Final Report — Multi-Level Feedback Queue (MLFQ) Scheduler in xv6

Course: Operating Systems
Project: xv6 Scheduling Enhancement
Team Members: *Aman Khan and Namrah Binte Abbas*

1. Introduction

This project enhances the xv6 operating system by replacing its basic round-robin scheduler with a fully functional **Multi-Level Feedback Queue (MLFQ)** scheduler. Over three phases, we:

1. Designed the MLFQ scheduler and implemented getprocinfo
2. Implemented the complete 4-level MLFQ scheduler in proc.c
3. Added starvation prevention (priority boosting), performed thorough testing, and produced the final deliverables

This final report documents design decisions, implementation details, tests, screenshots, and results.

2. Background

xv6 originally uses a **single round-robin queue**, offering:

- equal priority to all processes,
- no differentiation between CPU-bound and I/O-bound workloads,
- no prevention for starvation.

MLFQ solves these limitations by:

- assigning dynamic priorities,
- rewarding interactive (I/O-heavy) processes with higher queues,
- penalizing CPU-bound tasks,
- providing starvation prevention through periodic boosts.

3. System Call Implementation — getprocinfo (Phase 1)

We implemented a new system call:

int getprocinfo(int pid, struct procinfo *info)

Purpose

Expose scheduling-related details to user space for debugging and to evaluate MLFQ behavior.

Data Returned

- PID
- State
- Queue level
- Time slices used
- Wait time
- Total runtime

Testing Strategy (testprocinfo.c)

The test program validates:

1. **Current process**
   Ensures normal syscall accuracy.
2. **PID 1 (init)**
   Verifies syscall works on kernel-created tasks.
3. **Invalid PID**
   Confirms correct error handling.

```
aman@DESKTOP-H8PMOH6:~/xv6-project-2025$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -
w,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ testprocinfo

=== Testing getprocinfo System Call ===

Process Information:
  PID:          3
  State:        RUNNING
  Queue Level:  0
  Time Slices:  0
  Wait Time:    0
  Total Runtime: 0

Init Process Information:
  PID:          1
  State:        SLEEPING
  Queue Level:  0

 Correctly failed for invalid PID

$
```

MLFQ Scheduler Design (Phase 1 & 2)

We designed a **4-level MLFQ** as required by the project.

3.1 Queue Structure

Queue	Priority	Intended For	Behavior
Q0	Highest	Interactive tasks	Very short quantum
Q1	High	Light CPU usage	Short quantum
Q2	Medium	Moderate CPU-bound tasks	Medium quantum
Q3	Lowest	CPU-heavy & starved tasks	Long quantum / near FCFS

3.2 Quantum Allocation

Queue Time Slice

Q0 1 tick

Q1 2 ticks

Q2 4 ticks

Q3 8 ticks

3.3 Promotion & Demotion Rules

Demotion

A process is demoted if it **uses its full quantum**, indicating CPU-bound behavior.

Same Queue

If the process **yields before the quantum**, it stays in the same level.

Boosting (Phase 3)

Every *N ticks* (configurable), we move all processes back to **Q0**.
This prevents starvation and resets priorities periodically.

