Operating System (OS)
CS232

Scheduling Algorithm: Multi-Level Feedback Queue (MLFQ)

Dr. Muhammad Mobeen Movania

Outlines

- Recap
- Scheduling Algorithm: MLFQ
- Basic rules of MLFQ
- How MLFQ changes priority?
- How MLFQ avoids starvation and gaming?
- MLFQ parameter tuning
- Summary

Recap

- Ideally, we want a scheduler that minimizes turnaround time (like SJF, STCF) and also minimizes response time (like RR)
- The Problem is
  - The scheduler has no prior knowledge about the process like how much time it will need to finish and its general behavior
- One solution is
  - observe the process's behavior over time and adjust scheduler properties dynamically
- This is the core concept on which MLFQ are based

Scheduling Algorithm: MLFQ

- MLFQ has a number of distinct **queues**, each assigned a different **priority level**.
- At any given time, a job that is ready to run is on a single queue.
- MLFQ uses **priorities** to decide which job should run at a given time: a job with higher priority (i.e., a job on a higher queue) is chosen to run.
- For jobs with same priority, round robin scheduling is used.

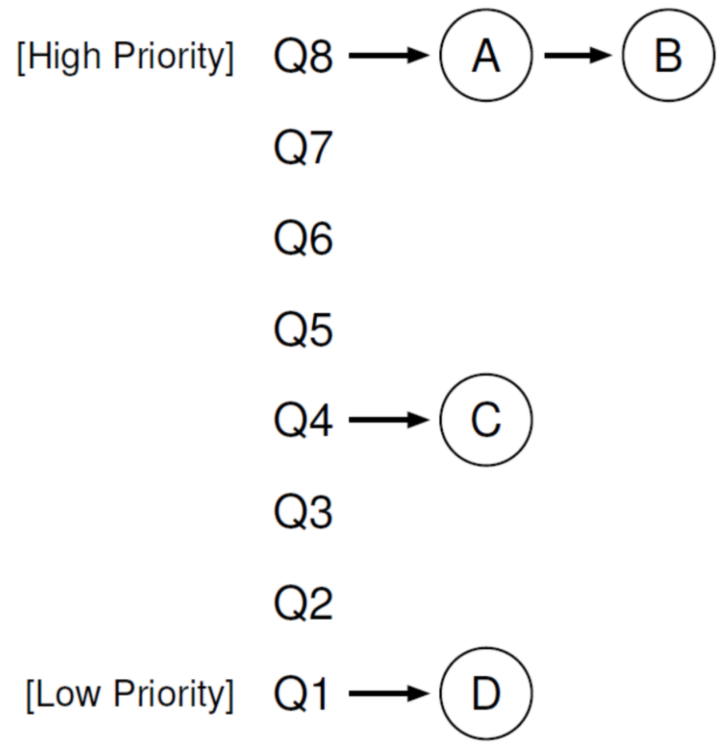

Figure 8.1: **MLFQ Example**

MLFQ

- Key Idea:
  - ***Vary priority*** of a job on its ***observed behavior***
- Priority of a process which releases CPU repeatedly (for e.g. due to I/O) is high.
- Priority of a process which uses CPU intensively for long duration is low.
- MLFQ tries to ***learn*** about processes as they run, and thus use the ***history*** of the job to predict its ***future behavior***

Basic Rules for MLFQ

- **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 (the topmost queue).
- **Rule 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
- **Rule 5:** After some time period S, move all the jobs in the system to the topmost queue. (priority boosting)

How does MLFQ change priority?

