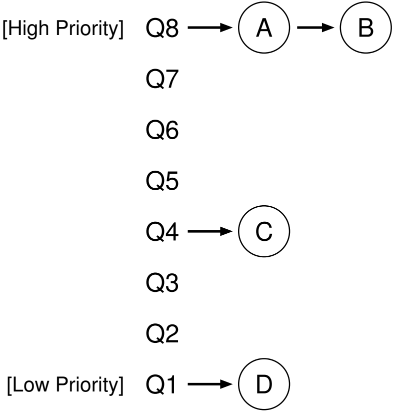


CS 250

OPERATING SYSTEMS

Lecture 5

The Multi-Level Feedback Queue

Instructor

Dr. Dhiman Saha

Relaxing #5 Workload Assumptions

- ~~1~~ ~~Each job runs for the same amount of time~~
- ~~2~~ ~~All jobs arrive at the same time~~
- ~~3~~ ~~Once started each job runs to completion~~
- ~~4~~ ~~All jobs only use the CPU (no I/O)~~
- ~~5~~ ~~The run-time of each job is known~~

Goal

Recall

- **Interactive** jobs care about **response** time
- **Batch** jobs care about **turnaround** time

Scheduling **without** perfect knowledge

How can we design a scheduler that both minimizes response time for interactive jobs while also minimizing turnaround time **without** a priori knowledge of job length?

Adaptive Scheduling

Recall

Algorithms like Round Robin **reduce response time** but are terrible for turnaround time. Why?

- How can the scheduler **learn, as the system runs, the characteristics of the jobs it is running**, and thus make better scheduling decisions?

Idea

Multiple levels of round-robin

Multi-Level Feedback Queue

- Due to Corbato et al. [1962]
- Later awarded Turing Award
- MLFQ has a number of distinct queues
- Each assigned a different **priority level**
- MLFQ uses **priorities** to decide which job should run at a given time

A job with higher priority is chosen to run.

Basic Rules

Rule 1

If $Priority(A) > Priority(B)$,
A runs (B doesn't).

Basic Rules

Rule 1

If $Priority(A) > Priority(B)$,
A runs (B doesn't).

Rule 2

If $Priority(A) = Priority(B)$,
A & B run in RR

Basic Rules

Rule 1

If $Priority(A) > Priority(B)$,
A runs (B doesn't).

Rule 2

If $Priority(A) = Priority(B)$,
A & B run in RR

Basic Rules

Rule 1 If $Priority(A) > Priority(B)$, A runs (B doesn't).

Rule 2 If $Priority(A) = Priority(B)$, A & B run in RR

Basic Rules

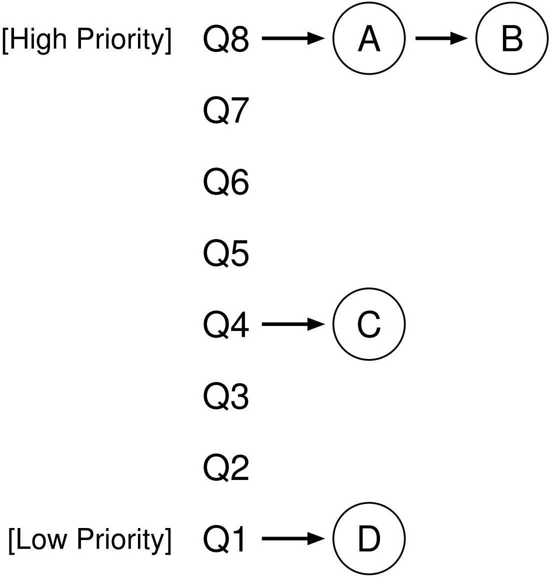
Rule 1

If $Priority(A) > Priority(B)$,
A runs (B doesn't).

Rule 2

If $Priority(A) = Priority(B)$,
A & B run in RR

Example



How to set priorities?

The key to MLFQ scheduling therefore lies in how the scheduler sets priorities

- Approach 1: User supplied. e.g. `nice`
- Approach 2: Observed behavior (learn from history)

Rather than giving a fixed priority to each job, MLFQ **varies** the priority of a job based on its observed behavior

How To Change Priority?

Rule 3

When a job enters the system, it is placed at the highest priority (the topmost queue).

Attempt #1 How To Change Priority?

Rule 3

When a job enters the system, it is placed at the highest priority (the topmost queue).

Rule 4a

If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue)

How To Change Priority?

Rule 3

When a job enters the system, it is placed at the highest priority (the topmost queue).

Rule 4a

If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue)

Rule 4b

If a job gives up the CPU before the time slice is up, it stays at the same priority level

Attempt #1 — How To Change Priority?

Rule 3: When a job enters the system, it is placed at the highest priority (the topmost queue).

