【OS】Day13(2)

【Ch8】Scheduling: Multi-Level-Feedback Queue(2)

8.3 Attempt #2: The Priority Boost

What could we do in order to guarantee that CPU-bound jobs will make some progress(even if it is not much?)

The simple idea is to periodically boost the priority of all the jobs in the system. There are many ways to achieve this, but let's do somethign simple: throw them all in the topmost queue; hence, a new rule:

- Rule 5: After some time period S, move all the jobs in the system to the topmost queue.

Our new rule solves two problems at once:

1. First, processes are guaranteed not to starve: by sitting in the top queue, a job will share the CPU with other high-priority jobs in a round-robin fashion, and evetually receive service.
2. Second, if a CPU-bound job has become interactive, the scheduler treats it properly once it has received the priority boost.

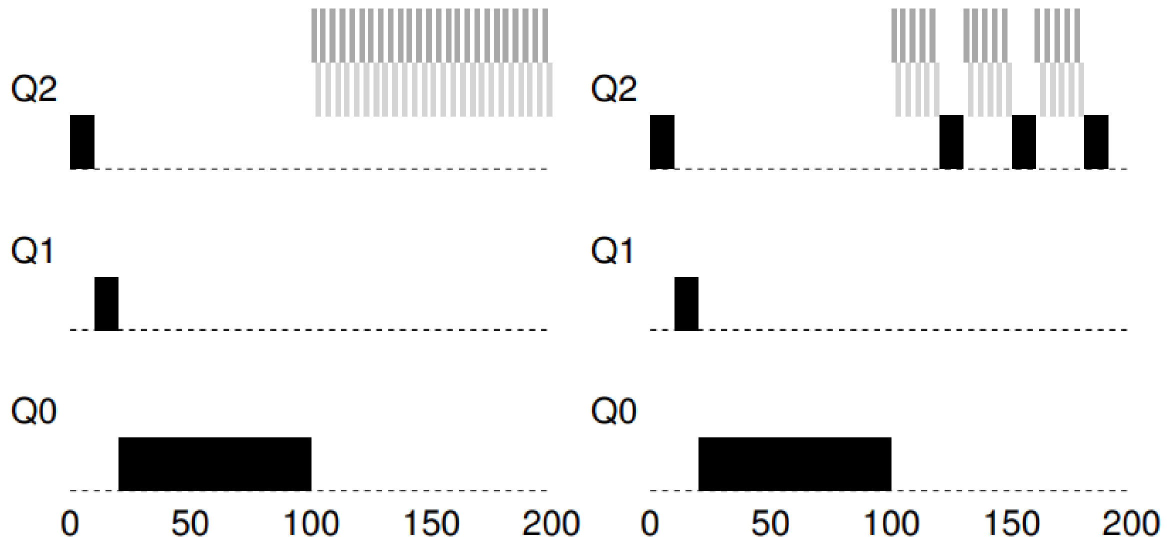

Figure 8.5: **Without (Left) and With (Right) Priority Boost**

8.4 Attempt #3: Better Accounting

We now have one more problem to solve: *how to prevent gaming of our scheduler?*

The solution here is to perform better accounting of CPU time at each level of the MLFQ. Instead of forgetting how much of a time slice a process used at a given level, the scheduler should keep track; once a process has used its allotment, it is demoted to the next priority queue.

Whether it uses the time slice in one long burst or many small ones does not matter.

We thus rewrite Rule 4a and 4b into the following:

- Rule4: Once a job uses up its time allotmenet at a given level(regardless of how many times it has given up the CPU), its priority is reduced.

The following figure shows an example of implementing better accounting:

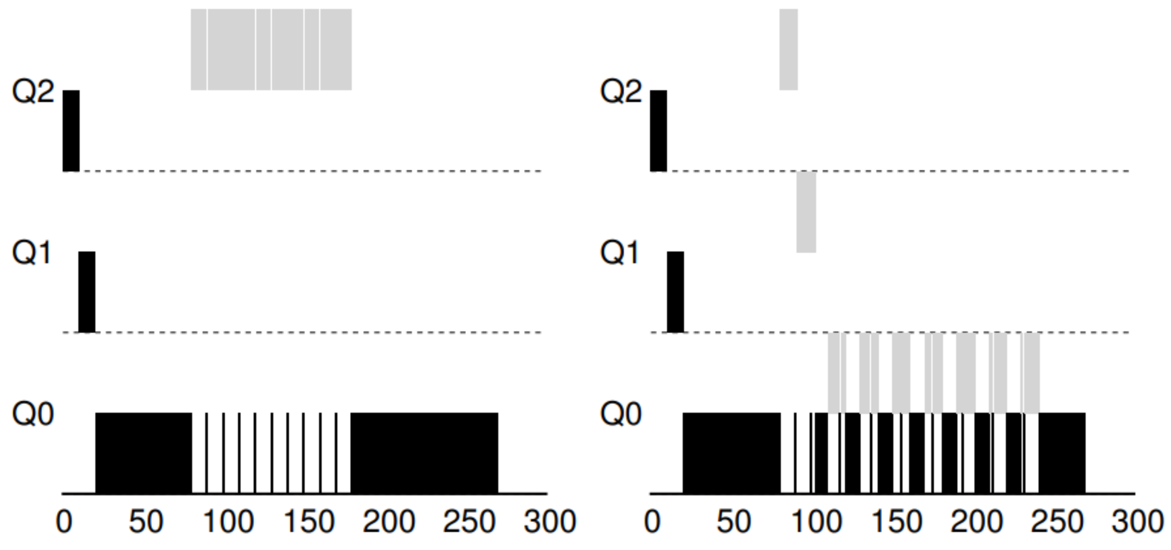

Figure 8.6: **Without (Left) and With (Right) Gaming Tolerance**

Without any protection from gaming, a process can issue an I/O jsut before a time slice ends and thus dominate CPU time.

8.5 Turning MLFQ and Other Issues

Most MLFQ variants allow for varying time-slice length across different queues:

- The high-priority queues are usually given short time slices; they are comprised of interactive jobs, after all ,and thus quickly alternating between them makes sense.
- The low-priority queues, in contrast, contain long-running jobs that are CPU-bound; hence, longer time slices work well.

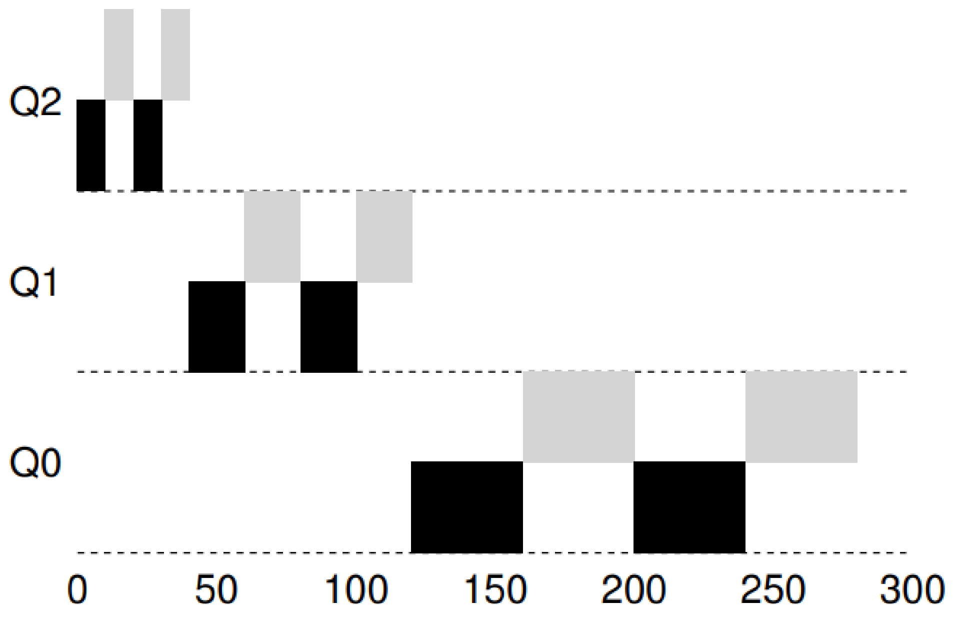

Figure 8.7: **Lower Priority, Longer Quanta**

Some systems reserve the highest priority levels for operating system work; thus typical user jobs can never obtain the highest levels of priority in the system.

8.6 MLFQ Summary

Hopefully now we can see why it is called MLFQ: it has multiple levels of queues, and uses feedback to determine the priority of a given job.

The refined set of MLFQ rules:

- Rule 1: If Priority(A) > Priority(B), A runs(B doesn't)
- Rule2: If Priority(A) = Priority(B), A&B run in Round-Robin fashion using the time slice(quantum length) of the given queue
- Rule 3: When a job enters the system, it is placed at the highest priority
- Rule 4: Once a job uses up its time allotmenet at a given level(regardless of how many times it has given up the CPU), its priority is reduced
- Rule 5: After some time period S, move all the jobs in the system to the topmost queue(highest priority level)