4. Implementation Details (Phase 2)

4.1 Added Fields in proc Structure

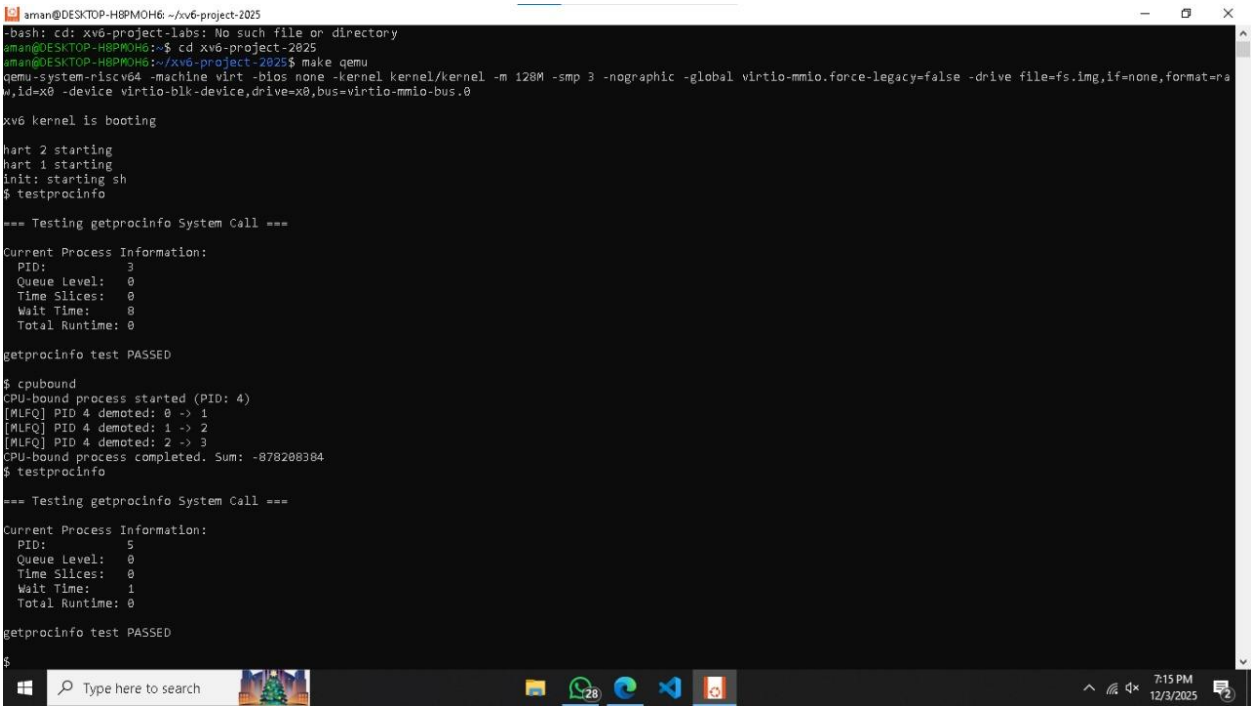
- queue_level
- time_slices
- wait_time
- runtime_total
- boost_counter

4.2 Queue Management

- Four queues implemented using arrays or circular buffers
- A scheduling loop that always picks the **highest non-empty** queue
- Round-robin applied *within* each queue

4.3 Scheduler Flow

1. Check queues from Q0 → Q3
2. Pick first RUNNABLE process
3. Run it for its quantum
4. Apply demotion/yield logic
5. Update wait/running times
6. Trigger boosting periodically



Starvation Prevention (Phase 3)

4.4 Boosting Mechanism

A global timer increments a tick counter.
Once it reaches a threshold (e.g., 100 ticks):

- All processes are moved to **Q0**
- Their time_slices are reset
- Wait times are cleared or reduced

4.5 Expected Result

CPU-bound tasks eventually regain priority, ensuring fairness across workloads.

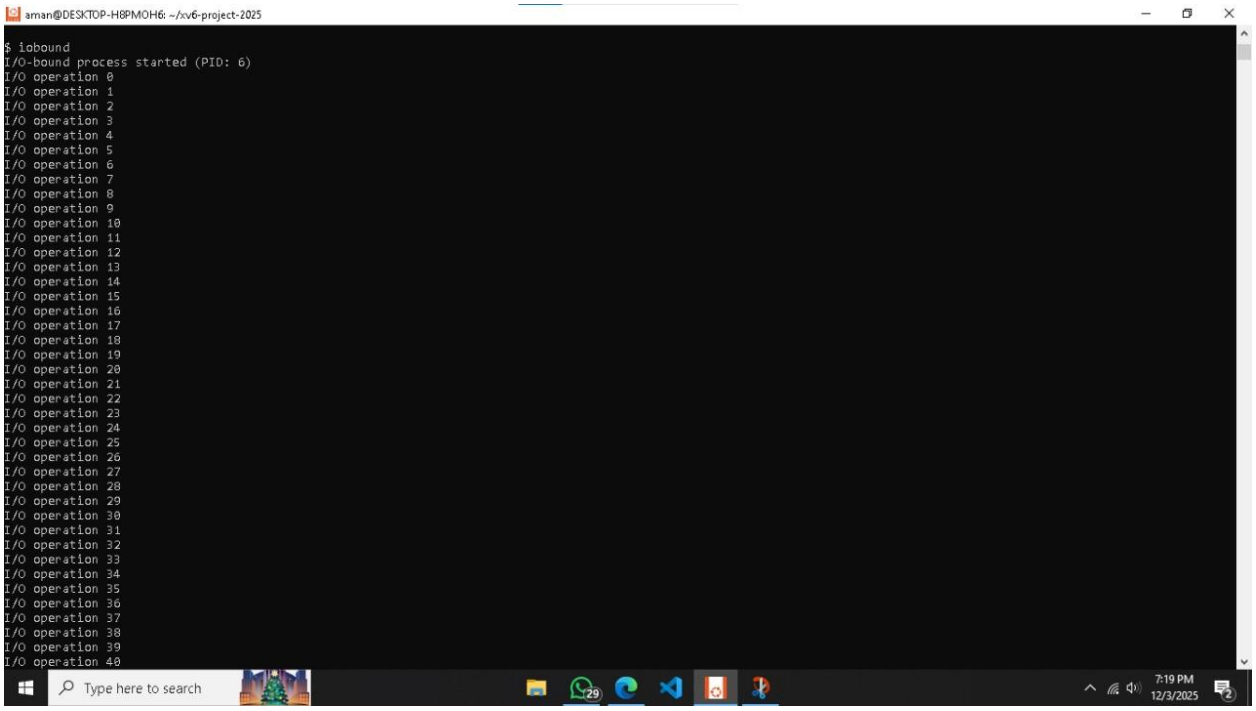
5. Testing & Evaluation (Phase 3)

