• Last class:
  – CPU Scheduling
• Today:
  – CPU Scheduling Algorithms and Systems
Scheduling Algorithms

• First-come, First-serve (FCFS)
  – Non-preemptive
  – Does not account for waiting time (or much else)
    • Convoy problem

• Shortest Job First
  – May be preemptive
  – Optimal for minimizing waiting time (how?)

• Lots more… And what do real systems use?
Priority Scheduling

• Each process is given a certain priority “value”.
• Always schedule the process with the highest priority.
Gantt Chart for Priority Scheduling

<table>
<thead>
<tr>
<th>Priority</th>
<th>Duration(s)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>P5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
Priorities

• Note that FCFS and SJF are specialized versions of Priority Scheduling
  – i.e. there is a way of assigning priorities to the processes so that Priority Scheduling would result in FCFS/SJF.

• *What would examples of those priority functions be?*
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

- Approach
  - 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
An example of Round Robin

<table>
<thead>
<tr>
<th>Arrival Time (s)</th>
<th>Job length (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 0</td>
<td>24</td>
</tr>
<tr>
<td>P2 0</td>
<td>3</td>
</tr>
<tr>
<td>P3 0</td>
<td>7</td>
</tr>
</tbody>
</table>

Time Quantum = 4 s
RR Time Quantum

• Round robin is virtually sharing the CPU between the processes giving each process the illusion that it is running in isolation (at 1/n-th the CPU speed).
• Smaller the time quantum, the more realistic the illusion (note that when time quantum is of the order of job size, it degenerates to FCFS).
• But what is the drawback when time quantum gets smaller?
RR Time Quantum

• For the considered example, if time quantum size drops to 2s from 4s, the number of context switches increases to ????

• But context switches are not free!
  – Saving/restoring registers
  – Switching address spaces
  – Indirect costs (cache pollution)
Scheduling Desirables

• SJF
  – Minimize waiting time
    • Requires estimate of CPU bursts
• Round robin
  – Share CPU via time quanta
    • If burst turns out to be “too long”
• Priorities
  – Some processes are more important
    – Priorities enable composition of “importance” factors
• No single best approach -- now what?
Round Robin with Priority

- Have a ready queue for each priority level.
- Always service the non-null queue at the highest priority level.
- Within each queue, you perform round-robin scheduling between those processes.
Round-Robin with Priority

Priority Levels
What is the problem?

• With fixed priorities, processes lower in the priority level can get *starved out*!

• In general, you employ a mechanism to “age” the priority of processes.
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:
  – $Q_0$ – RR with time quantum 8 milliseconds
  – $Q_1$ – RR 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
Scheduling in Systems
Solaris 2 Scheduling
### Solaris Dispatch Table

<table>
<thead>
<tr>
<th>priority</th>
<th>time quantum</th>
<th>time quantum expired</th>
<th>return from sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>10</td>
<td>160</td>
<td>0</td>
<td>51</td>
</tr>
<tr>
<td>15</td>
<td>160</td>
<td>5</td>
<td>51</td>
</tr>
<tr>
<td>20</td>
<td>120</td>
<td>10</td>
<td>52</td>
</tr>
<tr>
<td>25</td>
<td>120</td>
<td>15</td>
<td>52</td>
</tr>
<tr>
<td>30</td>
<td>80</td>
<td>20</td>
<td>53</td>
</tr>
<tr>
<td>35</td>
<td>80</td>
<td>25</td>
<td>54</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>30</td>
<td>55</td>
</tr>
<tr>
<td>45</td>
<td>40</td>
<td>35</td>
<td>56</td>
</tr>
<tr>
<td>50</td>
<td>40</td>
<td>40</td>
<td>58</td>
</tr>
<tr>
<td>55</td>
<td>40</td>
<td>45</td>
<td>58</td>
</tr>
<tr>
<td>59</td>
<td>20</td>
<td>49</td>
<td>59</td>
</tr>
</tbody>
</table>
Linux Scheduling

• Two algorithms: time-sharing and real-time
• Time-sharing (still abstracted)
  – Two queues: active and expired
  – In active, until you use your entire time slice (quantum), then expired
    • Once in expired, Wait for all others to finish (fairness)
  – Priority recalculation -- based on waiting vs. running time
    • From 0-10 milliseconds
    • Add waiting time to value, subtract running time
    • Adjust the static priority

• Real-time
  – Soft real-time
  – Posix.1b compliant – two classes
    • FCFS and RR
    • Highest priority process always runs first
The Relationship Between 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>
List of Tasks Indexed According to Priorities

<table>
<thead>
<tr>
<th>Priority</th>
<th>Task Lists</th>
<th>Priority</th>
<th>Task Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td></td>
<td>[0]</td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[140]</td>
<td></td>
<td>[140]</td>
<td></td>
</tr>
</tbody>
</table>
Summary

• CPU Scheduling
  – Algorithms
  – Combination of algorithms
    • Multi-level Feedback Queues

• Scheduling Systems
  – Solaris
  – Linux
• Next time: Review