More on Scheduling

Readings: Sections 10.1 and 10.2

Multiprocessor Systems

Loosely coupled or distributed multiprocessor, or cluster
- Consists of a collection of relatively autonomous systems, each processor having its own main memory and I/O channels

Functionally specialized processors
- There is a master, general-purpose processor;
- Specialized processors are controlled by the master processor and provide services to it

Tightly coupled multiprocessor
- Consists of a set of processors that share a common main memory and are under the integrated control of an operating system

Design Issues

- The approach taken will depend on the degree of granularity of applications and on the number of processors available

Scheduling on a multiprocessor involves three interrelated issues:
- Actual dispatching of a process
- Use of multiprogramming on individual processors
- Assignment of processes to processors
Process Scheduling

- In most traditional multiprocessor systems, processes are not dedicated to processors
- A single queue is used for all processors
  - If some sort of priority scheme is used, there are multiple queues based on priority, all feeding into the common pool of processors
- System is viewed as being a multi-server queuing architecture

Thread Scheduling

- An application can be a set of threads that cooperate and execute concurrently in the same address space
- On a uniprocessor, threads can be used as a program structuring aid and to overlap I/O with processing
- In a multiprocessor system threads can be used to exploit true parallelism in an application
- Dramatic gains in performance are possible in multi-processor systems
- Small differences in thread management and scheduling can have an impact on applications that require significant interaction among threads

1 - Load Sharing

- Simplest approach and the one that carries over most directly from a uniprocessor environment

  Advantages:
  - Load is distributed evenly across the processors, assuring that no processor is idle while work is available to do
  - No centralized scheduler required
  - The global queue can be organized and accessed using any of the schemes discussed in Chapter 9

- Versions of load sharing:
  - First-come-first-served (FCFS)
  - Smallest number of threads first
  - Preemptive smallest number of threads first
Disadvantages of Load Sharing

- Central queue occupies a region of memory that must be accessed in a manner that enforces mutual exclusion
  - Can lead to bottlenecks

- Preemptive threads are unlikely to resume execution on the same processor
  - Caching can become less efficient

- If all threads are treated as a common pool of threads, it is unlikely that all of the threads of a program will gain access to processors at the same time
  - The process switches involved may seriously compromise performance

2 - Gang Scheduling

- Simultaneous scheduling of the threads that make up a single process

  Benefits:
  - Synchronization blocking may be reduced, less process switching may be necessary, and performance will increase
  - Scheduling overhead may be reduced

3 - Dedicated Processor Assignment

- When an application is scheduled, each of its threads is assigned to a processor that remains dedicated to that thread until the application runs to completion

- If a thread of an application is blocked waiting for I/O or for synchronization with another thread, then that thread’s processor remains idle
  - There is no multiprogramming of processors

- Defense of this strategy:
  - In a highly parallel system, with tens or hundreds of processors, processor utilization is no longer so important as a metric for effectiveness or performance
  - The total avoidance of process switching during the lifetime of a program should result in a substantial speedup of that program

4 - Dynamic Scheduling

- For some applications it is possible to provide language and system tools that permit the number of threads in the process to be altered dynamically
  - This would allow the operating system to adjust the load to improve utilization

- Both the operating system and the application are involved in making scheduling decisions
  - The scheduling responsibility of the operating system is primarily limited to processor allocation
  - This approach is superior to gang scheduling or dedicated processor assignment for applications that can take advantage of it
Real-Time Systems

- The operating system, and in particular the scheduler, is perhaps the most important component.
- 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 tasks must be able to keep up with them.

Examples:

- Control of laboratory experiments
- Process control in industrial plants
- Robotics
- Air traffic control
- Telecommunications
- Military command and control systems

Real-time tasks

Deadlines
- Hard: One that must meet its deadline
- Soft: Has an associated deadline that is desirable but not mandatory

Periodicity
- Periodic: Runs every T period of time
- Aperiodic: Has a deadline by which it must finish or start, or both.

Characteristics of Real Time Systems

Real-time operating systems have requirements in five general areas:

- Determinism
- Responsiveness
- User control
- Reliability
- Fail-soft operation
Classes of Real-Time Scheduling Algorithms

Static table-driven approaches
- Performs a static analysis of feasible schedules of dispatching
- Result is a schedule that determines, at run time, when a task must begin execution

Static priority-driven preemptive approaches
- A static analysis is performed but no schedule is drawn up
- Analysis is used to assign priorities to tasks so that a traditional priority-driven preemptive scheduler can be used

Dynamic planning-based approaches
- Feasibility is determined at run time rather than offline prior to the start of execution
- One result of the analysis is a schedule or plan that is used to decide when to dispatch this task

Dynamic best effort approaches
- No feasibility analysis is performed
- System tries to meet all deadlines and aborts any started process whose deadline is missed

1 - Deadline Scheduling

- Real-time operating systems are designed with the objective of starting real-time tasks as rapidly as possible and emphasize rapid interrupt handling and task dispatching

- Real-time applications are generally not concerned with sheer speed but rather with completing (or starting) tasks at the most valuable times

- Priorities provide a crude tool and do not capture the requirement of completion (or initiation) at the most valuable time

Table 10.3
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>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>
</tbody>
</table>

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.
Table 10.4
Execution Profile of Five Aperiodic Tasks

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Execution Time</th>
<th>Starting Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>20</td>
<td>110</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>C</td>
<td>40</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>D</td>
<td>50</td>
<td>20</td>
<td>90</td>
</tr>
<tr>
<td>E</td>
<td>60</td>
<td>20</td>
<td>70</td>
</tr>
</tbody>
</table>

Rate Monotonic Scheduling

Figure 10.7 Periodic Task Timing Diagram

Table 10.5
Value of the RMS Upper Bound

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n(3^{1/n} - 1)$</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>$\infty$</td>
<td>$\ln 2 \approx 0.693$</td>
</tr>
</tbody>
</table>
Figure 10.8 Rate Monotonic Scheduling Example

(a) Arrival times and deadlines for task $T_i = (P_i, C_i)$:

- $P_i$ = period
- $C_i$ = processing time

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$T_2$</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>$T_3$</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

(b) Scheduling results

- $T_3$ deadline miss

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.