CPU Scheduling Algorithms

Notice: The slides for this lecture have been largely based on those from an earlier edition of the course text Operating Systems Concepts with Java, by Silberschatz, Galvin, and Gagne. Many, if not all, the illustrations contained in this presentation come from this source.
Basic Concepts

- You want to maximize **CPU utilization** through the use of multiprogramming.

- Each process repeatedly goes through cycles that alternate CPU execution (a **CPU burst**) and I/O wait (an **I/O wait**).

- Empirical evidence indicates that CPU-burst lengths have a distribution such that there is a large number of short bursts and a small number of long bursts.
Preemptive Scheduling

- In **cooperative** or **nonpreemptive** scheduling, when a process takes the CPU, it keeps it until the process either enters waiting state or terminates.

- In **preemptive scheduling**, a process holding the CPU may lose it. Preemption causes context-switches, which introduce overhead. Preemption also calls for care when a process that loses the CPU is accessing data shared with another process or kernel data structures.
Scheduling Criteria

These are **performance** metrics such as:

- **CPU utilization** — high is good; the system works best when the CPU is kept as busy as possible.
- **Throughput** — the number of processes that complete their execution per time unit.
- **Turnaround time** — amount of time to execute a particular process.
- **Waiting time** — amount of time a process has been waiting in the ready queue.
- **Response time** — amount of time it takes from when a request was submitted until the first response is produced, **not output** (for time-sharing environment).

It makes sense to look at averages of these metrics.
Scheduling Algorithms
First-Come, First-Served (FCFS)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>24</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
</tr>
</tbody>
</table>

- Suppose that the processes arrive in the order: $P_1, P_2, P_3$
- The Gantt Chart for the schedule is:

<table>
<thead>
<tr>
<th></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>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
FCFS

Suppose that the processes arrive in the order $P_2, P_3, P_1$

- The Gantt chart for the schedule is:
  
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_2$</td>
<td>$P_3$</td>
<td>$P_1$</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case.
- *Convoy effect*: all processes are stuck waiting until a long process terminates.
Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - **Nonpreemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - **Preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is **optimal** – gives minimum average waiting time for a given set of processes.

**Question:** Is this practical? How can one determine the length of a CPU-burst?
Non-Preemptive SJF

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

- SJF (non-preemptive)

![Process Scheduling Diagram]

- Average waiting time = $(0 + 6 + 3 + 7)/4 = 4$
Preemptive SJF

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

- SJF (preemptive)

Average waiting time = $(9 + 1 + 0 + 2)/4 = 3$
Determining Length of Next CPU-Burst

• We can only estimate the length.
• This can be done by using the length of previous CPU bursts, using exponential averaging:

\[ \tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n \]

1. \( t_n \) = actual length of \( n \)th CPU burst
2. \( \tau_{n+1} \) = predicted value for the next CPU burst
3. \( \alpha, 0 \leq \alpha \leq 1 \)
Exponential Averaging

\[ \tau_{n+1} = \alpha \tau_n + (1 - \alpha) \tau_n \]

Note that \( \alpha \) determines the relative weight on the measured length of the previous CPU-burst and on the estimated length of the previous CPU-burst.

\[
\begin{align*}
\tau_1 &= \alpha t_0 + (1 - \alpha)\tau_0 \\
\tau_2 &= \alpha \tau_1 + (1 - \alpha)\tau_1 = \alpha t_1 + (1 - \alpha)(\alpha t_0 + (1 - \alpha)\tau_0) = \\
&= \alpha t_1 + (1 - \alpha)\alpha t_0 + (1 - \alpha)^2 \tau_0 \\
&\vdots \\
\tau_{n+1} &= \alpha t_n + (1 - \alpha)\alpha \tau_{n-1} + \cdots + (1 - \alpha)^n \alpha t_{n-j} + \cdots + (1 - \alpha)^{n+1} \tau_0
\end{align*}
\]
Prediction of the Length of the Next CPU-Burst

<table>
<thead>
<tr>
<th>CPU burst ($t_j$)</th>
<th>6</th>
<th>4</th>
<th>6</th>
<th>4</th>
<th>13</th>
<th>13</th>
<th>13</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;guess&quot; ($t_j$)</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
Priority Scheduling

- A priority number (integer) is associated with each process.

- The CPU is allocated to the process with the highest priority (smallest integer = highest priority)
  - Preemptive
  - Nonpreemptive

- SJF is a priority scheduling where priority is the predicted next CPU-burst time.
- Problem: Starvation – low priority processes may never execute.
- Solution: Aging – as time progresses increase the priority of the process.
Round Robin (RR)

- Each process gets a small unit of CPU time (time \textit{quantum}), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.

- If there are $n$ processes in the ready queue and the time quantum is $q$, then each process gets $1/n$ of the CPU time in chunks of at most $q$ time units at once. No process waits more than $(n-1)q$ time units.

- Performance:
  - $q$ large $\Rightarrow$ FIFO.
  - $q$ small $\Rightarrow$ $q$ must be large with respect to context switch, otherwise overhead is too high.
## RR with Time Quantum = 20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>53</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>17</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>68</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>24</td>
</tr>
</tbody>
</table>

- The Gantt chart is:

<table>
<thead>
<tr>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>P_4</th>
<th>P_1</th>
<th>P_3</th>
<th>P_4</th>
<th>P_1</th>
<th>P_3</th>
<th>P_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20</td>
<td>37</td>
<td>57</td>
<td>77</td>
<td>97</td>
<td>117</td>
<td>121</td>
<td>134</td>
<td>154</td>
</tr>
</tbody>
</table>

- Typically, higher average turnaround than SJF, but better response.
Time Quantum and Context Switch Time

**Question:** What influences the choice of value for the quantum?
Turnaround Time Varies with the Time Quantum

<table>
<thead>
<tr>
<th>Process</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>6</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>7</td>
</tr>
</tbody>
</table>
Multilevel Queue

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)

- Each queue has its own scheduling algorithm.
  - foreground: RR
  - background: FCFS

- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes, i.e., 80% to foreground in RR.
  - 20% to background in FCFS.
Multilevel Queue Scheduling

highest priority

system processes

interactive processes

interactive editing processes

batch processes

student processes

lowest priority
Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way.

- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues,
  - scheduling algorithms for each queue,
  - method used to determine when to upgrade a process,
  - method used to determine when to demote a process,
  - method used to determine which queue a process will enter when that process needs service.
Example of Multilevel Feedback Queue

- Three queues:
  - $Q_0$ – time quantum 8 milliseconds
  - $Q_1$ – time quantum 16 milliseconds
  - $Q_2$ – FCFS

- Scheduling
  - A new job enters queue $Q_0$ which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue $Q_1$.
  - At $Q_1$ job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue $Q_2$. 
Multilevel Feedback Queues

quantum = 8

quantum = 16

FCFS