Rule 4a: If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue)

Rule 4b: If a job gives up the CPU before the time slice is up, it stays at the same priority level

How To Change Priority?

Rule 3

When a job enters the system, it is placed at the highest priority (the topmost queue).

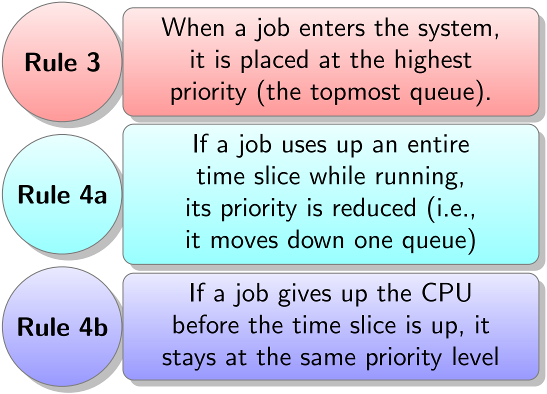
Rule 4a

If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue)

Rule 4b

If a job gives up the CPU before the time slice is up, it stays at the same priority level

How To Change Priority?



Rule 3: When a job enters the system, it is placed at the highest priority (the topmost queue).

Rule 4a: If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue)

Rule 4b: If a job gives up the CPU before the time slice is up, it stays at the same priority level

How To Change Priority?

Rule 3

When a job enters the system, it is placed at the highest priority (the topmost queue).

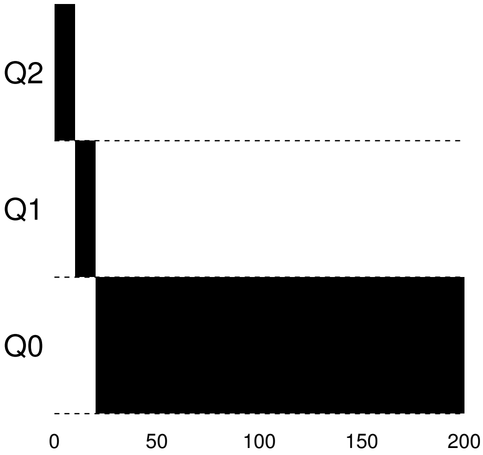
Rule 4a

If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue)

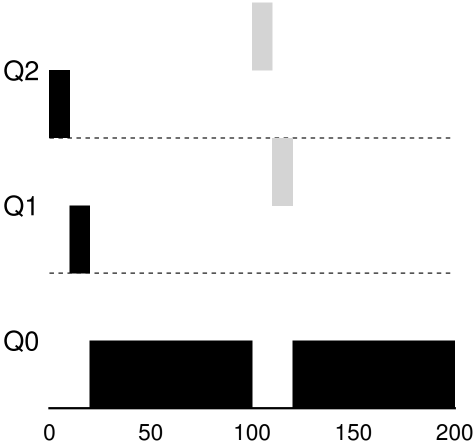
Rule 4b

If a job gives up the CPU before the time slice is up, it stays at the same priority level

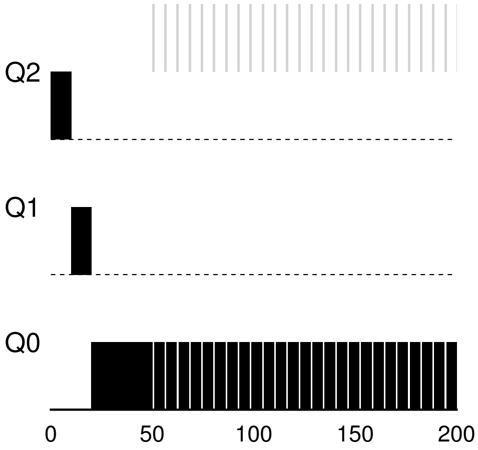
Example 1 Long-running Job Over Time



Example 2 Along Came An Interactive Job



Example 3 Mixed Workload



A Mixed I/O-intensive and CPU-intensive Workload

Problems With the Current MLFQ

Starvation

What if there are **too many** interactive jobs in the system?

Problem 2 Problems With the Current MLFQ

Gaming the scheduler

What if an user tricks the scheduler into giving you more than your fair share of the CPU?

Problems With the Current MLFQ

Change in behavior of a job

Is the current approach adaptive?

