Chapter 10
Multiprocessor and
Real-Time Scheduling
Contents

● Multiprocessor scheduling
● Real-Time scheduling
  ◦ Real-time systems and real-time OS
  ◦ Real-time scheduling
● Linux scheduling
● Unix SVR4 scheduling
● Windows 2000 scheduling
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 OS
Characterizing Multiprocessors

- Synchronization granularity
  - Frequency of synchronization between processes

- Degree of granularity
  - Independent Parallelism
  - Very Coarse Parallelism
  - Coarse Parallelism
  - Medium Parallelism
  - Fine-Grained Parallelism
Independent Parallelism

- No synchronization among processes
- Multiple unrelated processes
- Typical example is a time sharing system
  - One application is word processing
  - The other one is using a spreadsheet
- Average response time to the users will be less than that of the uniprocessor system
Very Coarse Parallelism

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

- Multiprocessing of concurrent processes in a multiprogramming environment
  - Similar to running many processes on one processor except it is spread to more processors
Medium Parallelism

- Parallel processing or multitasking within a single application
  - Single application is a collection of threads
  - Threads in a process usually interact so frequently
Fine-Grained Parallelism

- Much more complex use of parallelism than is found in the use of threads
  - Parallelism inherent in a single instruction stream
- Highly parallel applications
Table 10.1  Synchronization granularity and processes

<table>
<thead>
<tr>
<th>Grain Size</th>
<th>Description</th>
<th>Synchronization Interval (Instructions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine</td>
<td>Parallelism inherent in a single instruction stream.</td>
<td>&lt;20</td>
</tr>
<tr>
<td>Medium</td>
<td>Parallel processing or multitasking within a single application</td>
<td>20-200</td>
</tr>
<tr>
<td>Coarse</td>
<td>Multiprocessing of concurrent processes in a multiprogramming environment</td>
<td>200-2000</td>
</tr>
<tr>
<td>Very Coarse</td>
<td>Distributed processing across network nodes to form a single computing environment</td>
<td>2000-1M</td>
</tr>
<tr>
<td>Independent</td>
<td>Multiple unrelated processes</td>
<td>(N/A)</td>
</tr>
</tbody>
</table>
Design Issues of Multiprocessor Scheduling

- Assignment of processes to processors
- Use of multiprogramming on individual processors
- Actual dispatching of a process
Assignment of Processes to Processors

- Assign processes to processors on demand
  - Could be static or dynamic
  - Assigned to one processor from activation until its completion

- Means of assigning processes to processors
  - Master/slave architecture
  - Peer architecture
Assignment of Processes to Processors

- Master/slave architecture
  - Key kernel functions always run on a particular processor
    - Master is responsible for scheduling
  - Slaves only execute user programs
    - Slave sends service request to the master
- Disadvantages
  - Failure of master brings down whole system
  - Master can become a performance bottleneck
Assignment of Processes to Processors

- Peer architecture
  - Operating system can execute on any processor
    - Each processor does self-scheduling
  - Complicates the operating system
    - Must ensure that two processors do not choose the same process
    - Need to resolve and synchronize competing claims to resources
Use of Multiprogramming on Individual Processors

- In the environment of coarse-grained or independent synchronization granularity, use of multiprogramming is natural.

- For medium-grained applications running on a multiprocessor, situation is less clear:
  - It is no longer paramount that every single processor be busy as much as possible.
  - An application that consists of a number of threads may run poorly unless all of its threads are available to run simultaneously.
Dispatching a Process

● Actual selection of a process to run
  • Uniprocessor system
    – Use sophisticated algorithms to improve performance
  • Multiprocessor system
    – Simpler approaches may be more effective with less overhead
Process Scheduling

Traditional multiprocessor system

- Processes are not dedicated to processors
- Single queue for all processors
- Multiple queues are used for the case of using priorities
  - All queues feed to the common pool of processors
Multiprocessor Thread Scheduling

- An application can be a set of threads that cooperate and execute concurrently in the same address space.
- Threads running on separate processors yields a dramatic gain in performance.
Multiprocessor Thread Scheduling

