Assignment Report: Multi-Level Feedback Queue Scheduler Implementation

Meng Shuhan - 123090422

1 Introduction

This assignment aims to implement and extend the process scheduler of xv6, replacing the original Round-Robin (RR) scheduler with a Multi-Level Feedback Queue (MLFQ) scheduler. The MLFQ scheduler dynamically adjusts process priorities based on their CPU and I/O behavior, aiming to improve responsiveness for interactive processes while maintaining fairness for CPU-bound processes.

In this work, I modified xv6 kernel to include multiple scheduling queues, dynamic priority promotion and demotion, and logging mechanisms to track scheduling decisions. Test programs, including a custom `Q3_test`, were used to verify scheduler correctness.

2 Design

2.1 Scheduler Design

The implemented scheduler is a Multi-Level Feedback Queue (MLFQ) scheduler with three priority levels (Q0, Q1, Q2). Key design points:

1. **Queues:**
   (a) Q0: highest priority with shortest time slice (e.g., 3 ticks).
   (b) Q1: mid-priority with slightly longer time slice (e.g., 6 ticks).
   (c) Q2: lowest priority with longest time slice (e.g., 12 ticks).
2. New processes start in Q0.
3. If a process uses its entire time slice quota, it moves to the next lower queue:
   - Q0 $\rightarrow$ Q1 $\rightarrow$ Q2
4. Processes that block on I/O (e.g., `sleep()`) return to their original queue when woken.
5. Every 100 ticks, all processes in Q1 and Q2 are promoted to Q0 to avoid starvation.
6. For processes in the same queue, they are scheduled in a Round Robin manner.

2.2 Key Code Modifications

- **Data Structures**: Added `qlevel` and `time_used` fields to `struct proc` to track the current queue and elapsed ticks.
- **Scheduler Logic**: In `proc.c`, the main `scheduler()` function iterates over queues in priority order, selecting RUNNABLE processes, updating `time_used`, and handling demotion/promotion.
- **Logging**: Introduced a global `schedlog_enabled` flag and a helper function `schedlog_print()` to output formatted scheduling events (process creation, start, sleep, wakeup, demotion, promotion, and exit). This ensures logging occurs only when the `Q3_test` program runs.
- **Syscall Interface**: Added a new syscall `schedlog(on)` to enable or disable logging from user space.

2.3 Execution Flow with Logging

1. The `Q3_test` program calls `schedlog(1)` to enable scheduler logging.
2. Processes forked by `Q3_test` execute CPU- or I/O-bound workloads.
3. The kernel prints log lines.
4. After the program completes, `schedlog(0)` disables further logging.

3 Environment and Execution

- **Environment**: xv6-riscv, QEMU emulator, Ubuntu 20.04 host, RISCV64 toolchain.
- **Compilation**:
- **Execution**: Important note!: Here you can't make clean before make qemu, because user/usys.S file will be rewrited.

```
make qemu
$ Q3_test
```

- The output displays scheduler events in chronological order. Screenshots of sample runs show CPU and I/O processes starting, sleeping, waking, and finishing, matching the expected MLFQ behavior.

4 Test Cases:Provided Test Case

I modify the parameters in Q3_test:

```
cpu_worker(1, 200000);
io_worker(1, 2000);
cpu_worker(1, 250000);
cpu_worker(2, 180000);
io_worker(2, 2200);
```

The `Q3_test` program forks a mix of CPU-bound and I/O-bound processes. Expected behaviors:

- CPU-bound processes gradually demote through lower queues after consuming their time slices.
- I/O-bound processes frequently return to higher-priority queues after sleeping.
- Scheduler log shows start, run, sleep, wakeup, demotion, promotion, and exit events.

The tests verified that the scheduler respects queue priority and time slices, and logs match expected output formats.

5 Challenges and Solutions

- **Logging without affecting scheduling behavior**: Introduced `schedlog_enabled` flag to avoid printing during normal execution, ensuring logs appear only when intended.

6 Comparison with Original RR Scheduler

- **Throughput**: Comparable overall throughput for CPU-bound processes, but interactive I/O-bound processes experience lower latency.
- **Response Time**: MLFQ improves average response time for short and interactive processes due to higher-priority queue placement.
- **Fairness**: CPU-bound processes gradually demote, ensuring they do not starve I/O-bound processes.
- **Logging Advantage**: Scheduler logs allow precise observation of scheduling decisions, not possible in plain RR.

7 Conclusion

This assignment demonstrates the implementation of a Multi-Level Feedback Queue scheduler in xv6. By modifying the kernel scheduler and introducing logging, I verified the dynamic promotion and demotion of processes and observed improved responsiveness for interactive tasks compared to the original RR scheduler. I also learned careful kernel modification techniques, time accounting, and the importance of detailed logging for verification.