Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Algorithm Evaluation
Basic Concepts

- Maximum CPU utilization obtained with multiprogramming.
- A CPU burst is when CPU performs useful computation.
- An I/O burst consists of I/O operations being performed.
- CPU burst distribution.
Alternating Sequence of CPU And I/O Bursts
Histogram of CPU-burst Times
CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state.
  2. Switches from running to ready state.
  3. Switches from waiting to ready.
  4. Terminates.

- Scheduling under 1 and 4 is *nonpreemptive*.
- All other scheduling is *preemptive*. 
Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context.
  - switching to user mode.
  - jumping to the proper location in the user program to restart that program.

- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running.
Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible (40% lightly loaded, 90% heavily loaded system)
- Throughput – # 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)
Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time
First-Come, First-Served (FCFS) Scheduling

<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:

```
    P1  P2  P3
  0  24  27  30
```

- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
Suppose that the processes arrive in the order \( P_2, P_3, P_1 \).

- The Gantt chart for the schedule is:

<table>
<thead>
<tr>
<th></th>
<th>P₂</th>
<th>P₃</th>
<th>P₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
<td>30</td>
</tr>
</tbody>
</table>

- Waiting time for \( P_1 = 6; P_2 = 0; P_3 = 3 \)
- Average waiting time: \( \frac{6 + 0 + 3}{3} = 3 \)
- Much better than previous case.
- *Convoy effect* – schedule short process behind long process.
Shortest-Job-First (SJF) Scheduling

- 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.
Example of Non-Preemptive SJF

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0.0</td>
<td>7</td>
</tr>
<tr>
<td>P₂</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>P₄</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

- SJF (non-preemptive)

- Average waiting time = (0 + 6 + 3 + 7)/4 = 4
Example of 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 = $\frac{9 + 1 + 0 + 2}{4} = 3$
Determining Length of Next CPU Burst

- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging.
Priority Scheduling

- A priority number (integer) is associated with each process.
- The CPU is allocated to the process with the 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 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 (FCFS policy).
  - \( q \) small \( \Rightarrow \) \( q \) must be large with respect to context switch, otherwise overhead is too high.
Example of 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></th>
<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>
</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.
- Rule of Thumb: 80% of CPU bursts should be shorter than quantum.
Time Quantum and Context Switch Time
Turnaround Time Varies With The Time Quantum

<table>
<thead>
<tr>
<th>process</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>6</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</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

- Scheduler first executes jobs in $Q_0$ and only when $Q_0$ is empty does it execute jobs in $Q_1$ and the same for $Q_1$ and $Q_2$.

- 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 end of 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
Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous* or *heterogeneous processors* within a multiprocessor affect which processes can be executed by each processor.
- *Load Balancing.*
Real-Time Scheduling

- **Hard real-time** systems – required to complete a critical task within a guaranteed amount of time.

- **Soft real-time** computing – requires that critical processes receive priority over less fortunate ones.
Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload. (too much exact knowledge and OS are non-deterministic).

Queueing models – mathematical models working on distribution of CPU and I/O bursts (gives only an approximation since assumptions have to be made and mathematics of certain algorithms is difficult to work with).

Simulation – record processes and simulates response (expensive, difficult to design simulator and requires hours of computer time, no guarantee for accuracy).

Implementation (costliest and OS environment may change).
Evaluation of CPU Schedulers by Simulation
# Solaris 2 Scheduling

<table>
<thead>
<tr>
<th><strong>global priority</strong></th>
<th><strong>scheduling order</strong></th>
<th><strong>class-specific priorities</strong></th>
<th><strong>scheduler classes</strong></th>
<th><strong>run queue</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>highest</td>
<td>first</td>
<td>real time</td>
<td></td>
<td>kernel threads of real-time LWPs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
<td></td>
<td>kernel service threads</td>
</tr>
<tr>
<td>lowest</td>
<td>last</td>
<td>interactive and time sharing</td>
<td></td>
<td>kernel threads of interactive and time-sharing LWPs</td>
</tr>
</tbody>
</table>
# Windows 2000 Priorities

<table>
<thead>
<tr>
<th>Priority</th>
<th>Real-time</th>
<th>High</th>
<th>Above normal</th>
<th>Normal</th>
<th>Below normal</th>
<th>Idle priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-critical</td>
<td>31</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Highest</td>
<td>26</td>
<td>15</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Above normal</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Normal</td>
<td>24</td>
<td>13</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Below normal</td>
<td>23</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Lowest</td>
<td>22</td>
<td>11</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Idle</td>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
End of Section