- When a job enters the system, it is given highest priority (Rule 3).
- When a job consumes its time share fully its priority is reduced no matter if its consumed fully in one go or in pieces (Rule 4).
- Example
  - Single long running job

Example 1 – Single Long-Running Job

- Time slice: 10 ms
- Job enter highest priority queue (Q2)
- If time slice expires, the job's priority is reduced and its shifted to the next lower level queue (Q1)
- If time slice expires again, the job's priority is reduced and its shifted to the lowest level queue (Q0)

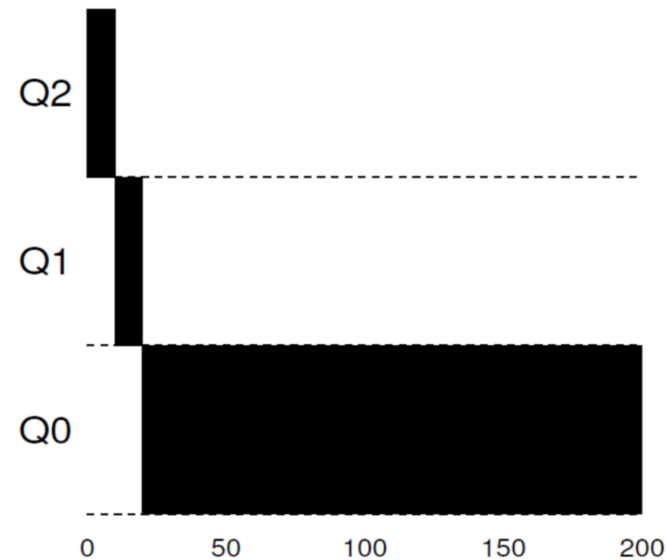

Figure 8.2: **Long-running Job Over Time**

Example 2 – A short job comes in

- Time slice: 10 ms
- Each process moves to next lower priority queue when it finishes its time slice
- Two jobs in system
  - A (CPU bound) (Black)
  - B (Interactive) (Gray)
- Process B arrives at T=100, runtime=20ms
- Process A moves to Q0 as its runtime is long whereas B finishes in 20ms so it is moved till Q1

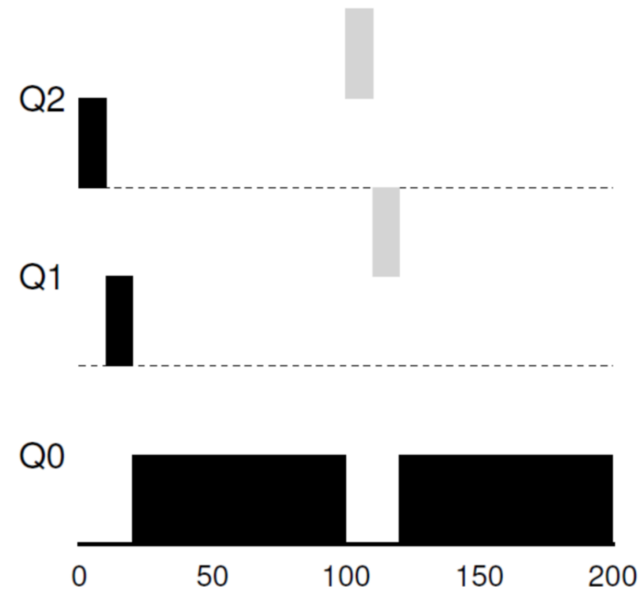

Figure 8.3: **Along Came An Interactive Job**

Key Takeaway

- MLFQ first assumes that the given process might be a short process giving it the highest priority.
  - If it is indeed a short process it finishes execution quickly
  - If it is a long process, it moves down to next lower queue giving it a lower priority
  - This way MLFQ approximates SJF

Example 3 – CPU+I/O intensive process

- Time slice: 10 ms
- Two jobs in system
  - A (CPU bound) (Black)
  - B (Interactive) (Gray)

