Scheduling:
The Multi-Level Feedback Queue

Prof. Yongtae Kim

Computer Science and Engineering
Kyungpook National University

Multi-Level Feedback Queue (MLFQ): Basic Rules

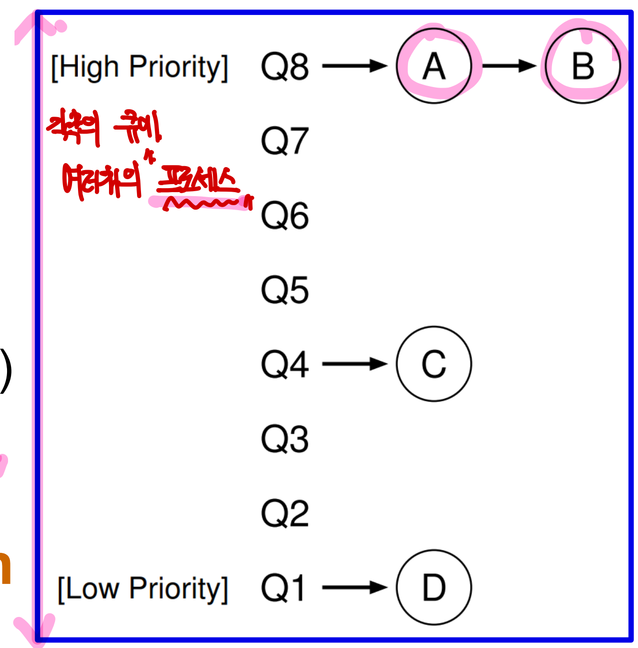
- **The fundamental problem Multi-Level Feedback Queue (MLFQ) tried to address is to:**

  프로세스가 언제 끝날지 모르는데.
  이걸 최적화 하고 싶다.

  1) optimize turnaround time by running shorter jobs first (don't know how long☹)
  2) minimize response time to make the systems responsive to interactive users

- **MLFQ has a number of distinct queues for each priority level**
  - A job that is ready to run is on a single queue
  - A job on a higher queue is chosen to run
  - RR is used among jobs in the same queue

  time slice 사용



- **Two basic rules for MLFQ are introduced**
  - **Rule #1**: If Priority(A) > Priority(B), A runs (B doesn't)
  - **Rule #2**: If Priority(A) = Priority(B), A & B run in RR

- **MLFQ varies the priority of a job based on its observed behavior**

Attempt #1: How to Change Priority

- **Our first attempt at a priority-adjustment algorithm is:**
  - **Rule #3**: When a job enters, it is placed at the highest priority (topmost queue)
  - **Rule #4a**: If a job uses up an entire time slice while running, its priority is reduced (moving 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

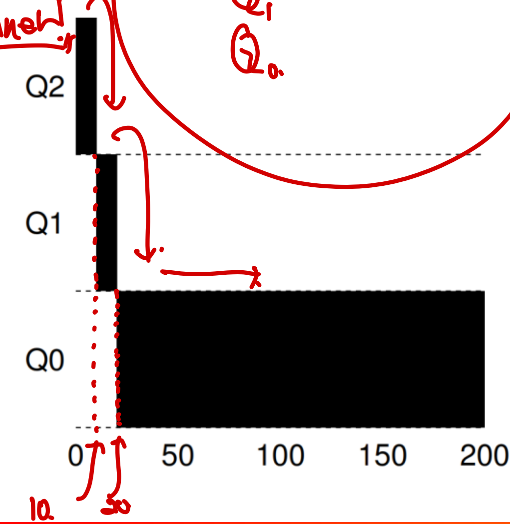
IO 처리등을 하러감.

- **In this manner, MLFQ can approximate SJF scheduling**

"추정 실행 시간이" 더 짧은 작업은 더 높은 우선순위

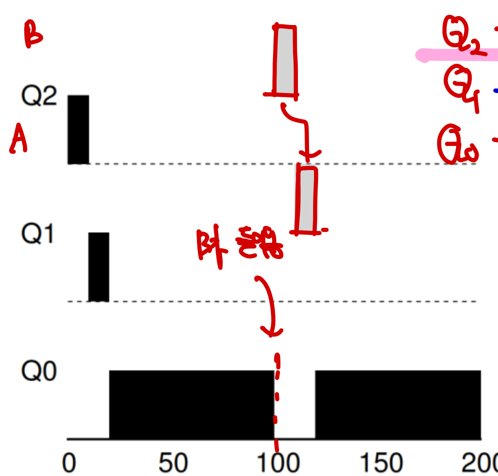
- **Example 1: A Single Long-Running Job**
  - Three-queue (Q0, Q1, Q2) scheduler
  - A job enters at the highest priority (Q2)
  - After a single time slice of 10, the scheduler reduces the job's priority by one (Q1)
  - After running at Q1 for a time slice, the job finally reduced to the lowest priority in the system (Q0), where it remains

