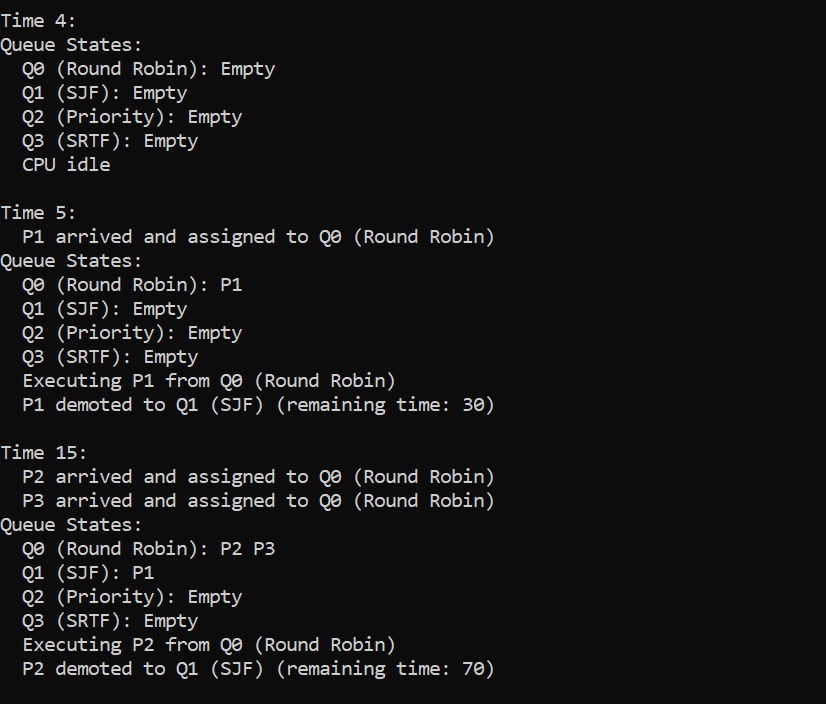
This project focuses on designing and implementing a short-term scheduler using a Multi-Level Feedback Queue (MLFQ) to manage process execution efficiently. Four scheduling algorithms: Round Robin, Shortest Job First (SJF), Priority Scheduling, and Shortest Remaining Time First (SRTF) were selected and implemented in a modular C++ program. Data structures were created to represent processes with attributes like arrival time, burst time, and priority. The scheduler supports context switching and tracks performance metrics such as average waiting time, turnaround time, and CPU utilization. Three distinct MLFQ configurations were tested to compare their effectiveness, with optimizations applied to enhance system performance.

