Scheduling Processes

Yannis Smaragdakis, U. Athens
Many Objectives

• Throughput
  – finishing many jobs (i.e., processes)

• Latency
  – finishing a job quickly

• Responsiveness
  – minimizing pauses, or first reaction

• Fairness
  – ensuring all jobs get a chance to run
Metrics

- Turnaround time
  \[ T_{\text{turnaround}} = T_{\text{completion}} - T_{\text{arrival}} \]
- Response time
  \[ T_{\text{response}} = T_{\text{firstrun}} - T_{\text{arrival}} \]
- CPU Utilization
Difficulties

- Jobs arrive at any time
- Their workload is unknown
- They don't just use the CPU, but may block
  - relinquishing the CPU! (Most common!)
- Their blocking duration is unpredictable
- Switching jobs has an overhead
Some Policies

- FIFO/First-Come, First-Served (FCFS)
- Shortest Job First (SJF)
  - do we know enough to do this well?
- Shortest Time-to-Completion First (STCF)
- Round Robin (RR) with time slices
  - tradeoff? I/O?
  - generally: fair policy => bad turnaround time
Need Adaptive Policy!

• First try:
  - *use priorities (many queues), RR if same*
  - *initially at high priority*
  - *a job that exhausts time slice gets reduced priority*
    - otherwise, if it blocks, stays at same priority

• Aim: I/O jobs get preference

• Weaknesses: can be gamed, can starve
Good Adaptive Policy: MLFQ

• Multi-Level Feedback Queue:
  - *use priorities (many queues), RR if same*
  - *initially at high priority*
  - *a job that exhausts time slice (in however many schedulings) gets reduced priority*
  - *after some time S, move all jobs to top priority*
Other Approaches

• Lottery scheduling: distribute tickets (e.g., process A gets 30%, B gets 60%, C gets 10%), hold lottery on every time slice

• Randomized algorithm: good for avoiding worst-case behavior