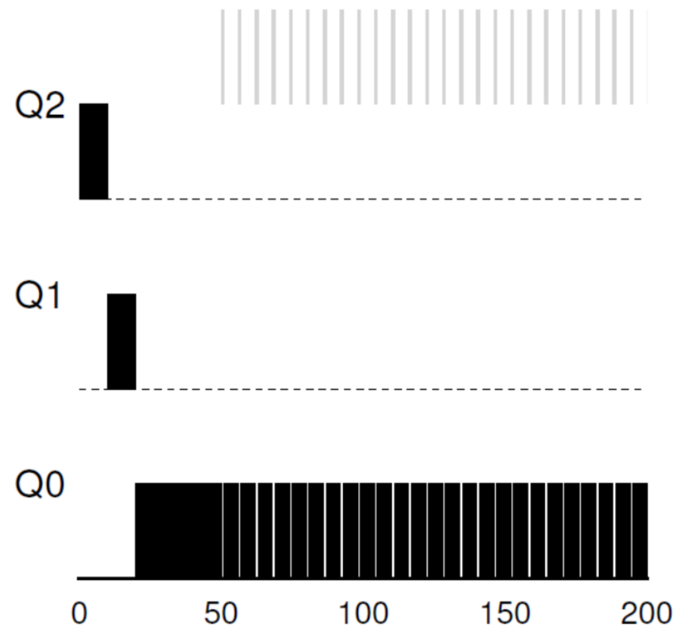

Figure 8.4: **A Mixed I/O-intensive and CPU-intensive Workload**

- Process B takes CPU for 1 ms so it remains on a higher priority queue (Q2)

Issues with running many interactive jobs

- CPU bound processes are starved as they are never scheduled
- Scheduler can be gamed by a process that issues an I/O request relinquishing the CPU thus keeping the process at the highest priority
- Solution
  - Instead of allowing process's time share to reset upon an I/O request, the total time share taken by a process with and without I/O should be considered (Rule 4)

Avoiding starvation

- Periodically boost priority of all the jobs
  - Prevents CPU bound processes to be starved as they are guaranteed CPU share after a given time period (S) when they are moved to the topmost queue.

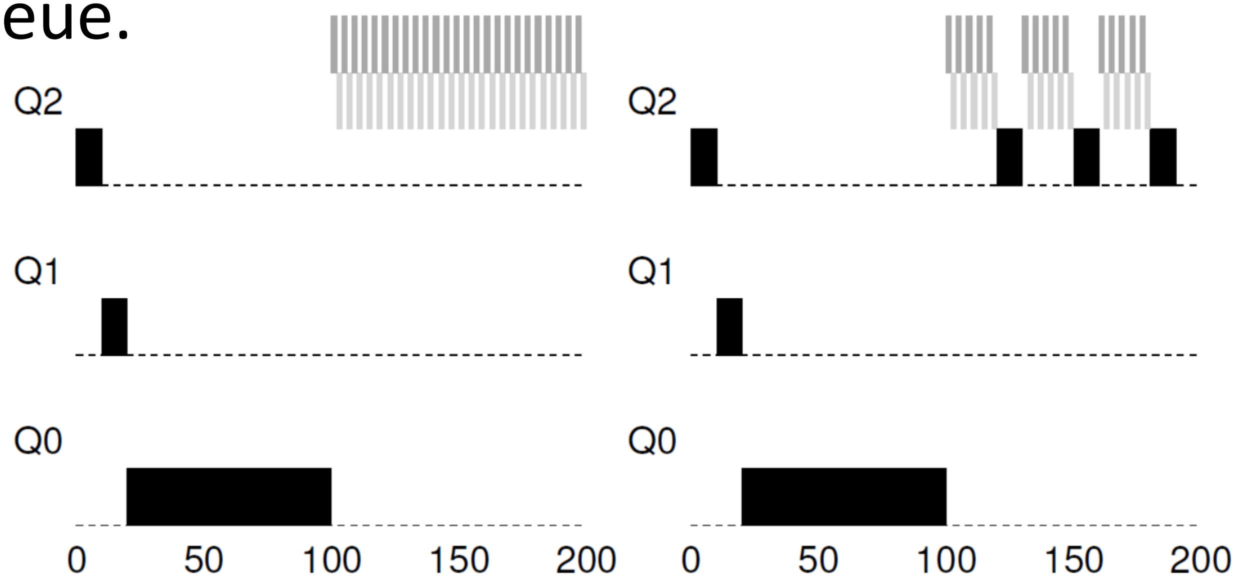

Figure 8.5: **Without (Left) and With (Right) Priority Boost**

Avoid gaming due to I/O

- Use better accounting that is keep note of the time share a process has used irrespective of whether it finished its time slice as one long burst or many small bursts.