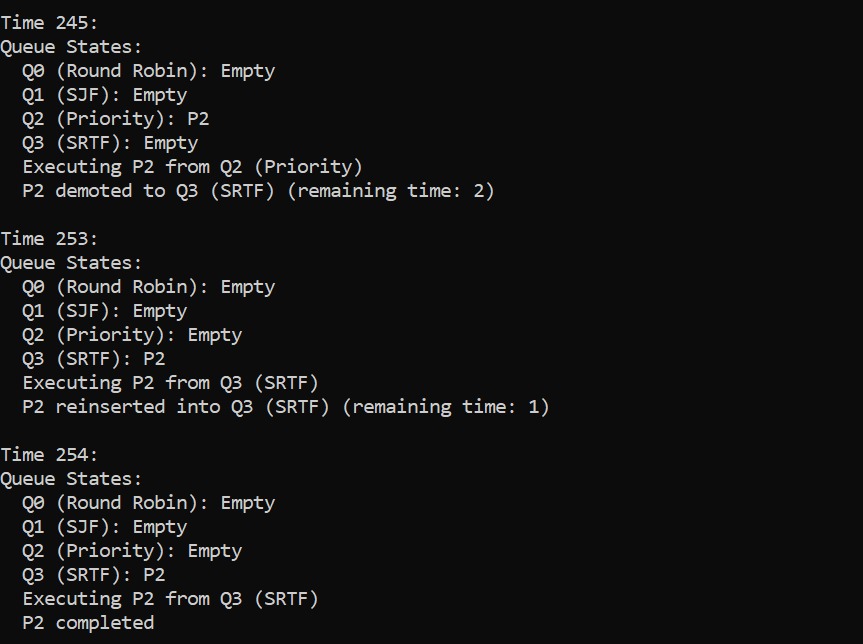
**3.Methodology**

The Multi-Level Feedback Queue (MLFQ) scheduler was developed in C++ with three distinct approaches, each utilizing four queues (Q0–Q3) with different scheduling algorithms. Processes begin in Q0 and are demoted to lower queues based on execution behavior or promoted to higher queues to prevent starvation (using a 5-unit wait threshold). Each approach was designed to test the interplay of preemptive and non-preemptive algorithms, with process attributes like burst time and priority influencing queue transitions. Below, each approach is detailed with its queue assignments and operational logic.

**3.1 First Approach**

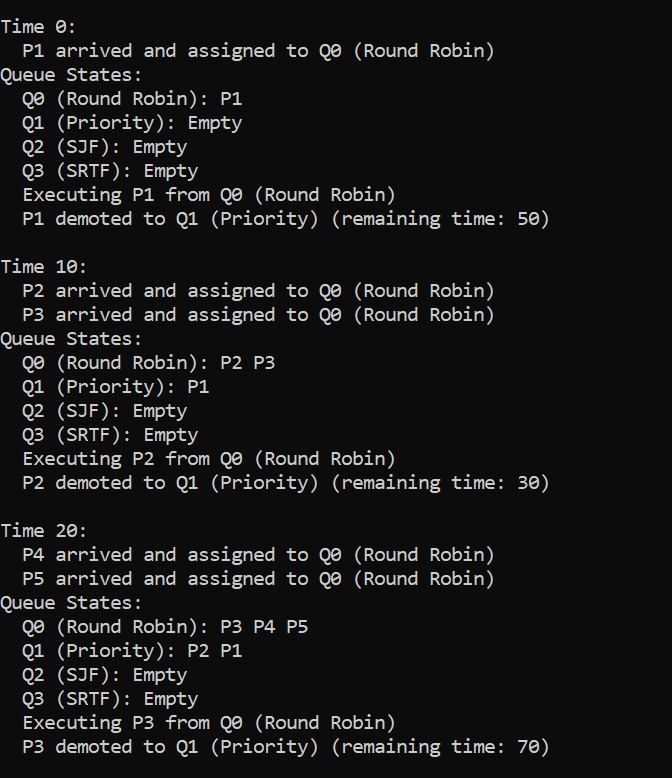
In the first approach, Q0 employs Round Robin (preemptive, quantum=10), executing processes for a fixed time slice before demoting them to Q1 if incomplete. Q1 uses Shortest Job First (SJF) (non-preemptive), selecting the process with the shortest burst time and running it to completion. Q2 implements Priority Scheduling (non-preemptive), executing the process with the highest priority (lowest priority number) until finished. Q3 applies Shortest Remaining Time First (SRTF) (preemptive), running the process with the least remaining time for 1-unit slices, preempting for higher-priority queues or shorter jobs. Processes rarely reach Q2 or Q3 due to Q1’s efficiency in completing short jobs.

