CPU Scheduling
Scheduling
Question

• What is the job of a CPU scheduler?

• What does preemption mean?

• Name a few scheduling algorithms.
Definitions

• Processes are also called jobs.

• Scheduling policy
  – A decision made by CPU as to which process to run next
Scheduling Metric

- Scheduling policies are compared with a metric – A way to measure something

- Turnaround time

\[ T_{\text{turnaround time}} = T_{\text{completion}} - T_{\text{arrival}} \]

\[ T_{\text{arrival}} = 0 \text{ (all jobs arrive at the same time)} \]
First In, First Out

Three jobs: A, B, C

• Assume each job runs for 10 seconds
  \[ \text{Average } T_{\text{turnaroundtime}} = \frac{(10 + 20 + 30)}{3} = 20 \]

• A Job runs for a different amount of time
  \[ \text{Average } T_{\text{turnaroundtime}} = \frac{(100 + 110 + 120)}{3} = 110 \]
Shortest Job First

The shortest job runs first.
Average $T_{\text{turnaround time}} = (10 + 20 + 120) / 3 = 50$

Jobs do not arrive at the same time
Average $T_{\text{turnaround time}} = (100 + (110 - 10) + (120 - 10)) / 3 = 103.33$
Shortest Time-to-Completion First

• When a new job arrives, the remaining time for all jobs is computed.
• Job A is preempted to run jobs B and C.

Average $T_{\text{turnaround time}} = \frac{(120 - 0) + (20 - 10) + (30 - 10)}{3} = 50$
Response Time

\[ T_{\text{response}} = T_{\text{firstrun}} - T_{\text{arrival}} \]

• For the example shown in shortest time-to-completion first:
  Average \( T_{\text{response}} = (0 + 0 + 10) = 3.33 \)

• Response time needs to be short for interactive systems.
Round Robin (RR)

- RR runs jobs using a time slice.
  - Interrupts jobs every *time slice* units of time
- Too short time slice has high overhead (context switch)
- RR does not do well using the *turnaround* time.
  - Delays finishing jobs
- RR is fair (divides CPU time fairly among processes)

Average $T_{\text{response}} = \frac{0 + 5 + 10}{3} = 5$ sec

Average $T_{\text{response}} = \frac{0 + 1 + 2}{3} = 1$ sec

Average $T_{\text{turnaround}} = \frac{13 + 14 + 15}{3} = 14$ sec
Incorporating I/O

- When a job makes an I/O request, it is blocked.
- CPU can run another job.
- After completing I/O, CPU resumes the job.
- Example
  - Jobs A and B run for 50 ms.
  - Job A runs for 10 ms and then makes an I/O request.

CPU time is used poorly  Better use of CPU time through overlap
Process States

• Running: CPU executes process instructions.

• Ready: ready to run, but scheduler decides not to run it at a given time.

• Blocked: a process performs some operation (e.g., I/O) that makes it not ready to run. Other processes use CPU.
Process States

- Running
- Ready
- Blocked

Transitions:
- Scheduled
- Descheduled
- I/O: initiate
- I/O: done
## Process States

<table>
<thead>
<tr>
<th>Time</th>
<th>Process₀</th>
<th>Process₁</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Running</td>
<td>Ready</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Running</td>
<td>Ready</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Running</td>
<td>Ready</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Blocked</td>
<td>Running</td>
<td>Process₀ initiates I/O</td>
</tr>
<tr>
<td>5</td>
<td>Blocked</td>
<td>Running</td>
<td>Process₀ is blocked, so Process₁ runs</td>
</tr>
<tr>
<td>6</td>
<td>Blocked</td>
<td>Running</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Ready</td>
<td>Running</td>
<td>I/O done</td>
</tr>
<tr>
<td>8</td>
<td>Ready</td>
<td>Running</td>
<td>Process₁ now done</td>
</tr>
<tr>
<td>9</td>
<td>Running</td>
<td>−</td>
<td>Process₀ now done</td>
</tr>
<tr>
<td>10</td>
<td>Running</td>
<td>−</td>
<td></td>
</tr>
</tbody>
</table>
Question

• For the workload shown here, show the schedule timelines for
  – FIFO
  – SJF (non-preemptive)
  – STCF (preemptive)
  – RR (time slice = 1 second)

<table>
<thead>
<tr>
<th>Job</th>
<th>Arrival time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
Multi-Level Feedback Queue (MLFQ)
Scheduling: Proportional Share
Agenda

• Multi-Level Feedback Queue
• Proportional share scheduler
• CPU scheduling exercise
• Directory compression demo
• Lab
Question

• How should a scheduler run a workload that consists of CPU-intensive and interactive jobs?