- General approaches for scheduling
  - Load sharing
    - Processes are not assigned to a particular processor
    - A global queue is maintained and each processor selects a thread from the queue
  - Gang scheduling
    - A set of related threads is scheduled to run on a set of processors at the same time
  - Dedicated processor assignment
    - Threads are assigned to a specific processor
  - Dynamic scheduling
    - Number of threads in a process can be altered during course of execution
Load Sharing

- A global queue of ready threads is maintained
  - Load is distributed evenly across the processors
    - Assures no processor is idle
  - No centralized scheduler required
    - When a processor is available, scheduling routine is run on that processor
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
- Preempted threads are unlikely to resume execution on the same processor
  - Cache use is less efficient
- It is unlikely that all threads of a program will gain access to the processors at the same time
Gang Scheduling

- Simultaneous scheduling of threads that make up a single process
- Useful for applications where performance severely degrades when any part of the application is not running
- Threads often need to synchronize with each other
Figure 10.2 Example of scheduling groups with four and one threads
Dedicated Processor Assignment

- An extreme form of gang scheduling
  - Dedicate a group of processors to an application for the duration of it
  - No multiprogramming
    - Some processors may be idle
- Can it be efficient?
  - In a highly parallel system, processor utilization is no longer so important
  - Avoidance of process switching should result in a substantial speedup of that program
Dynamic Scheduling

- Number of threads in a process may be altered dynamically by the application
- When a job requests processors
  - If there are idle processors, use them
  - New job may be assigned to a processor that is used by a job currently using more than one processor
  - Hold request until processor is available
- Upon release of processors
  - New job will 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
  - Each task has a deadline

- Tasks or processes attempt to control or react to events that take place in the outside world
  - These events occur in “real-time” and process must be able to keep up with them
Real-Time Systems

- Hard real-time task
  - Task that must meet its deadline

- Soft real-time task
  - Has an associated deadline that is desirable but not mandatory

- Periodic task
  - Task that must be executed periodically
    - Once per period $T$

- Aperiodic task
Real-Time Systems

- Control of laboratory experiments
- Process control plants
- Robotics
- Air traffic control
- Telecommunications
- Military command and control systems
Characteristics of Real-Time OS

- Determinism
- Responsiveness
- User control
- Reliability
- Fail-soft operation
Characteristics of Real-Time OS

- Determinism
  - Operations are performed at fixed, predetermined times or within predetermined time intervals
  - Concerned with how long OS delays before acknowledging an interrupt
    - Most of the requests for service are dictated by external events and timings
Characteristics of Real-Time OS

- Responsiveness
  - How long, after acknowledgment, it takes OS to service the interrupt
  - Aspects of responsiveness
    - Amount of time to begin execution of the interrupt
    - Amount of time to perform the interrupt
    - Effect of interrupt nesting
Characteristics of Real-Time OS

- User control
  - User has much broader control over the system
    - User specify priority
    - User specify paging and swapping
      - what processes must always reside in main memory
      - what disks algorithms to use
    - Specify what rights the processes have
Characteristics of Real-Time OS

● Reliability
  ✷ Degradation of performance may have catastrophic consequences
    – Financial loss
    – Equipment damage
    – Loss of life
Characteristics of Real-Time OS

- Fail-soft operation
  - Ability of a system to fail in such a way as to preserve as much capability and data as possible
  - Stability
    - Real-time system is stable if, in cases where it is impossible to meet all task deadlines, the system will meet the deadlines of its most critical, highest-priority tasks
Features of Real-Time OS

- Fast context switch
- Small size
- Ability to respond to external interrupts quickly
- Multitasking with IPC tools such as semaphores, signals, and events
- Files that accumulate data at a fast rate
- Preemptive scheduling based on priority
- Minimization of intervals during which interrupts are disabled
- Delay tasks for fixed amount of time
- Special alarms and timeouts
Scheduling of a Real-Time Process

Request from a real-time process

Real-time process added to run queue to await its next slice

Process 1  Process 2

Clock tick

Process n  Real-time process

Scheduling time

(a) Round-robin Preemptive Scheduler
Scheduling of a Real-Time Process

(b) Priority-Driven Nonpreemptive Scheduler
Scheduling of a Real-Time Process

Request from a real-time process → Wait for next preemption point

Current process ——> Real-time process

Preemption point ————> Scheduling time

(c) Priority-Driven Preemptive Scheduler on Preemption Points
Scheduling of a Real-Time Process