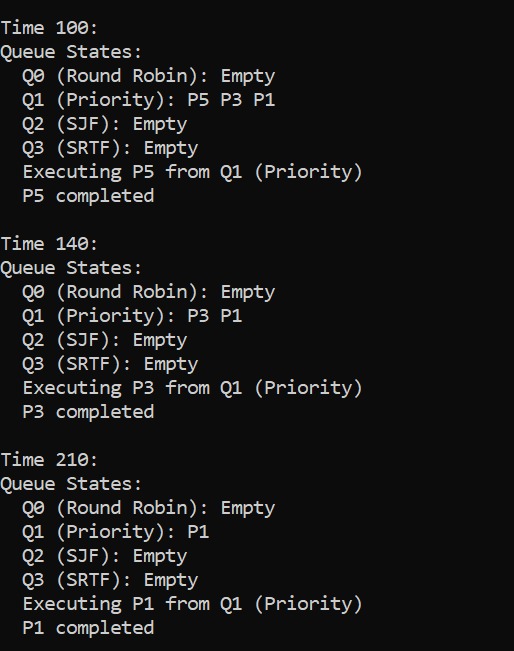




**3.2 Second Approach**

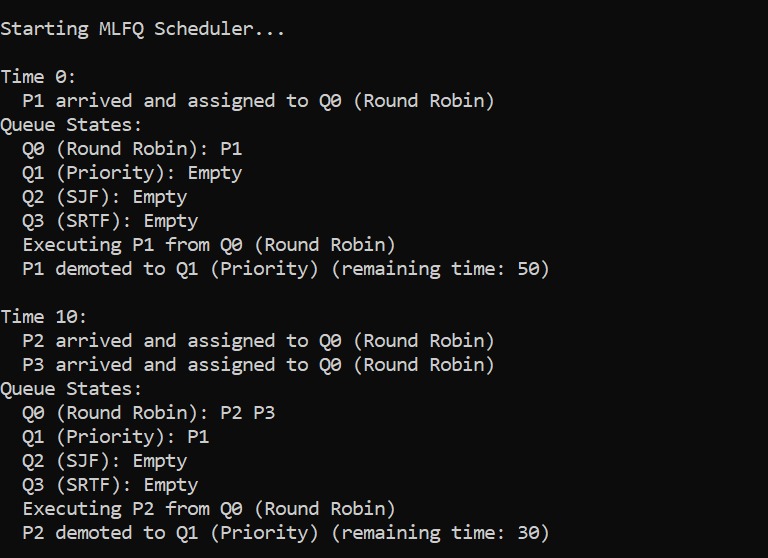
The second approach assigns Q0 to Round Robin (preemptive, quantum=10), demoting processes to Q1 after their time slice. Q1 uses Priority Scheduling (non-preemptive), running the highest-priority process (lowest priority number) to completion without interruption. Q2 employs Shortest Job First (SJF) (non-preemptive), completing the process with the shortest burst time. Q3 uses Shortest Remaining Time First (SRTF) (preemptive), executing 1-unit slices of the process with the least remaining time. Most processes complete in Q0 or Q1, as Q1’s non-preemptive nature retains processes until finished, limiting Q2 or Q3 usage.

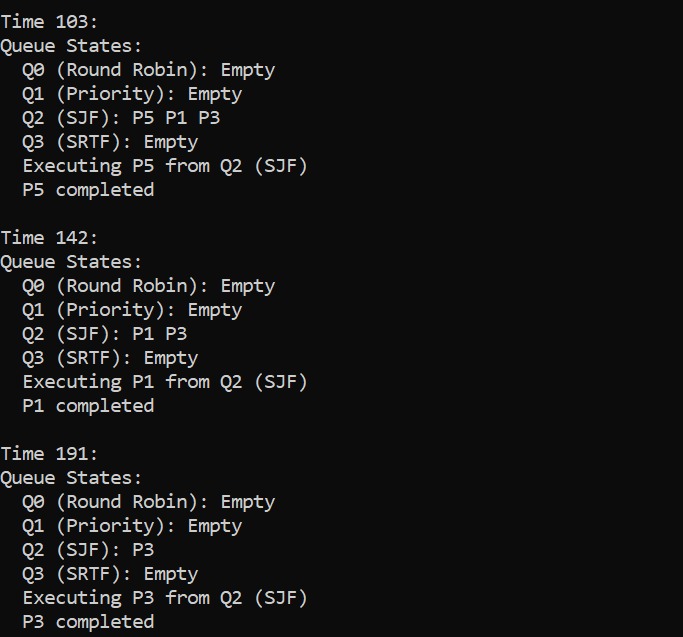




**3.3 Third Approach**

The third approach, estimated the most effective, uses Q0 for Round Robin (preemptive, quantum=10), demoting processes to Q1 after their slice. Q1 implements Priority Scheduling (preemptive), executing the highest-priority process for 1-unit slices, preempting for new Q0 arrivals or higher-priority Q1 processes, and demoting processes with priority > 5 to Q2. Q2 applies Shortest Job First (SJF) (non-preemptive), completing the shortest burst time process. Q3 uses Shortest Remaining Time First (SRTF) (preemptive), running 1-unit slices of the least remaining time process. This approach ensures balanced queue utilization by leveraging Q1’s preemption and demotion to distribute processes to Q2 or Q3.