" same manner "



MLFQ Examples

- **Example 2: Along Came a Short Job (Left figure)**
  - Two jobs: a long-running CPU-intensive job (A), a short-running interactive job (B)
  - A (black) has been running for some time and moves down to the lowest queue (Q0)
  - B (gray) arrives at time T=100 (run-time: 20), and is inserted into the highest (Q2)
  - B completes before reaching the bottom in two slices; then A resumes running at Q0

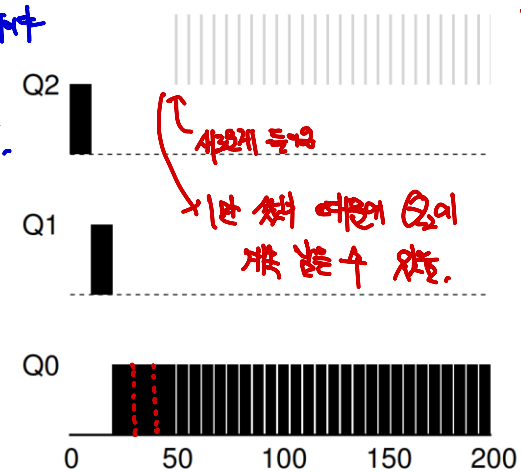


Example 2

$Q_2 \rightarrow$ B → 먼저 수행 되어야 함.

$Q_1 \rightarrow$ B → 분리함.

$Q_0 \rightarrow$ A



Example 3

1초 마다 10

$Q_2 \rightarrow$ B

Q_1

$Q_0 \rightarrow$ A

- **Example 3: What about I/O? (Right figure)**
  - An interactive job B that needs the CPU only for 1 before performing an I/O
  - MLFQ keeps B at the highest because B keeps releasing the CPU (**Rule #4b**)

Problems with Our Current MLFQ

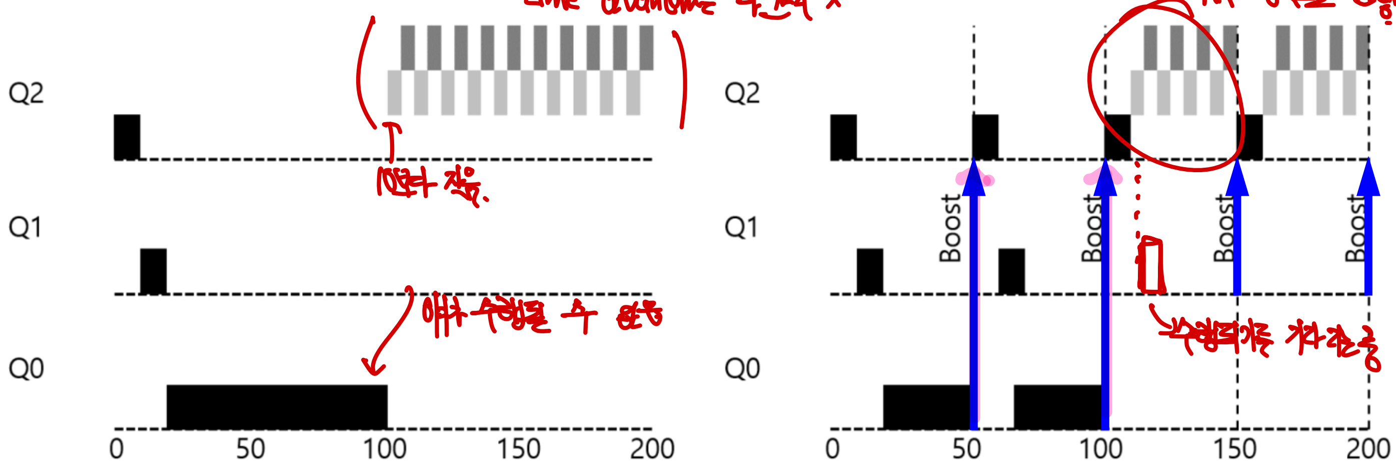
- **MLFQ in attempt #1 contains serious flaws**
  - The basic MLFQ seems to do a fairly good job, however, there are some problems such as starvation and gaming scheduler → 새로운게 계속 오면
    우선순위↓ starving
- **First, there is the problem of starvation**
  - If there are "too many" interactive jobs, long-running jobs will never receive any CPU time and they starve
    9.9 ms → IO
    (높은 우선순위 유지 = CPU 선점 가능)
- **Second, a smart user can rewrite program to game the scheduler**
  - What if a job relinquishes the CPU by an I/O after running 99% of a time slice?
  - This job gains a higher percentage of CPU time as it remains the same queue
- **Finally, a program may change its behavior over time**
  - What if a job is CPU-intensive at first but is changed to interactive one later?
  - Such a job would be out of luck and not be treated like other interactive ones
- **Let's try the attempt #2**

Attempt #2: The Priority Boost

- **The simple idea to avoid starvation problem is to periodically boost the priority of all the jobs**
  - **Rule #5**: after some time period *S*, move all the jobs to the topmost queue

우선순위를 period가 지나면 "바꿔주자."

