Operating Systems - Chapter 8 Summary

Scheduling: The Multi-Level Feedback Queue (MLFQ)

• MLFQ is one of the most influential scheduling algorithms, introduced in CTSS (1962).
• Goal: Balance turnaround time (like SJF/STCF) and response time (like RR) without knowing job lengths.
• Key idea: Learn from the history of jobs to predict future behavior.

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 round-robin.
• Rule 3: New jobs start at the highest priority queue.
• Rule 4: Once a job uses its time allotment at a level, it moves down one queue.
• Rule 5: After some time S, all jobs are boosted to the highest queue.

Key Scheduling Philosophy

• One major goal of MLFQ: since the scheduler doesn't know in advance if a job is short or long, it assumes it may be short and gives it high priority. – If the job is short, it finishes quickly with low turnaround time. – If the job is long, it gradually moves down to lower-priority queues, proving itself to be batch-like. • This way, MLFQ approximates SJF/STCF while still remaining fair.

Interactive Jobs and I/O

• Jobs that give up CPU early (e.g., waiting for I/O) stay at high priority.
• This ensures interactive jobs get quick response times.

Problems and Fixes

• Starvation: too many interactive jobs can prevent CPU-bound jobs from running.
– Solution: Rule 5 (periodic priority boost).
• Gaming: jobs could exploit I/O to retain high priority unfairly.
– Solution: Better accounting of CPU time; demotion occurs once allotment is used, regardless of I/O.
• Changing behavior: jobs may switch between CPU-bound and interactive phases.
– Solution: priority boosts help reclassify them correctly.

Tuning and Variants

• Parameters: number of queues, time slice per queue, allotments, and boost interval.
• High-priority queues: short time slices for interactive jobs.
• Low-priority queues: long slices for CPU-bound jobs.
• Implementations (e.g., Solaris, BSD, Windows NT) refine these rules with tables or formulas.

Summary

MLFQ uses multiple queues with dynamic priorities to balance responsiveness and throughput. It assumes jobs may be short, giving them priority, but demotes long jobs gradually. This adaptive strategy lets MLFQ approximate SJF for short jobs while ensuring fairness for long ones. Periodic boosts and better accounting prevent starvation and gaming. MLFQ remains a widely used and flexible scheduling algorithm in modern systems.