CS370 Operating Systems
Colorado State University
Yashwant K Malaiya
Fall 2017  Lecture 10b

Slides based on
• Text by Silberschatz, Galvin, Gagne
• Various sources
Questions from last time

- How are priorities levels determined?
  - Process importance/resource requirements/real-time needs/static or dynamic (aging etc)

- How is time quantum determined?
  - Responsiveness needed vs overhead. May vary.

- Which algorithms can be preemptive?
  - SJF/Priority: possible, RR: required

- Shortest remaining time?: obtain shortest remaining time for all processes

- Does a timer require hardware interrupts?
• Compare:
  – Calling a function
  – Starting a child
  – Creating a thread
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>

- Supplementary information: Suppose that the processes arrive in the order: $P_1$, $P_2$, $P_3$ but almost the same time. The Gantt Chart for the schedule is:

  ![Gantt Chart](image)

- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
- Throughput: $3/30 = 0.1$ per unit

Henry Gantt, 1910s
Example of 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>6</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>8</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>7</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>3</td>
</tr>
</tbody>
</table>

- All arrive at time 0.
- SJF scheduling chart

![Scheduling Chart]

- Average waiting time for $P_1, P_2, P_3, P_4 = (3 + 16 + 9 + 0) / 4 = 7$
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 priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≜ Starvation – low priority processes may never execute
  - Solution ≜ Aging – as time progresses increase the priority of the process

MIT had a low priority job waiting from 1967 to 1973 on IBM 7094! 😊
Example of Priority Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
<td>1 (highest)</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>P5</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

- Arrived at time 0 in order P1, P2, P3, P4, P5 (which does not matter)
- Priority scheduling Gantt Chart

- Average waiting time for P1, .. P5: \((6+0+16+18+1)/5 = 8.2\) msec
Round Robin (RR) with time quantum

- Each process gets a small unit of CPU time (time quantum \( q \)), usually 1-10 milliseconds. After this, the process is preempted, 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.
- Timer interrupts every quantum to schedule next process
- Performance
  - \( q \) large \( \Rightarrow \) FIFO
  - \( q \) small \( \Rightarrow q \) must be large with respect to context switch, otherwise overhead is too high (overhead typically in 0.5% range)
**Example of RR with Time Quantum = 4**

<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>

- Arrive a time 0 in order $P_1$, $P_2$, $P_3$: The Gantt chart is:

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_1$</th>
<th>$P_1$</th>
<th>$P_1$</th>
<th>$P_1$</th>
<th>$P_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>$P_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$P_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$P_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>$P_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td>$P_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td>$P_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$P_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$P_1$</td>
<td>$P_1$</td>
<td></td>
</tr>
</tbody>
</table>

- Waiting times: 10-4 = 6, 4, 7, average $17/3 = 5.66$ units
- Typically, higher average turnaround than SJF, but better response
- $q$ should be large compared to context switch time
- $q$ usually **10ms to 100ms**, context switch < 10 µsec

**Response time:** Arrival to beginning of execution: mean(0, 4, ...)
**Turnaround time:** Arrival to finish of execution: mean(30, 7, ...)
Performance Characteristics

- RR (Round Robin)
  - Average waiting time often high
  - Good average response time
    - Important for interactive or timesharing systems
  - To reduce response times, want smaller time quantum
    - But, smaller time quantum increases system overhead

- SJF
  - Best average waiting time
  - Some overhead with respect to estimates of CPU burst length & ordering ready ‘queue’
Multilevel Queue Scheduling

- System processes
- Interactive processes
- Interactive editing processes
- Batch processes
- Student 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
    • Perhaps process not getting enough cpu time, aging
  – method used to determine when to demote a process
    • Perhaps taking too much time
  – 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 (no time quantum limit)

• 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 $Q_2$

CPU-bound: priority falls, quantum raised,
I/O-bound: priority rises, quantum lowered
Thread Scheduling

• Thread scheduling is similar
• Distinction between user-level and kernel-level threads
• When threads are supported, threads are scheduled, not processes

Scheduling competition
• Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  — Known as **process-contention scope (PCS)** since scheduling competition is within the process
  — Typically done via priority set by programmer
• Kernel thread scheduled onto available CPU is **system-contention scope (SCS)** — competition among all threads in system

LWP layer between kernel threads and user threads in some older OSs
Iclicker Quiz
Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- **Assume Homogeneous processors** within a multiprocessor
- **Asymmetric multiprocessing** – individual processors can be dedicated to specific tasks at design time
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling,
  - all processes in common ready queue, or
  - each processor has its own private queue of ready processes
    - Currently, most common
- **Processor affinity** – process has affinity for processor on which it is currently running because of info in cache
  - soft affinity: try but no guarantee
  - hard affinity can specify processor sets
Note that memory-placement algorithms can also consider affinity Non-uniform memory access (NUMA), in which a CPU has faster access to some parts of main memory.
Multiple-Processor Scheduling – Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
  - **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
  - **Pull migration** – idle processors pulls waiting task from busy processor
  - Combination of push/pull may be used.
Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core now common
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens
  - See next
Multithreaded Multicore System

Memory stalls due to cache miss

Single thread

Temporal multithreading. (Simultaneous multithreading allows threads to compute in parallel)
Real-Time CPU Scheduling

- Can present obvious challenges
  - **Soft real-time systems** – no guarantee as to when critical real-time process will be scheduled
  - **Hard real-time systems** – task must be serviced by its deadline

- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time

- For hard real-time must also provide ability to meet deadlines
  - **Periodic** ones require CPU at constant intervals
Virtualization and Scheduling

- Virtualization software schedules multiple guests onto CPU(s)
- Each guest doing its own scheduling
  - Not knowing it doesn’t own the CPUs
  - Can effect time-of-day clocks in guests
- VMM has its own scheduler
- Various approaches have been used
  - Workload aware, Guest OS cooperation, etc.
Operating System Examples

- Solaris scheduling: 6 classes, Inverse relationship between priorities and time quantum
- Windows XP scheduling: 32 priority levels (real-time, not real-time levels)
- Linux 2.5 kernel scheduling: 2.5 kernel
  - Priority-based, preemptive
  - Two priority ranges (real time and nice)
  - Time quantum longer for higher priority processes (ranges from 10ms to 200ms)
- OSX
  - Multilevel feedback queue with round robin
- Linux 2.6.23 Kernel scheduling: newer – “Completely fair scheduler” (CFS):
  - 140 priority levels (“nice value”)
  - “Virtual run time” for each process (weighted CPU execution time)
  - Smallest value of vrt picked for using CPU
  - Variable timeslice, number and priority of the tasks in the queue
- Approaches evolve.