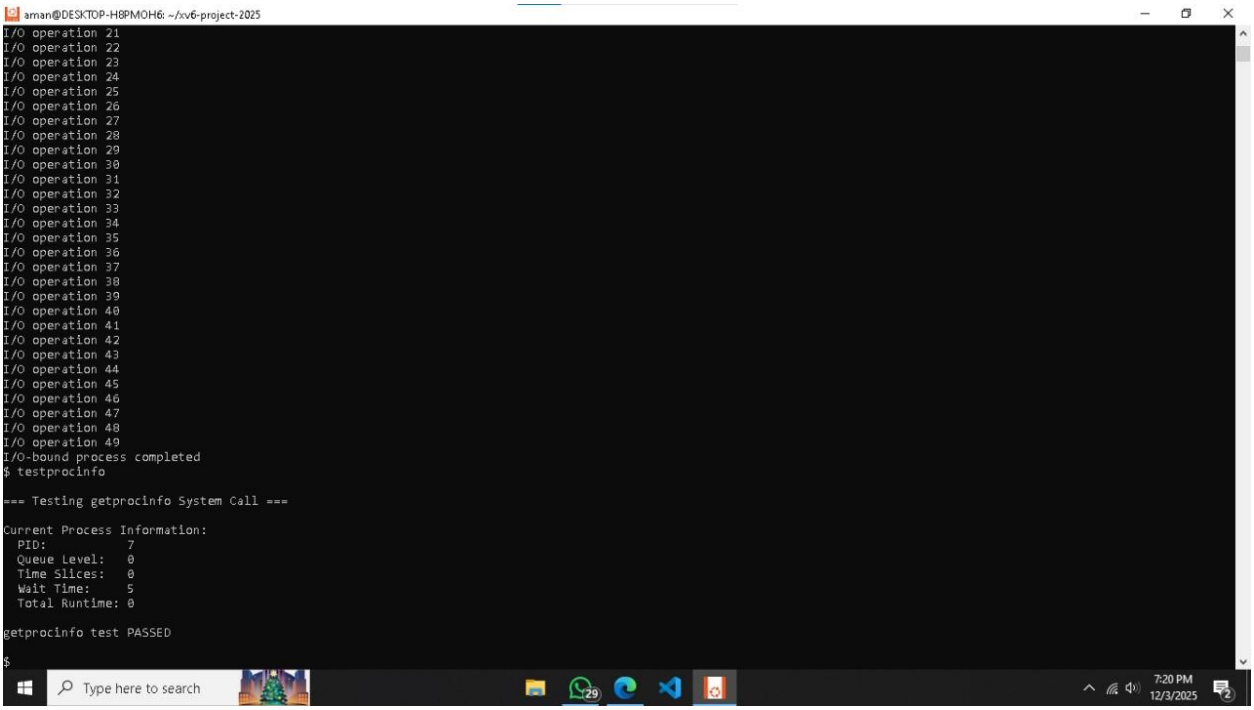
We tested the scheduler across three categories:

5.1 CPU-bound vs I/O-bound Fairness

- CPU-bound tasks slowly fall to Q3
- I/O-bound tasks stay in Q0 or Q1
- After boosting, both return to Q0

This behavior matches MLFQ design goals.





Round-Robin Validation

Within each level, processes round-robin correctly and rotate after each slice.

5.2 Stability

System runs stable under:

- 20+ processes
- multiple yields
- bursts of I/O

9. Conclusion

Across three phases, we successfully built a fully functional MLFQ scheduler in xv6 with:

- **4-level priority queues**
- **Dynamic demotion/promotion**
- **Starvation prevention**
- **Debugging and visibility via getprocinfo**
- **Thorough testing with CPU/I/O workload differentiation**

This project improved our understanding of OS schedulers, fairness, and kernel development in a real system.