**CPU Scheduling**

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
  - Batch systems
  - Interactive systems

Based on original slides by Silberschatz, Galvin and Gagne

---

**Basic Concepts**

- CPU–I/O Burst Cycle
  - Process execution consists of a cycle of CPU execution and I/O wait.
  - CPU-Bound Processes
  - I/O-Bound Processes

---

**Basic Concepts (2)**

- Maximum CPU utilization obtained with multiprogramming
  - Different part of the systems can be active simultaneously allowing parallel execution of processes
  - The scheduling algorithm should mix appropriately CPU-bound and I/O-Bound Processes
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 in different situations:
  - Non-preemptive scheduling
  - The running process terminates
  - The running process performs an I/O operation or waits for an event
  - Preemptive scheduling
  - The running process has exhausted its time slice
  - A process A transits from blocked to ready and is considered more important than process B that is currently running

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - Context Switch
  - 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.

- Context-switches should be minimized

Type of scheduling

- Batch Systems 
  - Maximize the resource utilization

- Interactive Systems 
  - Minimize response times

- Real-Time Systems 
  - Meet Temporal Constraints
Objectives

- General
  - Fairness
  - Load Balancing

- Batch Systems
  - CPU utilization (% of time the CPU is executing processes)
  - Throughput (# of processes executed per time unit)
  - Turnaround time (amount of time to execute a particular process)

- Interactive Systems
  - Response time (amount of time it takes from when a request was submitted until the first response is produced, not output)

- Real-Time Systems
  - Temporal Constraints
  - Avoid data loss
  - Avoid Quality of Service (QoS) degradation

Scheduling for Batch Systems

- FCFS
  - First-Come First-Served

- SJF
  - Shortest Job First

- SRJF
  - Shortest Remaining Job First

First-Come, First-Served (FCFS)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>24</td>
</tr>
<tr>
<td>P₂</td>
<td>3</td>
</tr>
<tr>
<td>P₃</td>
<td>3</td>
</tr>
</tbody>
</table>

Suppose that the processes arrive in the order: P₁, P₂, P₃

- Waiting time for P₁ = 0; P₂ = 24; P₃ = 27
- Average waiting time: (0 + 24 + 27)/3 = 17
CPU Scheduling

FCFS (Cont.)

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

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

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:
  - Non-preemptive
    - Once CPU given to the process it cannot be preempted until completes its CPU burst.
  - Preemptive (Shortest-Remaining-Time-First or SRTF),
    - If a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as SJF.
- SJF is optimal
  - Gives minimum average waiting time for a given set of processes that are simultaneously available.

Example of Non-Preemptive SJF

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

<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 (preemptive)
- Average waiting time = \(\frac{9 + 1 + 2 + 0}{4} = 3\)

### Scheduling for Interactive Systems

- Round-Robin (RR)
- Priority-based
- Shortest Process Next
  - Approximated SJF
- Multi-level

### Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum). After this time has elapsed, the process is preempted and added to the end of the ready queue.
- \(n\) processes in the ready queue; time quantum = \(q\)
  - Each process gets \(\frac{1}{n}\) 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 ⇒ FIFO
  - \(q\) small ⇒ \(q\) 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>P₁</td>
<td>53</td>
</tr>
<tr>
<td>P₂</td>
<td>17</td>
</tr>
<tr>
<td>P₃</td>
<td>68</td>
</tr>
<tr>
<td>P₄</td>
<td>24</td>
</tr>
</tbody>
</table>

| Time Quantum and Performance |

- Time quantum impacts on:
  - Number of Context Switches
  - Average Response Time
- RR is fair by definition
  - All processes are given the same chances

Priority Scheduling
- A priority number (integer) is associated with each process:
  - Static vs. Dynamic Priority
- The CPU is allocated to the process with the highest priority (smallest integer = highest priority):
  - Preemptive vs. Non-preemptive
- SJF (SRJF) is a priority scheduling where priority is the predicted (remaining) CPU burst time.
- Problem
  - Starvation – low priority processes may never execute.
- Solution
  - Aging – as time progresses increase the priority of the process.
- Classes of priority
Real-Time Scheduling

- Soft real-time computing – requires that critical processes receive priority over less fortunate ones.
- Hard real-time systems – required to complete a critical task within a guaranteed amount of time.
  - Rate Monotonic
  - Earliest Deadline First

Rate Monotonic Scheduling

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- \( P_1 (T=50 \text{ ms}), P_2 (T=100 \text{ ms}) \)
- \( P_1 \) assigned a higher priority than \( P_2 \).

