Process Scheduling

- Process scheduling
  - Select a process from ready queue for execution
- Evaluation metrics
  - CPU/device utilization
  - Time busy/time observed
  - System throughput
  - Jobs completed/time observed
  - Turnaround time
    - Start-to-finish time for non-interactive jobs
  - Waiting time
  - Time spent in ready queue
  - Response time
    - Start-to-finish time for interactive jobs

Processes and State Transitions

- Three states: Ready, Running, and Waiting
  - Why a process makes a transition:
    1. an action of the process (non-preemptive scheduling)
    2. occurrence of an external event (preemptive scheduling)
  - 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

Scheduling Policies

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

- The discipline corresponding to FIFO queuing
- If a process blocks while executing, CPU is given to next in queue
- Example — 3 processes w/ compute times 12, 3, and 3
  - Job arrival order P₁, P₂, P₃
  - Execution
    | Time | P₁ | P₂ | P₃ |
    |------|----|----|----|
    | 0    |    |    |    |
    | 12   |    |    |    |
    | 15   |    |    |    |
    | 18   |    |    |    |
  - Average response time = (12 + 15 + 18)/3 = 15

**Example**

- Job arrival order P₂, P₃, P₁
  - Execution
    | Time | P₂ | P₃ | P₁ |
    |------|----|----|----|
    | 0    |    |    |    |
    | 3    |    |    |    |
    | 6    |    |    |    |
    | 10   |    |    |    |
  - 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 wait →
      I/O devices remain idle

Scheduling Policies
Shortest-Job-First (SJF)

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

Shortest-Job-First Scheduling
Optimal Mean Turnaround—when jobs are available simultaneously

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

  Mean turnaround time = \( (r_1 + r_2 + r_3 + r_4 + r_5)/6 \)

SJF:

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>9</td>
</tr>
<tr>
<td>P_2</td>
<td>12</td>
</tr>
<tr>
<td>P_3</td>
<td>34</td>
</tr>
<tr>
<td>P_4</td>
<td>62</td>
</tr>
</tbody>
</table>

Can switching the execution order reduce response time?

XYZ:

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>9</td>
</tr>
<tr>
<td>P_2</td>
<td>12</td>
</tr>
<tr>
<td>P_3</td>
<td>34</td>
</tr>
<tr>
<td>P_4</td>
<td>62</td>
</tr>
</tbody>
</table>

Mean turnaround time = \( (r_1 + r_2 + r_3 + r_4 + r_5 + r_6)/6 \)

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 Process Next Scheduling

Estimating execution time

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

\[
\tau_n \quad \text{duration of the } n\text{th CPU burst}
\]
\[
\tau_{n+1} \quad \text{predicted duration of the } (n+1)\text{st CPU burst}
\]
\[
\tau_{n+1} = \alpha \tau_n + (1-\alpha)\tau_n \quad \text{for } 0 \leq \alpha \leq 1
\]

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 = \( \tau_n \), then a priority scheduler becomes a SJF scheduler.

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, PS, ...

- 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 unit 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

```
CPU
/-------------------
|                  |
|  q                |
|                  |
```

- Timer Interrupt
- Process Completion or I/O Request
- \(P_a P_b P_c \ldots\)

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

```
Level 1  Level 2  Level 3  Level n
\[ q = t_0 \]
\[ q = 2t_0 \]
\[ \vdots \]
\[ q = 2^{n-1} t_0 \]
```

High Priority

Low Priority

CPU

Real-time scheduling

- Real-time processes have timing constraints
  - Timing constraints are translated into deadlines or rate requirements
- A set of \(m\) periodic events is schedulable if:
  
  \[
  \sum_{i=1}^{m} C_i \leq \sum_{i=1}^{m} P_i \leq 1
  \]

  where

  - \(C_i\) is the CPU time required to process event \(i\)
  - \(P_i\) is the period (1/rate) at which event \(i\) occurs

Two dominant real-time scheduling policies:

- "Rate-Monotonic" scheduling
  - Priority = period
- "Deadline" scheduling
  - Priority = release time + deadline

Rate monotonic is guaranteed to work if:

\[
\sum_{i=1}^{m} C_i \leq m(2^{\log m} - 1)
\]

or, if \(m \to \infty\)

\[
\sum_{i=1}^{m} C_i \leq m^2
\]
Scheduling Policies

Real-time scheduling

- Real-time processes have **timing constraints**
  - Timing constraints are translated into deadlines or rate requirements

Two dominant real-time scheduling policies
- **Rate-Monotonic scheduling**
  - Priority = 1/rate (the "period")
- **Deadline scheduling**
  - Priority = release time + deadline

Example: Digital video playout

```c
/* Main processing loop */
loop
  data = read(network)
  video_frame = decompress(data)
  write(frame_buffer, video_frame)
end loop
```

Timing constraint: Execute loop once every 33 ms.

Scheduling Policies example

Consider scheduling audio and video playout processes
- The audio process processes 1 audio sample every 20 ms
- The video process processes 1 video frame every 33 ms

Audio Process

```
| 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45
```

Video Process

```
| 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45
```

**Rate-monotonic scheduling**: priority = 1/rate = period

Scheduling Policies: Summary

- **FCFS**:
  - Not fair, and poor average waiting times
- **Priority Scheduling**:
  - Not fair and starvation possible
- **Round Robin**:
  - Fair, but poor average waiting times
- **SJF/SRTF**:
  - Not fair, but average waiting time is minimized
  - Requires accurate prediction of computation times
  - Starvation is possible
- **MLF**:
  - An approximation to SJF
- **Real-time scheduling**:
  - Needed to meet timeliness constraints
- **Lottery Scheduling**:
  - Assign tickets to processes in proportion that reflects the desired relative CPU usage