Attempt #2

The Priority Boost

Rule 5

After some time period S, move all the jobs in the system to the topmost queue

Priority Boost — Solves Two Problems

Solved

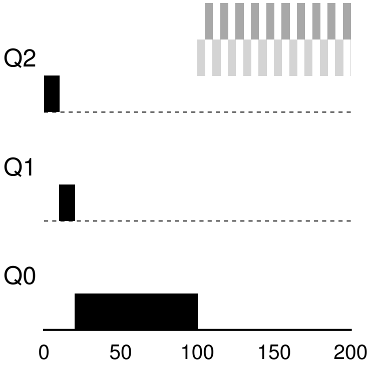
- Starvation
- Adapting to change in behavior

Why? — Still Unsolved

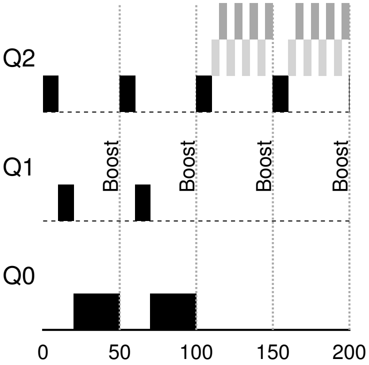
- Gaming the scheduler.

Example

- Value of $S = 50ms$



Without Boost



With Boost

How to decide the value of S ?

The Voo-Doo Constant (coined by John Ousterhout)

- Too high, and long-running jobs could starve
- Too low, and interactive jobs may not get a proper share of the CPU.

Attempt #3

Better Accounting

~~Rule 4a~~

~~If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue)~~

~~Rule 4b~~

~~If a job gives up the CPU before the time slice is up, it stays at the same priority level~~

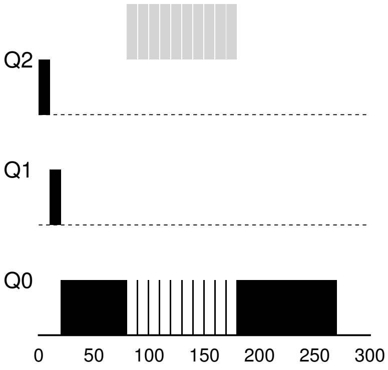
Rule 4

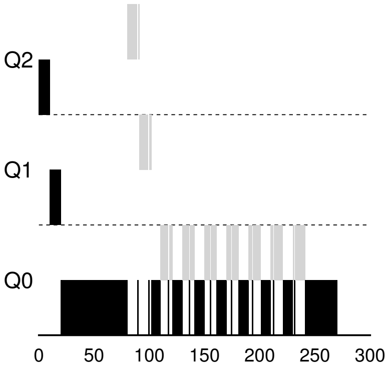
Once a job uses up its time allotment at a given level, its priority is reduced

Solves Gaming Problem

- Perform better accounting of CPU time at each level of the MLFQ
- Keep track of actual time spent on the CPU
- Demote once allotted time is over
- Regardless of how many times the job has given up the CPU

Gaming Tolerance (GT)





Without GT

With GT

Parameterizing MLFQ

- How many queues should there be?
- How big should the time slice be per queue?
- How often should priority be boosted in order to avoid starvation and account for changes in behavior?

No easy answers to these questions

Varying Time-slice Length

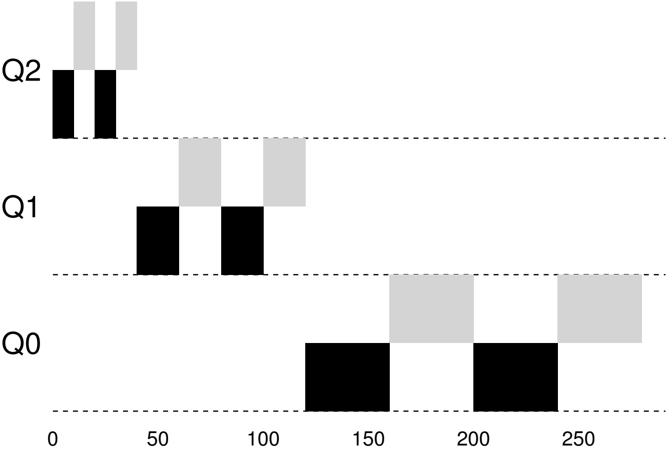
Varying time-slice length across different queues

- The high-priority queues are usually given short time slices; they are comprised of interactive jobs,
- The low-priority queues, in contrast, contain long-running jobs that are CPU-bound; hence, longer time slices work well

Other features

advice on setting priorities e.g. `nice` command on UNIX

Example Varying Time-slice Length



Rule 1

If $Priority(A) > Priority(B)$, A runs (B doesn't).

Rule 2

If $Priority(A) = Priority(B)$, A & B run in RR

Rule 3

When a job enters the system, it is placed at the highest priority

Rule 4

Once a job uses up its time allotment at a given level, its priority is reduced

Rule 5

After some time period S, move all the jobs in the system to the topmost queue

Idea

Lottery Scheduling

Approach:

- Processes are alloted lottery tickets
- Whoever wins runs
- Higher priority implies more tickets