Priorities

- Scheduler will always choose a process of higher priority over one of lower priority
- Have multiple ready queues to represent each level of priority
- Lower-priority may suffer starvation
  - Allow a process to change its priority based on its age or execution history
Figure 9.4 Priority Queuing
Decision Mode

• Nonpreemptive
  – Once a process is in the running state, it will continue until it terminates or blocks itself for I/O

• Preemptive
  – Currently running process may be interrupted and moved to the Ready state by the operating system
  – Allows for better service since any one process cannot monopolize the processor for very long
Process Scheduling Example

Table 9.4 Process Scheduling Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>
First-Come-First-Served (FCFS)

- Each process joins the Ready queue
- When the current process ceases to execute, the oldest process in the Ready queue is selected
First-Come-First-Served (FCFS)

- Also called FIFO
- Performs much better for long processes
  - A short process may have to wait a very long time before it can execute
- Favors CPU-bound processes
  - I/O processes have to wait until CPU-bound process completes
**Round-Robin**

- Uses preemption based on a clock
  - quantum $q$
- An amount of time is determined that allows each process to use the processor for that length of time
Round-Robin

- Clock interrupt is generated at periodic intervals
- When an interrupt occurs, the currently running process is placed in the read queue
  - Next ready job is selected
- Known as *time slicing*
(a) Time quantum greater than typical interaction

(b) Time quantum less than typical interaction
Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler
Shortest Process Next

- Nonpreemptive policy
- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>
Shortest Process Next

- Need to predict (or estimate) run time
- If estimated time for process not correct, the operating system may abort it
- Possibility of starvation for longer processes
Shortest Process Next

- Predictions (using simplest calculations)
  - $T_i$ time for $i$-th instance of process
  - $S_i$ predicted execution time for $i$-th instance
  - Simplest scenario, e.g. batch processing in burst mode
    \[ S_{n+1} = \frac{1}{n} \sum_{i=1}^{n} T_i \]
  - Avoiding recalculating entire sum
    \[ S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n \]
Shortest Process Next

• Previous calculations assumed equal weight
  – typically give higher weight to recent instances
  – exponential averaging: $\alpha$ a constant weight factor ($0 < \alpha < 1$)

\[
S_{n+1} = \alpha T_n + (1 - \alpha) S_n
\]
\[
S_{n+1} = \alpha T_n + (1 - \alpha) \alpha T_{n-1} + \ldots
\]
\[
+ (1 - \alpha)^i \alpha T_{n-i} + \ldots + (1 - \alpha)^n S_1
\]
Figure 9.8  Exponential Smoothing Coefficients
Figure 9.9 Use of Exponential Averaging