Multi-Level Feedback Queue (MLFQ) Scheduler Implementation on xv6-RISC-V

Muhammad Hussain - 29004
Sarfaraz Ahmed - 24520

December 4, 2025

Abstract

This report summarizes the implementation of a Multi-Level Feedback Queue (MLFQ) scheduler for xv6-RISC-V. The work includes the design of priority queues, scheduling logic, system calls for process monitoring, and test programs for evaluating CPU-bound and I/O-bound behavior. The new scheduler improves responsiveness, prevents starvation through periodic boosting, and provides observable metrics for debugging and analysis.

Contents

1 Introduction

The default xv6-RISC-V scheduler uses a simple round-robin mechanism that does not differentiate between CPU-bound and I/O-bound processes. This leads to low responsiveness for interactive processes. A Multi-Level Feedback Queue (MLFQ) scheduler addresses this by employing multiple priority levels, dynamic priority adjustment, and different time quanta.

The objective of this project is to replace xv6's round-robin scheduler with a fully functional MLFQ design, including data structures, queue operations, priority management, and system-level monitoring tools.

1.1 Goals

- Prioritize interactive processes using small time quanta.
- Gradually demote CPU-bound processes.
- Prevent starvation through periodic priority boosting.
- Provide system calls for process and scheduler statistics.

2 MLFQ Design Overview

The scheduler consists of four priority queues (Q0–Q3) with exponentially increasing quanta:

Level	Queue	Quantum	Behavior
0	Q0	2 ticks	Interactive
1	Q1	4 ticks	Mixed
2	Q2	8 ticks	CPU-bound
3	Q3	16 ticks	Lowest priority

Processes start in Q0. If they use up their quantum without yielding, they are demoted. If they sleep or block (typical of I/O-bound workloads), they remain at the same priority. To avoid indefinite demotion, all processes are periodically boosted to Q0.

3 Data Structures and System Calls

3.1 Process Structure Extensions

The following fields were added to `struct proc`:

```
int queue_level;
uint64 time_in_queue;
uint64 time_slices;
struct proc *queue_next;
```

They track priority, time usage, and queue linkage.

3.2 Queue Representation

Each queue is implemented as a simple linked list stored globally inside the scheduler. Operations include enqueue, dequeue, and removal during sleep/exit.

3.3 System Calls

Three system calls were added:

1. `getprocinfo()`: Returns a process's queue level and time statistics.
2. `boostproc()`: Boosts a specific PID to Q0 (for testing).
3. `mlfq_stats()`: Returns global statistics such as queue lengths and boosts.

4 Scheduler Implementation

4.1 Scheduling Loop

The scheduler iterates from Q0 down to Q3, selecting the first runnable process:

```
for(int level = 0; level < 4; level++) {
    struct proc *p = dequeue(level);
    if(p && p->state == RUNNABLE) {
        // Run p for its quantum
    }
}
```

4.2 Quantum Enforcement

A process runs until:

- It blocks (I/O-bound).
- It finishes (exit).
- It exhausts its time quantum (CPU-bound).

On quantum expiration, the process is demoted unless already in Q3.

4.3 Starvation Prevention

MLFQ algorithms may cause long-running CPU-bound processes to stay indefinitely in low-priority queues. To prevent this starvation, xv6 performs a periodic system-wide priority boost.

Every fixed interval (e.g., every 200 timer ticks), all processes—regardless of their queue level—are moved back to the highest-priority queue (Q0). This ensures:

- Low-priority processes regain CPU access.
- Long-running workloads do not remain stuck in Q3.

- The system behaves fairly over long periods.

The mechanism is implemented in the timer interrupt:

```
if(ticks % BOOST_INTERVAL == 0) {
    boost_all_processes();
}
```

This global boost restores fairness while still preserving the feedback nature of the scheduler.

4.4 Priority Boosting

Every fixed interval (e.g., 200 ticks), all processes are moved to Q0:

```
if(ticks % BOOST_INTERVAL == 0)
    boost_all_processes();
```

This prevents starvation and restores responsiveness.

5 Testing and Evaluation

5.1 CPU-Bound Tests

A loop-intensive program was used to verify:

- Demotion from Q0 $\rightarrow$ Q1 $\rightarrow$ Q2 $\rightarrow$ Q3.
- Increasing quantum lengths.

Tracing using `getprocinfo()` showed correct demotion patterns.

5.2 I/O-Bound Tests

A program repeatedly performing `sleep()` calls demonstrated:

- Processes remain in Q0.
- High responsiveness and minimal delay.

5.3 Boosting Tests

Through long-running CPU-bound workloads, starvation prevention was confirmed:

- Processes in Q3 were periodically restored to Q0.
- All processes eventually received CPU time.

6 Conclusion

This project successfully implemented a complete MLFQ scheduler for xv6-RISC-V, including priority-based queues, dynamic feedback, starvation prevention, and system call support for monitoring. CPU-bound processes are gradually demoted, while I/O-bound processes retain high responsiveness. The scheduler meets classical MLFQ goals and provides a clean extension to the original xv6 kernel.