**4. Challenges**

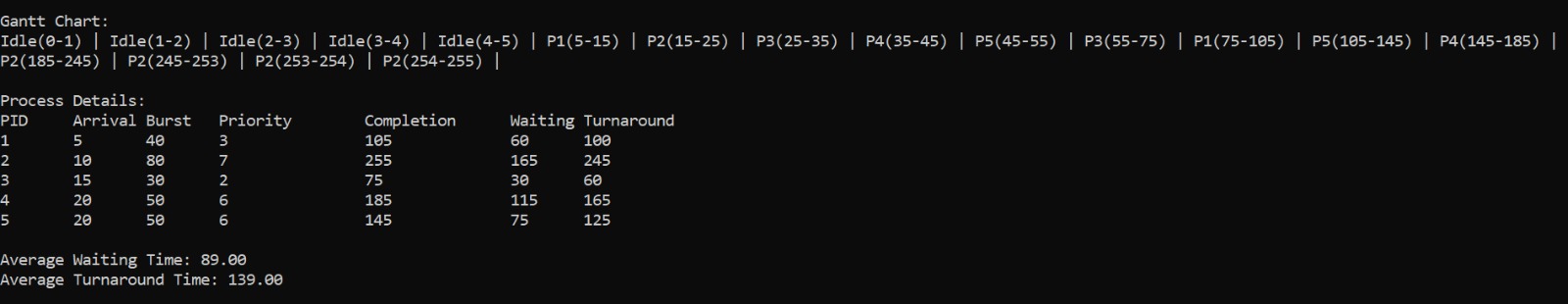
Implementing the MLFQ scheduler presented several challenges. Ensuring seamless context switching between queues required precise state management. In the first approach, CPU utilization was lower due to frequent idle periods when processes were stuck in lower queues Q2, Q3, which were underutilized. Balancing preemption in Q1 for the third approach was complex, as overly frequent preemptions increased overhead. Debugging demotion logic to ensure processes reached Q2 or Q3 without overloading Q0 or Q1 was time-consuming. Input validation for burst times and priorities also needed careful handling to prevent runtime errors.

**5. Optimization**

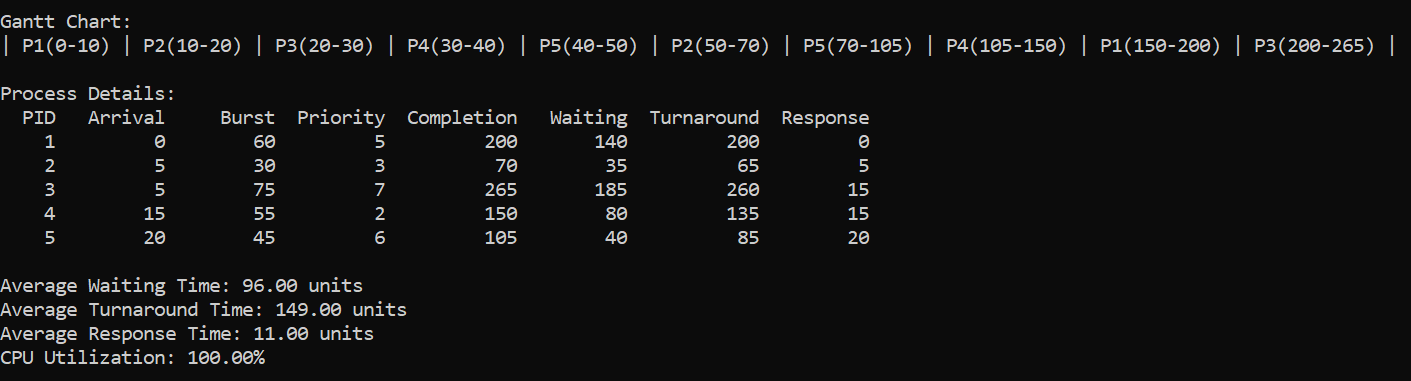
To enhance performance, several optimizations were explored. The quantum for Q0 was set to 10 units to balance responsiveness and throughput. In the third approach, a demotion threshold of priority > 5 for Q1 ensured low-priority processes moved to Q2, improving queue utilization. Adjusting this threshold (e.g., to priority > 3) was tested to increase Q2 or Q3 activity, reducing waiting times. The starvation threshold was fine-tuned to 5 units to prevent long-running processes from hogging lower queues, enhancing fairness across all approaches.

