CS 202: SCHEDULING

- Last class: Finished concurrency
- Next few classes
  - Today: Scheduling
  - Thursday: Dive into a bug from the past
    - Want to derive lessons from system design. PLEASE READ BEFORE CLASS!
- Next Tuesday: NO CLASS (MONDAY SCHEDULE)

Some comments on feedback
- Being lost, or unclear
  - Please ask everyone to say none of this makes sense.
    - Happy to revisit again.
- Labs not covered in lectures
  - stat (2)

- How this all fits together.

Back to scheduled programming
We talked about threads & processes

- Proc1
  - A
  - B
- Proc2
  - C
- Proc3
  - D

Scheduler OS

- Core 0
- Core 1

Core Idea: ONLY SOME THREADS Executing AT A Time.

Two Questions:

- Mechanism: How to switch between threads
  CONTEXT SWITCHES Nov.

Things To Know

- Costly: Takes cycles, affects caches

Policy: What to run NEXT

Today
Preemptive Scheduling:

Preemption

Yield: Allow invoke scheduler.

Preemption: Timers interrupt.

Make sure time/length of process does not hold onto processor (come) too long.

Hold onto process (come) too long.

PCB → TCB

Process/Thread State

Completed → Ready

Requeued

Scheduling

Scheduler

Yeld

Blocked

Running
Scheduling Policy (Algorithm)

- Can do nearly anything to choose next thread/process
  (someone, somewhere probably tried)

- Going to look at a few today
  ... but need a way to compare them.

Metrics

- Turnaround time/completion time/response time
  "How quickly does a process appear to work?"

Interactive Tasks:
- Editors
- Browsers
- Terminals
- ...
Textbook: **Output Time**
*Many texts call this response time

**Batch Processing**
- Video/Audio encoding
- Scientific computation
- Simulation
- Model training

Textbook: **Turnaround Time**

**System Throughput**

Number of processes that can finish/produce output every second
- Fairness
  - All processes/users get the same chance to use processor*
  
  - No process starves

  - High priority (important) threads/process get most of the time
Building Intuition: No I/O

FCFS (First come first served) Non-Preamtrv

Batch

1. \( p_1 \) (24 seconds)
2. \( p_2 \) (3 seconds)
3. \( p_3 \) (2 seconds)

Run until Process Finishes

Average Turnaround Time? \( \frac{80}{3} \approx 27 \)

Average Throughput?

What happens to Turnaround time & throughput if we change arrival order?
B) **SHORTEST JOB FIRST (SJF) OR SHORTEST TIME TO COMPLETION FIRST (STCF)**

**SHORTEST JOB FIRST (Non-Preemptive)**

1. P3 (2 seconds)
2. P2 (4 seconds)
3. P1 (24 seconds)

**SHORTEST TIME TO COMPLETION FIRST (Preemptive)**

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival</th>
<th>Time to Completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

C) **ROUND ROBIN**
- Preemptive. Quanta / Time Slice
  How Long A Process Gets To Run

- Go To Next Process

(Makes Tput & Turnaround Time A Bit Harder?)

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival</th>
<th>Time To Completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

Quanta = 1

Turnaround Time = \[12 + 12 + 18\]

Tput
Adding Back I/O

A: Only CPU, 1 week

B: Only CPU, 1 week

C: Run forever
   \( \rightarrow 1\text{ms CPU} \xrightarrow{\text{Block}} 10\text{ms disk} \)
   \( \uparrow \text{ Unblock} \)

- FIFO

- Round Robin: 100ms QUANTA

- Round Robin: 1ms QUANTA
STC F (using CPU time as completion time)

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1 week</td>
</tr>
<tr>
<td>B</td>
<td>1 week</td>
</tr>
<tr>
<td>C</td>
<td>1 ms</td>
</tr>
<tr>
<td></td>
<td>Disk</td>
</tr>
</tbody>
</table>

Observation: STC F is good, but
- Impractical: Don’t know compute time a-priori
Problem with Strict Priority
Multi-Level Feedback Queue

Feedback: Change priority based on how much CPU

Linux
- Fair Scheduling
- Lottery Scheduling

A: 10 tickets
B: 5 tickets
C: 15 tickets
Things To Take Away

- Process States

New → Runnable ← Running → Exit

↑

Blocked ←

- Scheduling is a common problem
  → must consider requirements
  when deciding algorithm

- Some of the simple scheduling policies we talked about are very common in practice:
  - Round-Robin
  - MLFQ

- Often, valuable to compare to
good but impractical algorithms
  (e.g., SCfE) is often useful.