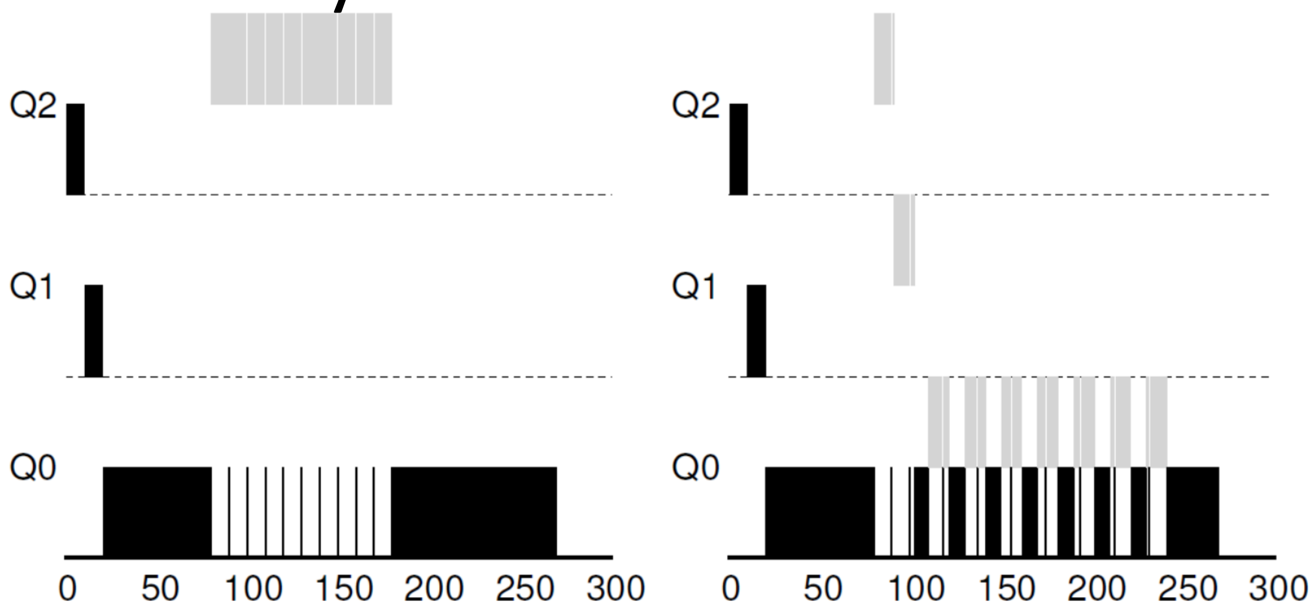

Figure 8.6: **Without (Left) and With (Right) Gaming Tolerance**

MLFQ Parameter Tuning

- Few key parameters
  - How many queues should there be?
  - How big should the time slice be per queue?
  - How often should priority be boosted?
- No easy answer
  - Usually OS's provide config files with default values that OS admins can modify as per need instead of using **Voo-Doo Constants**
  - Solaris MLFQ provides a table that admins can change
    - 60 queues
    - Time slices: 20ms – x00ms
    - Priorities boosted every 1 sec
  - FreeBSD scheduler calculates priority as a function of CPU usage
    - CPU usage is decayed over time resulting in giving less priority

Summary

- We discussed MLFQ
  - Multiple levels of queues
  - Uses feedback to determine priority depending on the process behavior without prior knowledge of job's running time
  - Uses five rules when assigning priorities to and scheduling processes
  - Provides the best of both worlds by
    - Minimizing turnaround times like (SJF, STCF)
    - Minimizing response time like (RR)