MLFQ Scheduler – Week 1

Muhammad Hussain - 29004
Sarfaraz Ahmed - 24520

December 4, 2025

Overview

This week implements the foundational components required for the Multi-Level Feedback Queue (MLFQ) scheduler in xv6-RISC-V. Work includes defining priority queues, integrating fields into the process structure, and adding the `getprocinfo()` system call. These components prepare xv6 for dynamic priority scheduling in later weeks.

1 MLFQ Architecture

The scheduler uses four queues with exponentially increasing quanta:

Level	Queue	Quantum	Priority	Use
0	Q0	2	Highest	Interactive
1	Q1	4	High	Mixed
2	Q2	8	Low	CPU-bound
3	Q3	16	Lowest	Heavy CPU-bound

Promotion, Demotion, Boosting

Processes that use their entire quantum are demoted (Q0→Q1→Q2→Q3). If a process yields before using its full quantum, it stays at its level—capturing I/O-bound behavior. Every 100 ticks, all RUNNABLE processes are boosted to Q0 to prevent starvation.

2 Kernel Data Structures

Four fields were added to `struct proc`:

```
int queue_level;        // Current queue (0-3)
uint64 time_in_queue;   // Ticks in current quantum
uint64 time_slices;     // Total slices received
struct proc *queue_next;// For queue linkage
```

User-space accesses information via:

```
struct procinfo {
  uint64 pid;
  uint64 queue_level;
  uint64 time_in_queue;
  uint64 time_slices;
};
```

These fields are initialized in `allocproc()` and cleared in `freeproc()`.

3 getprocinfo() System Call

The new syscall (ID 22) returns MLFQ metadata about the calling process:

```
uint64 sys_getprocinfo(void) {
  uint64 uptr; struct procinfo info;
  if(argaddr(0,&uptr)<0) return -1;
  struct proc *p=myproc();

  info.pid=p->pid;
  info.queue_level=p->queue_level;
  info.time_in_queue=p->time_in_queue;
  info.time_slices=p->time_slices;

  if(copyout(p->pagetable, uptr,
     (char*)&info, sizeof(info)) < 0) return -1;
  return 0;
}
```

User program example:

```
struct procinfo i;
getprocinfo(&i);
printf("%d %d %d\n", i.pid, i.queue_level, i.time_in_queue);
```

4 Build & Verification

```
$ make clean && make && make qemu
$ getprocinfo
```

Expected behavior: new processes start in Q0; `time_slices` increments as they run; xv6 remains fully functional with round-robin scheduling for each queue.

Conclusion

Week 1 establishes a complete MLFQ data infrastructure: priority queues, process metadata, a userspace query syscall, and lifecycle integration. These components allow fair scheduling, starvation prevention, and accurate priority tracking for the full MLFQ scheduler in upcoming stages.