Basic Scheduling Policies

- First In, First Out (FIFO) / First Come, First Served (FCFS)
  - job queue – processes run in the order they arrive
  - non-preemptive – current process runs until it blocks or terminates
  - suffers from the convoy effect
- Shortest Job First (SJF)
  - job priority queue – processes run in order of increasing CPU burst time
  - non-preemptive – current process runs until it blocks or terminates
  - short jobs arriving at the same time as long jobs go first, but short jobs arriving after long jobs still have to wait

Basic Scheduling Policies, Continued

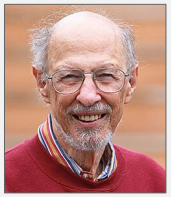
- Shortest Time to Completion First (STCF) / Shortest Time Remaining First (STRF) / Preemptive Shortest Job First (PSJF)
  - job priority queue – processes run in order of increasing CPU burst time
  - the current process can be preempted when a new job arrives
    - the new job will run instead if its burst time is less than the remaining time for the current job
  - jobs must still wait for those ahead of them to run to completion
- Round Robin (RR) / time-slicing
  - job queue – jobs are run in order of arrival
  - each process runs for a fixed amount of time
  - processes are preempted when their time slice is up
  - good for response time and fairness but turnaround suffers

Tradeoffs and Difficulties

- in general, a fair policy fares poorly with respect to performance
- optimizing response time (or turnaround) tends to come at the expense of the other
- and a major problem: SJF/STCF require knowing how long jobs will take

Multi-level Feedback Queue (MLFQ)

- the challenge: how to optimize both turnaround time (like SJF) and response time (like RR) without knowing the job size ahead of time?
- MLFQ is due to Fernando José Corbató (1926-2019)
  - American computer scientist
  - known for pioneering work in time-sharing operating systems
    - MIT Compatible Time-Sharing System (CTSS)
    - Multics, an inspiration for Unix
    - passwords
  - recipient of the 1990 Turing Award



https://en.wikipedia.org/wiki/Fernando_J._Corbat%C3%B3

Multi-level Feedback Queue (MLFQ)

- scheduling policy
  - assign a priority to each job
  - maintain multiple job queues, one for each priority level
  - schedule higher priority jobs first, using RR scheduling for jobs at the same priority level
- priorities are adjusted over time
  - new jobs are assigned the highest priority
  - a job that uses up its time slice at a given priority level is moved to a lower priority
  - periodically reset all processes to the highest priority level

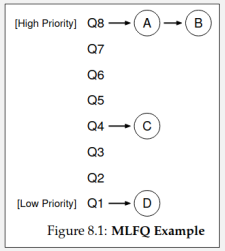


Figure 8.1: **MLFQ Example**

Adjusting Priorities

- new jobs are assigned the highest priority
  - assume the process is a short I/O bound process – response time is important

Adjusting Priorities

- a job that uses up its time slice at a given priority level is moved to a lower priority
  - gives short I/O-bound processes priority over long CPU-bound processes
  - adapts to the process' current behavior and doesn't require knowing the CPU burst time in advance

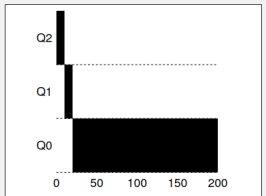


Figure 8.2: **Long-running Job Over Time**

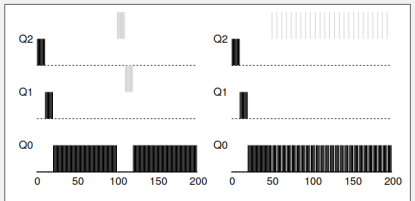


Figure 8.3: **Along Came An Interactive Job: Two Examples**

Adjusting Priorities

- periodically reset all processes to the highest priority level
  - avoids starvation of low priority or CPU-bound processes
  - allows processes that are no longer CPU bound to be handled more responsively

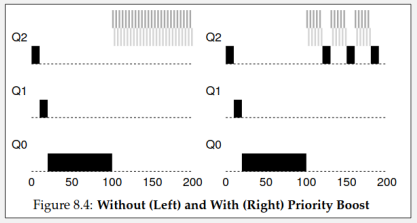


Figure 8.4: **Without (Left) and With (Right) Priority Boost**

Anti-Gaming

- what happens if a job always yields the CPU just before the end of its time slice?
  - if the priority only decreases when a job uses its full time slice, this job will remain at the highest priority
  - instead of only considering the last burst, keep track across bursts and reduce the priority only when the process has used its full allotment at that level

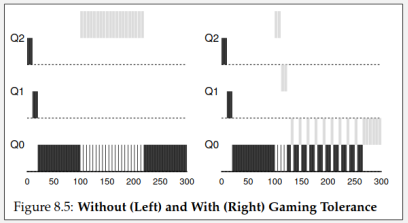


Figure 8.5: **Without (Left) and With (Right) Gaming Tolerance**

MLFQ Summary

- **Rule 1:** If Priority(A) $>$ Priority(B), A runs (B doesn't).
- **Rule 2:** 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 (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.

Tuning

- adjusting parameters
  - how many priority levels? how long is a time slice? how often are jobs boosted? etc
  - OS comes with default parameters, but they must be tuned based on workload

Other Options

- can vary the time slice by priority
  - e.g. longer time slice for lower priority, long-running jobs

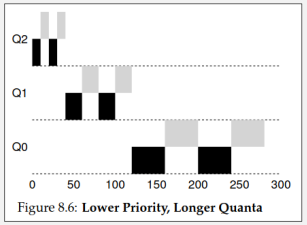


Figure 8.6: **Lower Priority, Longer Quanta**

- can reserve highest priority levels for OS jobs
- can allow user advice to set priorities
  - e.g. `nice`

Benefits

MLFQ –

- prioritizes short, I/O bound processes over long, CPU bound ones
  - how: I/O bound processes will stay at the highest priority level because they don't use up their time slices
  - why: turnaround is better when short processes don't have to wait for long ones
- adjusts to actual process behavior
- is practical
  - does not require unrealistic assumptions like knowing CPU burst time