**6. Results**

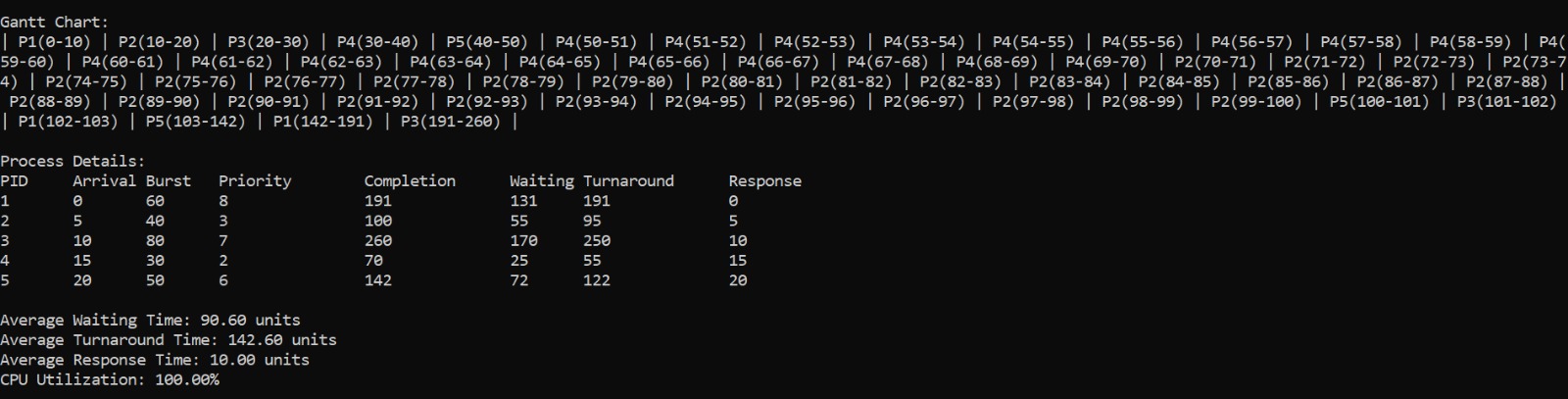
The three approaches were evaluated using a consistent workload: 5 processes, burst times 30–80, priorities 2–8. The first approach achieved an average waiting time of 89 units and turnaround time of 139 units but had lower CPU utilization of <100% due to idle periods.



The second approach yielded an average waiting time of 96 units, turnaround time of 149 units, and 100% CPU utilization.



The third approach recorded an average waiting time of 90 units, turnaround time of 142 units, and 100% CPU utilization. In the first and second approaches, all processes executed in Q0 or Q1, underutilizing Q2 Q3. The third approach, with preemptive Priority Scheduling in Q1, effectively used Q2-SJF and Q3-SRTF by demoting low-priority processes, ensuring balanced queue utilization.



|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Approach 1** | **Approach 2** | **Approach 3** |
| **Avg. Waiting Time** | 89 units | 96 units | 90 units |
| **Avg. Turnaround Time** | 139 units | 149 units | 142 units |
| **CPU Utilization** | <100% | 100% | 100% |
| **Queue Utilization** | Low Q2/Q3 use | Low Q2/Q3 use | Balanced |

**7. Conclusion**

The MLFQ scheduler successfully demonstrated the impact of different scheduling algorithms on system performance. The third approach, with preemptive Priority Scheduling in Q1, proved superior due to its effective use of all queues, ensuring fairness and adaptability despite higher waiting times compared to the second approach. The first approach offered competitive metrics but failed to utilize lower queues, while the second approach achieved the lowest waiting time but lacked Q2 or Q3 execution.