(d) Immediate Preemptive Scheduler

Figure 10.4 Scheduling of Real-Time Process
Real-Time Scheduling

• Classes of algorithms
  • Static table-driven
    – Try to develop the complete schedule
    – Determines when a task must begin execution
  • Static priority-driven preemptive
    – Traditional priority-driven scheduler can be used
  • Dynamic planning-based
    – An attempt is made to create a schedule that contains the previously scheduled tasks as well as the new arrival
  • Dynamic best effort
    – When a task arrives, the system assigns a priority based on the characteristics of the task
Deadline Scheduling

- Information about a task
  - Ready time
  - Starting deadline: a time by which a task must begin
  - Completion deadline
  - Processing time
  - Resource requirements
  - Priority
  - Subtask structure
    - Mandatory and optional subtask
Schedulability

• Let $\tau = \{ \tau_1, \tau_2, \ldots, \tau_n \}$ be a task set
• $\tau_i$ is said to be schedulable if it meets its deadline all the time
• $\tau$ is said to be schedulable if each task in $\tau$ is schedulable
Earliest Deadline Scheduling

- At each scheduling points, the task with the earliest deadline is selected to be run next
  - Dynamic, priority-based preemptive scheduling
  - Applicable to both periodic and aperiodic tasks
  - Scheduling tasks with the earliest deadline minimized the fraction of tasks that miss their deadlines
### Two Tasks

Table 10.2 Execution Profile of Two Periodic Tasks

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Execution Time</th>
<th>Ending Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(1)</td>
<td>0</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>A(2)</td>
<td>20</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>A(3)</td>
<td>40</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>A(4)</td>
<td>60</td>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>A(5)</td>
<td>80</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(1)</td>
<td>0</td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>B(2)</td>
<td>50</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 10.5  Scheduling of Periodic Real-time Tasks with Completion Deadlines
Rate Monotonic Scheduling

- Assigns priorities to tasks on the basis of their periods
  - Highest-priority task is the one with the shortest period
  - Applicable only to periodic tasks
  - Static, priority-based preemptive scheduling
Figure 10.7  Periodic Task Timing Diagram
Figure 10.8: A Task Set with RMS [WARR91]
Table 10.4 Value of the RMS Upper Bound

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n(2^{\mathcal{H}_n} - 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>0.828</td>
</tr>
<tr>
<td>3</td>
<td>0.779</td>
</tr>
<tr>
<td>4</td>
<td>0.756</td>
</tr>
<tr>
<td>5</td>
<td>0.743</td>
</tr>
<tr>
<td>6</td>
<td>0.734</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$\infty$</td>
<td>$\ln 2 \approx 0.693$</td>
</tr>
</tbody>
</table>
Linux Scheduling

- Scheduling classes
  - SCHED_FIFO: First-in-first-out real-time threads
  - SCHED_RR: Round-robin real-time threads
  - SCHED_OTHER: Other, non-real-time threads
    - Traditional Unix scheduling algorithm is used here

- Within each class multiple priorities may be used
Figure 10.9 Example of Linux scheduling
UNIX SVR4 Scheduling

- Set of 160 priority levels divided into three priority classes
  - Real time(159 ~ 100)
  - Kernel(99 ~ 60)
  - Time-shared(59 ~ 0)
<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>Kernel</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Time-shared</td>
<td>59</td>
<td>last</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Figure 10.10 SVR4 priority classes
UNIX SVR4 Scheduling

- Scheduling
  - Highest preference to real-time processes
  - Next-highest to kernel-mode processes
  - Lowest preference to other user-mode processes

- Processes at a given priority level are executed in round-robin fashion
Figure 10.11  SVR4 Dispatch Queues
Windows 2000 Scheduling

- Priorities are organized into two bands or classes
  - Real time
    - All threads have a fixed priority that never changes
  - Variable
    - Thread’s priority may change during it’s lifetime
- Each band consists of 16 priority levels
- Priority-driven preemptive scheduler
Figure 10.11  Windows 2000 Thread Dispatching Priorities