Multiprocessor and Real-Time Scheduling

Chapter 10
Classifications of Multiprocessor

✓ Loosely coupled multiprocessor
  ▪ each processor has its own memory and I/O channels

✓ Functionally specialized processors
  ▪ such as I/O processor
  ▪ controlled by a master processor

✓ Tightly coupled multiprocessing
  ▪ processors share main memory
  ▪ controlled by operating system
Independent Parallelism

✔ Separate processes running
✔ No synchronization
✔ Example is time sharing
  ▪ average response time to users is less
Very Coarse Parallelism

- Distributed processing across network nodes to form a single computing environment
- Good when the interaction among processes is infrequent
  - overhead of network would slow down communications
Coarse Parallelism

✓ Similar to running many processes on one processor except it is spread to more processors
✓ Multiprocessing
Medium Parallelism

- Parallel processing or multitasking within a single application
- Single application is a collection of threads
- Threads usually interact frequently
Process Scheduling

✓ Single queue for all processes
✓ Multiple queues are used for priorities
✓ All queues feed to the common pool of processors
✓ Specific scheduling disciplines are less important with more than one processor
Threads

✓ Each thread executes separately from the rest of the process
✓ An application can be a set of threads that cooperate and execute concurrently in the same address space
✓ Running each thread on a separate processor yields a dramatic gain in performance
Multiprocessor Thread Scheduling

✓ Load sharing
  ▪ processes are not assigned to a particular processor

✓ Gang scheduling
  ▪ a set of related threads is scheduled to run on a set of processors at the same time
Multiprocessor Thread Scheduling

✓ Dedicated processor assignment
  ▪ threads are assigned to a specific processor

✓ Dynamic scheduling
  ▪ number of threads can be altered during the course of execution
Load Sharing

✓ Load is distributed evenly across the processors
✓ Assures no processor is idle
✓ No centralized scheduler required
✓ Use global queues
✓ Uniprocessor scheduling directly applies to this case
✓ The most common method
Disadvantages of Load Sharing

- Central queue needs mutual exclusion
  - may be a bottleneck when more than one processor looks for work at the same time
- With global queue, preempted threads are unlikely to resume execution on the same processor
  - hence local processor cache use is less efficient
- If all threads are in the global queue, eligible threads cannot gain access to the idle processors at the same time
Gang Scheduling

- Simultaneous scheduling of threads that make up a single process; assignment of threads to processors kept until preempted
- Useful for applications where performance severely degrades when any part of the application is not running
- Rationale: threads often need to synchronize with each other
Dedicated Processor Assignment

✓ When application is scheduled, its threads are assigned to a processor for the duration of application’s execution

✓ Disadvantage: Some processors may be idle

✓ Advantage: Avoids process switching
Dynamic Scheduling

✓ Number of threads in a process changes dynamically (by the application)

✓ Operating system adjusts the processor load using some of these strategies:
  - assign idle processors to new threads
  - new arrivals may be assigned to a processor by taking away a processor from some other application that uses > 1 processor
  - hold request until processor is available
  - new arrivals may be given a processor before existing running applications
Real-Time Systems

✓ Correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced

✓ Tasks or processes attempt to control or react to events that take place in the outside world

✓ These events occur in “real time” and processes must keep up with them
Real-Time Systems

✓ Control of laboratory experiments
✓ Process control plants
✓ Robotics
✓ Air traffic control
✓ Telecommunications
Characteristics of Real-Time Operating Systems

✓ Correctness depends not only on the result produced by computation, but also by the timing and deadlines

✓ Deterministic
  ▪ operations are performed at fixed, predetermined times or within predetermined time intervals
  ▪ concerned with how long the operating system delays before acknowledging an interrupt
Characteristics of Real-Time Operating Systems

✓ Responsiveness
  - how long, after acknowledgment, it takes the operating system to service the interrupt
  - includes amount of time to begin execution of the interrupt
  - includes the amount of time to perform the interrupt
Characteristics of Real-Time Operating Systems

✓ User control
   - specify paging
   - what processes must always reside in main memory
   - rights of processes
Characteristics of Real-Time Operating Systems

✓ Reliability
  - degradation of performance may have catastrophic consequences
  - most critical, high priority tasks execute
Features of Real-Time Operating Systems

- Fast context switch
- Small size
- Ability to respond to external interrupts quickly
- Multitasking with interprocess communication tools such as semaphores, signals, and events
- Files that accumulate data at a fast rate
Features of Real-Time Operating Systems

✓ Preemptive scheduling based on priority
  - immediate preemption allows operating system to respond to an interrupt quickly
✓ Minimization of intervals during which interrupts are disabled
✓ Delay tasks for fixed amount of time
✓ Special alarms and timeouts
Real-Time Scheduling

✓ Static table-driven
  ▪ determines statically what the schedule should be; the schedule tells which tasks to dispatch and when

✓ Static priority-driven preemptive
  ▪ traditional priority-driven scheduler is used

✓ Dynamic planning-based
  ▪ Like static, but schedules are periodically recomputed

✓ Dynamic best effort
  ▪ No analysis: OS tries to execute each job before its deadline; a job is aborted, if its deadline is not met
Deadline Scheduling

✓ Real-time applications are not concerned with speed but with completing tasks
✓ Scheduling tasks with the earliest deadline minimized the fraction of tasks that miss their deadlines
  ▪ includes new tasks and amount of time needed for existing tasks
### Scheduling of Real-Time Tasks

<table>
<thead>
<tr>
<th>Arrival times</th>
<th>Service</th>
<th>Starting deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  B  C  D  E</td>
<td>A  C  D  E</td>
<td>B  (missed)  C  E  D  A</td>
</tr>
</tbody>
</table>

**Requirements**

- **Arrival times:** A, B, C, D, E
- **Starting deadline:** B, C, E, D, A

**Earliest deadline**

- **Arrival times:** A, B, C, D, E
- **Service:** A, C, E, D
- **Starting deadline:** B (missed), C, E, D, A

"A" starts because there is no other job. "B" misses its deadline.
Scheduling of Real-Time Tasks

Requirements

Arrival times
0 10 20 30 40 50 60 70 80 90 100 110 120
A B C D E

Starting deadline
B C E D A

Earliest deadline with unforced idle times

Arrival times
A B C D E

Service
B C E D

Starting deadline
B C E D A
Scheduling of Real-Time Tasks

Arrival times
A B C D E

Starting deadline
B C E D A

Requirements

First-come first-served (FCFS)

Arrival times
A B C D E

Service
A C D

Starting deadline
B (missed) C E (missed) D A
UNIX System V Release 4
Scheduling

✓ Set of 160 priority levels divided into three priority classes, each with its queue
✓ Basic kernel is not preemptive; split with *preemption points* to improve processing

<table>
<thead>
<tr>
<th>Priority Class</th>
<th>Global Value</th>
<th>Scheduling Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-time</td>
<td>159</td>
<td>first</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Kernel</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Time-shared</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>last</td>
</tr>
</tbody>
</table>
Windows NT Priority Relationship

2 bands of priorities: 0-15 variable, 16-31 real-time. Priorities are fixed in the real-time band. Uses round-robin within each priority level.

Thread’s dynamic priority goes up, if thread is interrupted for I/O. It goes down if thread hogs CPU.