Process Scheduling

- Process scheduling
  - Select a process from ready queue for execution
- Evaluation metrics
  - CPU/device utilization
  - System throughput
  - Waiting time
  - Response time

Processes and State Transitions

- Three states: Ready, Running, and Waiting

When a process makes a transition:
1. from running to waiting
2. from running to ready
3. from waiting to ready
   (3a. a process is created)
4. from running to terminated

Why a process makes a transition:
1. an action of the process
2. occurrence of an external event

Scheduling Policies

**First-Come-First-Served (FCFS)**

- The discipline corresponding to FIFO queuing
- Example — 3 processes with compute times 12, 3, and 3
  - Job arrival order: $P_1$, $P_2$, $P_3$

**Execution**

<table>
<thead>
<tr>
<th>Time</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Average response time** = $(12 + 15 + 18) / 3 = 15$

**Job arrival order: $P_2$, $P_1$, $P_3$**

**Execution**

<table>
<thead>
<tr>
<th>Time</th>
<th>$P_2$</th>
<th>$P_1$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Average response time** = $(3 + 6 + 18) / 3 = 9$
### FCFS Scheduling (Cont’d.)

- **Advantage:**
  - Simple

- **Disadvantages:**
  - Average waiting time is highly variable
  - Short jobs may wait behind long ones !!
  - May lead to poor overlap between I/O and CPU processing
  - CPU bound processes will make I/O bound processes to wait if I/O devices remain idle

### Scheduling Policies

#### Shortest-Job-First (SJF)

- **Select the shortest job first**
  - Enqueue jobs in order of estimated completion time

#### SJF Scheduling --- The Catch

- **It’s unfair!!**
  - Continuous stream of short jobs will starve long jobs

- **Needs clairvoyance**
  - Need to know the execution time of a process
  - Simple solution: ask the user!
  - Yeah, right!!

- **So, what if you don’t subscribe to the Psychic Network??**

---

### Shortest-Job-First Scheduling

*An optimal policy for minimizing response times*

- **Intuition:** Consider an SJF execution of a set of processes

**Average response time = \( r_1 + r_2 + r_3 + r_4 + r_5 + r_6 \)**

**SJF:**

\[
\begin{array}{cccccccc}
P_1 & P_2 & P_3 & P_4 & P_5 & P_6 \\
0 & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 \\
\end{array}
\]

Can switching the execution order reduce response time?

**XYZ:**

\[
\begin{array}{cccccccc}
P_1 & P_2 & P_3 & P_4 & P_5 & P_6 \\
0 & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 \\
\end{array}
\]

**Average response time = \( (r_1 + r_2 + r_3 + r_4 + r_5 + r_6) + (c_4 + c_5 - 2c_3) / 6 \)**
Short-Job-First Scheduling

Estimating execution time

- Jobs are enqueued in order of estimated completion time
  - "Recent history is a good indicator of the near future"

\[ t_n \quad \text{--- duration of the } n\text{th CPU burst} \]
\[ \hat{t}_{n+1} \quad \text{--- predicted duration of the } (n+1)\text{st CPU burst} \]
\[ \hat{t}_{n+1} = t_n + (1-a)t_n \quad \text{for } 0 \leq a \leq 1 \]

Scheduling Policies

Priority Scheduling (PS)

- Assign a priority (a number) to each job and schedule jobs in order of priority
  - Typically low priority values = "high priority"
  - E.g., if priority = \[ \frac{1}{n} \], then a priority scheduler becomes a SJF scheduler.

Priority Scheduling

Avoiding starvation

- Aging
  - Gradually increase a process's priority (decrease its priority value) over time

Non Pre-emptive vs. Pre-emptive Scheduling

- Non Pre-emptive Scheduling:
  - Once a process begins execution, it occupies CPU until it finishes or it blocks
  - Advantage: simplicity, but...
  - Creates problems... (like what?)
  - Examples: FCFS, SJF, F5, ...

- Pre-emptive Scheduling
  - A process is switched back and forth between running and ready states
  - Advantage: more efficient, better capabilities, but...
  - More complex and needs hardware support (e.g., timer interrupts)
  - Examples: Round Robin, Shortest Remaining Time First (SRTF), Multi-level Feedback Queue (MLF)
**Scheduling Policies**

**Round-Robin Scheduling (RR)**

- Allocate the processor in discrete units called *quanta* (or *time-slices*).
- Switch to the next ready process at the end of each quantum.
  - Processes execute every \((n - 1)q\) time units.

![Diagram of Round-Robin Scheduling](image)

**RR Scheduling: Selecting a Time Quantum**

- Too large
  - Long waiting time
  - Degenerates to FCFS in the limit
- Too small
  - Responsive, but...
  - Throughput suffers due to large context switch overhead

- **Goal:**
  - Select a time quantum that balances this tradeoff
  - Rule of thumb: maintain context switch overhead to less than 1%

**Scheduling Policies**

**Multi-level feedback queues (MLF)**

- \(n\) priority levels — priority scheduling between levels, round-robin within a level.
- Quantum size decreases with priority level.
- Jobs are demoted to lower priority levels if they don’t complete within the current quantum.

![Diagram of Multi-level Feedback Queues](image)