The CPU Scheduling Problem

We have multiple processes/threads, but only one CPU
How much time does each process/thread get on CPU?

Possibilities
- Keep it till done
- Each uses it a bit and passes it on
- Each gets proportional to what they pay

Which is the best policy?

There is No Single Best Policy

Depends on the goals of the system
Different for
- your personal computer
- large time-shared computer
- computer controlling a nuclear power plant

Might even have multiple (conflicting) goals
Scheduling: An Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

What order minimizes average turnaround time?

Turnaround time: time between arrival and departure
- Arrive, wait for CPU (waiting time)
- Use CPU (service time), depart

Shortest First Is Provably Optimal

Given n processes with service times S₁, S₂, S₃, ..., Sₙ

Average Turnaround Time (ATT)
- \[ = \frac{(S₁ + (S₁ + S₂) + (S₁ + S₂ + S₃) + \ldots + (S₁ + \ldots + Sₙ))}{n} \]
- \[ = \frac{((n \times S₁) + ((n-1) \times S₂) + ((n-2) \times S₃) + \ldots + Sₙ)}{n} \]

S₁ has maximum weight (n), minimize it
S₂ has next-highest weight (n-1), minimize it after S₁
In general: order by shortest to longest

Consider Different Arrival Times

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Longest First vs. Shortest First
**First-Come First-Served**

Allocate CPU in the order that processes arrive

<table>
<thead>
<tr>
<th>Process</th>
<th>Departure Time</th>
<th>Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>P₂</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>P₃</td>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>

Average Turnaround Time = \( \frac{5 + 7 + 6}{3} = 6.0 \)

Simple, non-preemptive, poor for short processes

---

**Round Robin**

Time-slice CPU: give each processes a quantum in turn

<table>
<thead>
<tr>
<th>Process</th>
<th>Departure Time</th>
<th>Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>P₂</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>P₃</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Average Turnaround Time = \( \frac{9 + 6 + 1}{3} = 5.3 \)

Simple, preemptive, more overhead than FCFS

---

**Shortest Process Next**

Select process with shortest execution time

<table>
<thead>
<tr>
<th>Process</th>
<th>Departure Time</th>
<th>Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>P₂</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>P₃</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

Average Turnaround Time = \( \frac{5 + 8 + 3}{3} = 5.3 \)

Optimal for non-preemptive, must know exec times

---

**Shortest Remaining Time**

Select process with shortest remaining time

<table>
<thead>
<tr>
<th>Process</th>
<th>Departure Time</th>
<th>Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>P₂</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Average Turnaround Time = \( \frac{9 + 4 + 1}{3} = 4.7 \)

Optimal, must know execution times
Multi-Level Feedback Queues

Multiple ready queues 0, 1, ..., n
Always select process in lowest-numbered queue
Run selected process for $2^i$ quantums (for queue i)
If process doesn’t block, place in next higher queue (except last)

Example using Feedback Queues

Preemptive: if a new process arrives, current process goes back to queue it came from

Average Turnaround Time = (9 + 5 + 1)/3 = 5.0
Favors shorter processes over longer, dynamic

Priority Scheduling

Select process with highest priority
Example: $P_1$ = medium, $P_2$ = high, $P_3$ = low

Allows scheduling based on arbitrary criteria
• External, e.g., based on who’s willing to pay most
• Internal, e.g., past CPU usage (dynamic)

Fair Share (Proportional Share)

Processes get predetermined fraction of CPU time
Example: $P_1$ to get 25%, $P_2$ to get 50%, $P_3$ to get 25%
Each quantum, which process got least of its fair share?
What is fair share? Getting what you’re supposed to get
Computing Ratios for Fair Share

Determine utilization: fraction of CPU time received

Compute ratio: utilization to fraction promised
- If ratio < 1, process getting less than its FS
- If ratio > 1, process getting more than its FS
- If ratio = 1, process getting exactly its FS

Example
- Utilization: P1 40%, P2 40%, P3 15%
- Promised: P1 25%, P2 50%, P3 25%
- Ratio: P1 40/25 = 1.6, P2 40/50 = 0.8, P3 15/25 = 0.6

Process with lowest ratio needs CPU most: schedule it

Calculating Exponential Averages

Exponential average: \( A_{i+1} = \alpha M_i + (1 - \alpha) A_i \)
- \( A_i \): exponential average at time \( i \)
- \( M_i \): measurement at time \( i \)
- \( \alpha \): weight (0 - 1) on recent vs. past history
  - large \( \alpha \) emphasizes recent history
  - small \( \alpha \) emphasizes past history

Example for P1 utilization (\( \alpha = 0.25 \), assume \( A_0 = 0.5 \))
- \( M_i \): 1 0 0 0 1 0 1
- \( A_i \): 0.63 0.47 0.35 0.26 0.45 0.34 0.50
- True: 1 0.50 0.33 0.25 0.40 0.33 0.45

Real Time Scheduling

Correctness of real-time systems depend on
- logical result of computations
- the timing of these results

Type of real-time systems
- Hard vs. soft real-time
- Periodic vs. aperiodic

Scheduling
- Earliest deadline
- Rate monotonic scheduling
Periodic Processes (or Tasks)

Periodic processes: perform computation at certain rate

\[ C = \text{compute time}, \ T = \text{period}, \ U = \frac{C}{T} = \text{utilization} \]

Can processes be ordered so that deadlines are met?
- Should \( P_1 \) run before or after \( P_2 \)?

Example

<table>
<thead>
<tr>
<th>( P_1 )</th>
<th>( C )</th>
<th>( T )</th>
<th>( U )</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>30</td>
<td>50%</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>50%</td>
<td></td>
</tr>
</tbody>
</table>

Sum of utilizations does not exceed 100%

\( P_1 \) running before \( P_2 \) causes \( P_2 \) to miss all deadlines!

Earliest Deadline

Schedule the process that has the earliest deadline

Requires sorting of deadlines, \( O(n \log n) \) complexity

Also works for aperiodic processes
Earliest Deadline

Example: C T U
P1 30 60 50%
P2 10 40 25%
P3 7.5 30 25%
Meets all deadlines (sum of utilizations = 100%)

Rate Monotonic Scheduling

Prioritize based on rates (rate = 1/period) - no sorting!
Deadlines guaranteed met if:

\[ U_1 + U_2 + \ldots + U_n \leq n \left(2^{1/n} - 1\right) \]
Limited to periodic processes

Example: C T U Rate
P1 30 60 50% 1/60 = 0.017
P2 10 40 25% 1/40 = 0.025
P3 7.5 30 25% 1/30 = 0.033
Fails: P3 misses two deadlines!

Example: C T U Rate
P1 20 60 33% 1/60 = 0.017
P2 10 40 25% 1/40 = 0.025
P3 5 30 16.7% 1/30 = 0.033
Success: \( U_1 + U_2 + U_3 = 75\% \leq 3 \left(2^{1/3} - 1\right) = 78\% \)
## Rate Monotonic Scheduling

<p>| | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>T</td>
<td>U</td>
<td>Rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₁</td>
<td>25</td>
<td>60</td>
<td>41.7%</td>
<td>1/60 = 0.017</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td>10</td>
<td>40</td>
<td>25%</td>
<td>1/40 = 0.025</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td>5</td>
<td>30</td>
<td>16.7%</td>
<td>1/30 = 0.033</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Failure: $U₁ + U₂ + U₃ = 83.3% > 3 \left(\frac{2^{1/3} - 1}{3}\right) = 78\%$