Earliest Deadline First Scheduling

- Priorities are assigned according to deadlines:
  - the earlier the deadline, the higher the priority;
  - the later the deadline, the lower the priority
- \( P_1: p=50 \text{ ms}, t=25 \text{ ms} \) \( t/p=0.50 \)
- \( P_2: p=80 \text{ ms}, t=35 \text{ ms} \) \( t/p=0.44 \)
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
    - Example: 80% to foreground in RR; 20% to background in FCFS

Multilevel Queue Scheduling

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
**Example of Multilevel Feedback Queue**

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

- Scheduling
  - A new job enters queue $Q_0$, which is served FCFS.
  - When it gains CPU, it 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 $Q_2$.

**Operating System Examples**

- Windows XP scheduling
- Linux scheduling

**Windows XP Scheduling**

- Thread scheduling based on
  - Priority
  - Preemption
  - Time slice
- A thread is executed until one of the following events occur
  - The thread has terminated its execution
  - The thread has exhausted its assigned time slice
  - The has executed a blocking system call
  - A thread with a higher-priority thread has entered the ready queue
Kernel Priorities

- Kernel priority scheme: 32 priority levels
  - Real-time class (16-31)
  - Variable class (1-15)
  - Memory management thread (0)

- A different queue for each priority level
  - Queues are scanned from higher levels to lower levels
  - When no thread is found a special thread (idle thread) is executed

Win32 API priorities

- API Priority classes
  - REALTIME_PRIORITY_CLASS ➔ Real-time Class
  - HIGH_PRIORITY_CLASS ➔ Variable Class
  - ABOVE_NORMAL_PRIORITY_CLASS ➔ Variable Class
  - NORMAL_PRIORITY_CLASS ➔ Variable Class
  - BELOW_NORMAL_PRIORITY_CLASS ➔ Variable Class
  - IDLE_PRIORITY_CLASS ➔ Variable Class

- Relative Priority
  - TIME_CRITICAL
  - HIGHEST
  - ABOVE_NORMAL
  - NORMAL
  - BELOW_NORMAL
  - LOWEST
  - IDLE

Windows XP Priorities

<table>
<thead>
<tr>
<th></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>

Default Base Priority
Class Priority Management

- A thread is stopped as soon as its time slice is exhausted
- Variable Class
  - If a thread stops because time slice is exhausted, its priority level is decreased
  - If a thread exits a waiting operation, its priority level is increased
  - Waiting for data from keyboard, mouse → significant increase
  - Waiting for disk operations → moderate increase
- Background/Foreground processes
  - The time slice of the foreground process is increased (typically by a factor 3)

Linux Scheduling

- Task scheduling based on
  - Priority levels
  - Preemption
  - Time slices
- Two priority ranges: real-time and time-sharing
  - Real-time range from 0 to 99
  - Nice range from 100 to 140
- The time-slice length depends on the priority level

Priorities and Time-slice length

<table>
<thead>
<tr>
<th>numeric priority</th>
<th>relative priority</th>
<th>time quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>highest</td>
<td>200 ms</td>
</tr>
<tr>
<td>99</td>
<td>real-time tasks</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>other tasks</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>lowest</td>
<td>10 ms</td>
</tr>
</tbody>
</table>
RunQueue

- The runqueue consists of two different arrays
  - Active array
  - Expired array

<table>
<thead>
<tr>
<th>active array</th>
<th>expired array</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>task lists</td>
</tr>
<tr>
<td>[0]</td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>[140]</td>
<td></td>
</tr>
</tbody>
</table>

Priority Calculation

- Real time tasks have static priority
- Time-sharing tasks have dynamic priority
  - Based on nice value ± 5
  - ± 5 depends on how much the task is interactive
  - Tasks with low waiting times are assumed to be scarcely interactive
  - Tasks with large waiting times are assumed to be highly interactive
- Priority re-computation is carried out every time a task has exhausted its time slice

Questions?