MLFQ Scheduler - Multilevel Feedback Queue Implementation

Operating Systems Course Project Proposal

Ayyan, Aisha, Hammad
Data Science Department

Submitted to: Sir Fahim
November 19, 2025

1 Project Overview

The MLFQ (Multilevel Feedback Queue) Scheduler is a C++ implementation of a CPU scheduling algorithm that demonstrates key concepts in Operating Systems. This project implements a sophisticated scheduling algorithm that uses multiple priority queues with different time quantums to balance system responsiveness and efficiency.

1.1 Objective

The primary objective of this project is to:

- Implement the MLFQ scheduling algorithm as described in operating systems literature
- Demonstrate how different process types are handled with varying priority levels
- Provide a visual representation of the scheduling process
- Allow for experimentation with different parameter configurations

2 Problem Statement

In operating systems, CPU scheduling is a critical component that determines which process gets access to the CPU at any given time. Traditional scheduling algorithms like First-Come-First-Served, Shortest Job First, or Round Robin have limitations when dealing with diverse process types (I/O-intensive vs CPU-intensive). The challenge is to implement a scheduling algorithm that can:

- Provide good response time for interactive processes
- Ensure CPU-intensive processes get completed in reasonable time

- Prevent process starvation
- Adapt to changing process behavior

3 MLFQ Algorithm Description

The Multilevel Feedback Queue (MLFQ) algorithm addresses these challenges using multiple queues with different priority levels and time quantums:

3.1 Key Rules

1. Priority-based execution: If Priority(A) ¿ Priority(B), A runs before B
2. Round-robin within queues: If Priority(A) == Priority(B), processes run in round-robin fashion
3. Priority demotion: When a process uses its complete time quantum, it moves to a lower priority queue
4. Priority boost (aging): After a certain period, all processes are moved to the highest priority queue to prevent starvation

3.2 Queue Structure

- Multiple priority queues (typically 3-5 levels)
- Each queue has its own time quantum (exponentially increasing from highest to lowest priority)
- New processes start in the highest priority queue
- Processes can move between queues based on their behavior

4 Proposed Implementation

4.1 Technical Approach

- Programming Language: C++17
- Build System: CMake
- Architecture: Modular design with separate classes for Process, Queue, Scheduler, and Visualizer
- Memory Management: Smart pointers for automatic memory management
- Visualization: Terminal-based with optional FLTK GUI

4.2 System Components

1. Process Class: Represents a process with state, timing metrics, and execution methods
2. ProcessQueue Class: Manages a single priority level queue
3. MLFQScheduler Class: Core scheduling algorithm implementing MLFQ rules
4. Visualizer Class: Terminal-based visualization of queues and process execution
5. Main Program: User interface with multiple execution modes

4.3 Features

- Interactive and automatic execution modes
- Configurable parameters (number of queues, time quantums, boost interval)
- Real-time visualization of all queues and current process
- Process metrics (wait time, turnaround time, response time, CPU utilization)
- Gantt chart generation
- Multiple scheduling algorithm options for the last queue (Round Robin, Shortest Job First, Priority Scheduling)
- Example process sets for demonstration

5 Project Scope

5.1 In Scope

- Complete MLFQ algorithm implementation
- Process scheduling with all 4 MLFQ rules
- Terminal-based visualization
- Multiple execution modes (interactive, automatic, quick)
- Metrics calculation and display
- Configuration management
- Example process sets with different characteristics
- Optional FLTK GUI support

5.2 Out of Scope

- Advanced process synchronization mechanisms
- Memory management simulation
- I/O scheduling
- Real hardware interaction
- Multi-processor scheduling beyond basic concepts

6 Timeline and Milestones

7 Technical Specifications

7.1 Build Requirements

- C++17 compatible compiler (GCC 7+, Clang 5+, MSVC 2017+)
- CMake 3.10 or higher
- FLTK library (optional, for GUI support)
- Make or Ninja build system

7.2 Performance Requirements

- Support for up to 20 processes simultaneously
- Real-time visualization updates
- Execution time complexity: O(N) per time step where N is the number of processes
- Memory usage: O(N + Q) where Q is the number of queues

Milestone	**Planned Date**	**Deliverable**
Core Algorithm	Week 1	Basic MLFQ scheduler with multiple queues
Process Management	Week 2	Process creation, execution, and tracking
Visualization	Week 3	Terminal-based display of queues and execution
Testing	Week 4	Unit tests and integration testing
Documentation	Week 4	Complete project documentation
GUI Integration	Week 5	Optional FLTK GUI implementation
Final Review	Week 5	Project completion and review

Table 1: Project Timeline

8 Expected Outcomes

Upon completion of this project, the following outcomes are expected:

1. A fully functional MLFQ scheduler implementing all 4 core rules
2. Comprehensive understanding of CPU scheduling algorithms
3. Practical experience with C++ object-oriented design
4. Working visualization system that demonstrates the algorithm behavior
5. Measurable performance metrics showing the effectiveness of MLFQ
6. Educational tool for further study of operating system concepts

9 Conclusion

The MLFQ Scheduler project provides a comprehensive implementation of important operating systems concepts. The project will demonstrate the practical application of the MLFQ algorithm in balancing responsiveness and efficiency for different types of processes. The implementation will serve as both an educational tool and a foundation for future research in scheduling algorithms. This project will enhance understanding of operating system internals, particularly the critical component of process scheduling, and provide hands-on experience with system-level algorithm implementation and visualization.