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

Longest First vs. Shortest First

Longest First

Shortest First

Shortest First Is Provably Optimal

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

Average Turnaround Time (ATT)
\[
\text{ATT} = \frac{S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \ldots + (S_1 + \ldots + S_n)}{n}
\]
\[
\text{ATT} = \frac{ [(n \times S_1) + ((n-1) \times S_2) + ((n-2) \times S_3) + \ldots + S_n] }{n}
\]

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>

Waiting
Executing
**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>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>

Average Turnaround Time = (5 + 7 + 6)/3 = 6.0
Simple, non-preemptive, poor for short processes

**Round Robin**
Time-slice CPU: give each process a quantum in turn

<table>
<thead>
<tr>
<th>Process</th>
<th>Departure Time</th>
<th>Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Average Turnaround Time = (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>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

Average Turnaround Time = (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>1</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Average Turnaround Time = (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**

<table>
<thead>
<tr>
<th>Process</th>
<th>Utilization (%)</th>
<th>Promised (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>40</td>
<td>25</td>
</tr>
<tr>
<td>P2</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>P3</td>
<td>15</td>
<td>25</td>
</tr>
</tbody>
</table>

**Ratio**

- P1: 40/25 = 1.6
- P2: 50/50 = 1
- 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 \( A_0 = 0.5 \))**

<table>
<thead>
<tr>
<th>Time</th>
<th>Utilization</th>
<th>Promised</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100/25</td>
<td>0/50</td>
<td>P2 smallest, schedule P2</td>
</tr>
<tr>
<td>2</td>
<td>50/25</td>
<td>100/50</td>
<td>tie, schedule either</td>
</tr>
<tr>
<td>3</td>
<td>33/25</td>
<td>100/50</td>
<td>0/25 schedule P3</td>
</tr>
<tr>
<td>4</td>
<td>25/25</td>
<td>67/50</td>
<td>P3 done, schedule P1</td>
</tr>
<tr>
<td>5</td>
<td>40/25</td>
<td>50/50</td>
<td>schedule P2</td>
</tr>
</tbody>
</table>

---

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

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

---

**Example**  
\[
\begin{array}{c|c|c|c}
\text{Process} & \text{C} & \text{T} & \text{U} \\
\hline
P_1 & 15 & 30 & 50\% \\
P_2 & 10 & 20 & 50\% \\
\end{array}
\]

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:

<table>
<thead>
<tr>
<th>Process</th>
<th>Start Time</th>
<th>Completion Time</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>60</td>
<td>50%</td>
</tr>
<tr>
<td>P2</td>
<td>10</td>
<td>40</td>
<td>25%</td>
</tr>
<tr>
<td>P3</td>
<td>7.5</td>
<td>30</td>
<td>25%</td>
</tr>
</tbody>
</table>

Meets all deadlines (sum of utilizations = 100%)

---

### Rate Monotonic Scheduling

Prioritize based on rates (rate = 1/period) - no sorting!

Deadlines guaranteed met if:

\[ U_1 + U_2 + \cdots + U_n \leq n \left(\frac{2^1}{n} - 1\right) \]

Limited to periodic processes

---

Example:

<table>
<thead>
<tr>
<th>Process</th>
<th>Start Time</th>
<th>Completion Time</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>20</td>
<td>60</td>
<td>0.017</td>
</tr>
<tr>
<td>P2</td>
<td>10</td>
<td>40</td>
<td>0.025</td>
</tr>
<tr>
<td>P3</td>
<td>5</td>
<td>30</td>
<td>0.033</td>
</tr>
</tbody>
</table>

Success: \( U_1 + U_2 + U_3 = 75\% \leq 3 \left(\frac{2^{1/3}}{3} - 1\right) = 78\% \)

Fails: P3 misses two deadlines!
Rate Monotonic Scheduling

Example:  

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

Failure: $U_1 + U_2 + U_3 = 83.3\% > 3 \left(\frac{2^{1/3}}{3} - 1\right) = 78\%$