- **Let's see an example**
  - A long-running job (A) with two short-running interactive jobs
  - The long-running job gets starved once the two short jobs arrives (see left)
  - There is a "priority boost every 50" and thus the long job can be run by getting boosted to the highest queue every 50 and getting to run periodically (see right)



RR 방식으로 진행

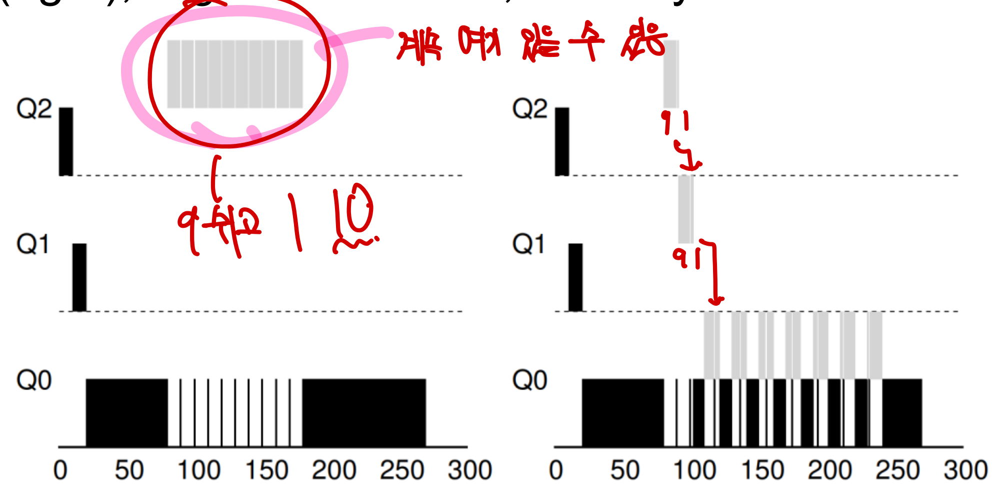
starvation 문제가 해결될 수 있다.

but: game scheduling 문제가 해결되지 않았음.

Attempt #3: Better Accounting

- **How to prevent gaming of our MLFQ scheduler?**
  - The solution is to perform better accounting of CPU time at each level by tracking how much of a time slice a process used at a given level
  - **Rule #4** (rewriting #4a/b): 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

- **Let's look an example**
  - In the old rules (left), a process can issue an I/O just before a time slice ends and thus dominate CPU time
  - In new rule (right), regardless of I/O, it slowly moves down and fairly share CPU



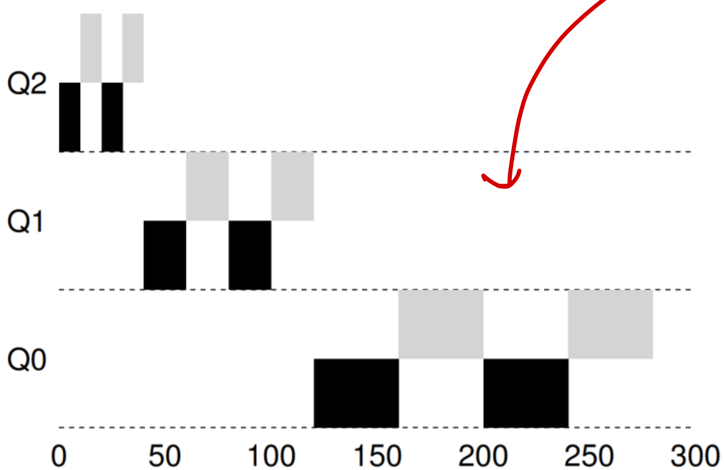
Tuning MLFQ and Other Issues

Queue [illegible]

- **A few other issues arise with MLFQ, particularly, parameters**
  - How many queues should there be?
  - How big should the time slice be per queue?
  - How often should priority be boosted to avoid starvation?

No easy answer (some experience)

- **Most MLFQ variants allow for varying time slice length for queues**
  - The high-priority queues are usually given short time slices (quickly alternating between interactive jobs) while the low-priority ones has longer time slices
  - Solaris MLFQ uses a table for easy configuration (60 queues, boost every 1s)



Lower Priority, Longer Quanta

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Solaris MLFQ Table