• What does the proportional share scheduler guarantee?
MLFQ

- Minimize response time for interactive jobs and minimize turnaround time

- Scheduler does not know the execution length of jobs.
MLFQ Basics

• Consists of multiple distinct queues
• Each queue has a priority level.
• A job is in one of queues.
• Two rules for scheduling jobs A and B
  ▪ Rule 1: If Priority(A) > Priority(B), A runs
  ▪ Rule 2: If Priority(A) = Priority(B), A and B run with round robin
MLFQ Basics

• Predict future behavior of a job based on its history

• Reduce the priority of a job that intensively uses CPU

• Increases the priority of a job that repeatedly performs I/O
MLFQ Basics

[High Priority]  Q8 → A → B
Q7
Q6
Q5
Q4 → C
Q3
Q2

[Low Priority]  Q1 → D

MLFQ changes priority of jobs over their lifetimes.
How to Change Priority

• Three rules

  – Rule 3: If a job enters a system, it is placed in a queue with the highest priority.

  – Rule 4a: If a job uses the entire time slice, its priority is reduced by one.

  – Rule 4b: If a job does not use the entire time slice, its priority is not changed.
Long-running Job

CPU-bound job
Time slice is 10 ms
A Long Came A Short Job

Approximate shortest-job first
Assume that B is a short-running job
What About I/O

Job B stays in Q2 because it releases CPU sooner than time slice.
MLFQ Problem

- Starvation
  - CPU-bounds jobs may not receive CPU time.

- Game the scheduler
  - Trick scheduler to stay in the highest priority queue

- An CPU-bound jobs becomes an interactive job.
• **Rule 5:** After a time period, move all jobs to the highest-priority queue.
• CPU-bounds jobs are not starved.
• If a CPU-bound job become interactive, it receives CPU time.
Better Accounting

Keep track of how much of time slice is used

**Rule 4:** After a job uses up time slice, it moves to low-priority queue.
Varying Time Slice

• Vary time slice per queue

• High-priority queues have short time slices.
  – Scheduler alternates among jobs quickly.

• Low-priority queues have long time slices.
Varying Time Slice

10-ms time slice

20-ms time slice

40-ms time slice
Scheduling: Proportional Share
Proportional Share Scheduler

• Try to guarantee that each job receives a certain percentage of CPU time

• Lottery scheduling
  – Time to time, hold a lottery to select the next process for scheduling
  – A process that should run more often has more chances to win.
Lottery Scheduling

• Lottery scheduling
  – **Ticket** is a share of CPU that a process receives.
  – **Percent tickets** represent the percentage of CPU time that the process receives.

• Example
  – Processes A and B
  – 100 tickets
  – Assume **A** holds tickets 0 to 74 and **B** holds the remaining tickets
Lottery Scheduling

Winning tickets 63 85 70 39 76 17 29 41 36 39 10 99 68 83 63
Schedule A B A A A B A A A A A A B A B A

Job A gets 73% of time slices instead of the expected 75%.

Given a random number (winning ticket)
Until the winning process is found:
Add up a process’s number of tickets to a counter
If counter > ticket the winner is found
else go to the next process
Stride Scheduling

• Deterministic scheduling

• Stride: inversely proportional to number of tickets
  – Divide process’s tickets by a very large number
  – \( \text{stride}_A = 100, \text{stride}_B = 200, \text{stride}_C = 40 \)

• Pass: a counter that is incremented by the stride after a process is scheduled.
Stride Scheduling

Select a process with a minimum pass
Increment the process’s pass by its stride

At the end of a cycle, each process runs exactly in proportion to its number of tickets.