Chapter 10 Multiprocessor and Real-Time Scheduling

Agenda

- **Multiprocessor Scheduling**
- **Real-Time Scheduling**
- **Linux Scheduling**
- **Linux Virtual Machine Process Scheduling**

Agenda

- **Multiprocessor Scheduling**
  - Granularity
  - Design Issues
  - Process Scheduling
  - Thread Scheduling
- **Real-Time Scheduling**
  - Background
  - Characteristics of Real-Time Operating Systems
  - Real-Time Scheduling
  - Deadline Scheduling
  - Rate Monotonic Scheduling
  - Priority Inversion

Classifications of 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

Synchronization Granularity and Processes

Grain Size	Description	Synchronization Interval (Instructions)
Fine	Parallelism inherent in a single instruction stream.	<20
Medium	Parallel processing or multitasking within a single application	20-200
Coarse	Multiprocessing of concurrent processes in a multiprogramming environment	200-2000
Very Coarse	Distributed processing across network nodes to form a single computing environment	2000-1M
Independent	Multiple unrelated processes	not applicable

Independent Parallelism

- No explicit synchronization among processes
  - each represents a separate, independent application or job
- Typical use is in a time-sharing system

each user is performing a particular application

multiprocessor provides the same service as a multiprogrammed uniprocessor

because more than one processor is available, average response time to the users will be less

Coarse and Very Coarse-Grained Parallelism

- Synchronization among processes, but at a very gross level
- Good for concurrent processes running on a multiprogrammed uniprocessor
  - can be supported on a multiprocessor with little or no change to user software

Medium-Grained Parallelism

- Single application can be effectively implemented as a collection of threads within a single process
  - programmer must explicitly specify the potential parallelism of an application
  - there needs to be a high degree of coordination and interaction among the threads of an application, leading to a medium-grain level of synchronization
- Because the various threads of an application interact so frequently, scheduling decisions concerning one thread may affect the performance of the entire application

Fine-Grained Parallelism

- Represents a much more complex use of parallelism than is found in the use of threads
- Is a specialized and fragmented area with many different approaches

Design Issues

Scheduling on a multiprocessor involves three interrelated issues:

- actual dispatching of a process
- use of multiprogramming on individual processors
- assignment of processes to processors

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

Assignment of Processes to Processors

Assuming all processors are equal, it is simplest to treat processors as a pooled resource and assign processes to processors on demand

static or dynamic needs to be determined

If a process is permanently assigned to one processor from activation until its completion, then a dedicated short-term queue is maintained for each processor

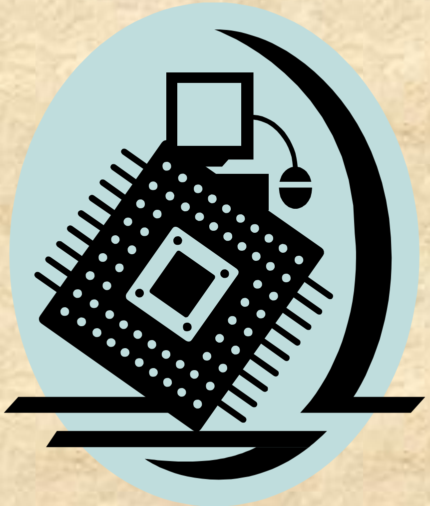
advantage is that there may be less overhead in the scheduling function

allows group or gang scheduling

- A disadvantage of static assignment is that one processor can be idle, with an empty queue, while another processor has a backlog
  - to prevent this situation, a common queue can be used
  - another option is dynamic load balancing

Assignment of Processes to Processors

- Both dynamic and static methods require some way of assigning a process to a processor
- Approaches:
  - Master/Slave
  - Peer



Master/Slave Architecture

- Key kernel functions always run on a particular processor
- Master is responsible for scheduling
- Slave sends service request to the master
- Is simple and requires little enhancement to a uniprocessor multiprogramming operating system
- Conflict resolution is simplified because one processor has control of all memory and I/O resources

Disadvantages:

- failure of master brings down whole system
- master can become a performance bottleneck

Peer Architecture

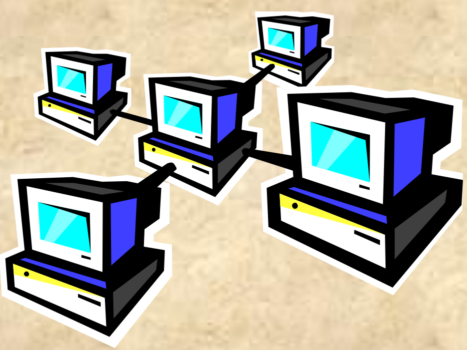
- Kernel can execute on any processor
- Each processor does self-scheduling from the pool of available processes

Complicates the operating system

- operating system must ensure that two processors do not choose the same process and that the processes are not somehow lost from the queue

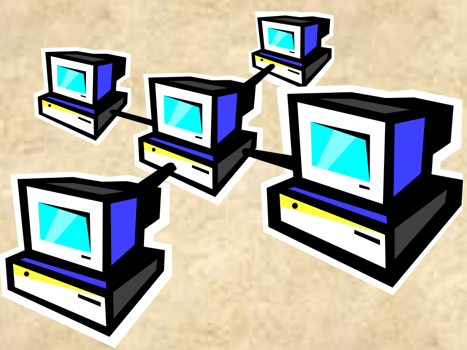
Process Scheduling

- Usually 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
- System is viewed as being a multi-server queuing architecture



Process Scheduling

- With static assignment: should individual processors be multiprogrammed or should each be dedicated to a single process?
- Often it is best to have one process per processor; particularly in the case of multithreaded programs where it is advantageous to have all threads of a single process executing at the same time.



Thread Scheduling

- Thread execution is separated from the rest of the definition of a process
- 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 kernel-level 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

Approaches to Thread Scheduling

Four approaches for multiprocessor thread scheduling and processor assignment are:

Load Sharing

processes are not assigned to a particular processor

Gang Scheduling

a set of related thread scheduled to run on a set of processors at the same time, on a one-to-one basis

Dedicated Processor Assignment

provides implicit scheduling defined by the assignment of threads to processors

Dynamic Scheduling

the number of threads in a process can be altered during the course of execution

Load Sharing

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

Advantages:

- load is distributed evenly across the processors
- no centralized scheduler required
- the global queue can be organized and accessed using any of the schemes discussed in Operating systems-1

- Versions of load sharing:
  - first-come-first-served
  - 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

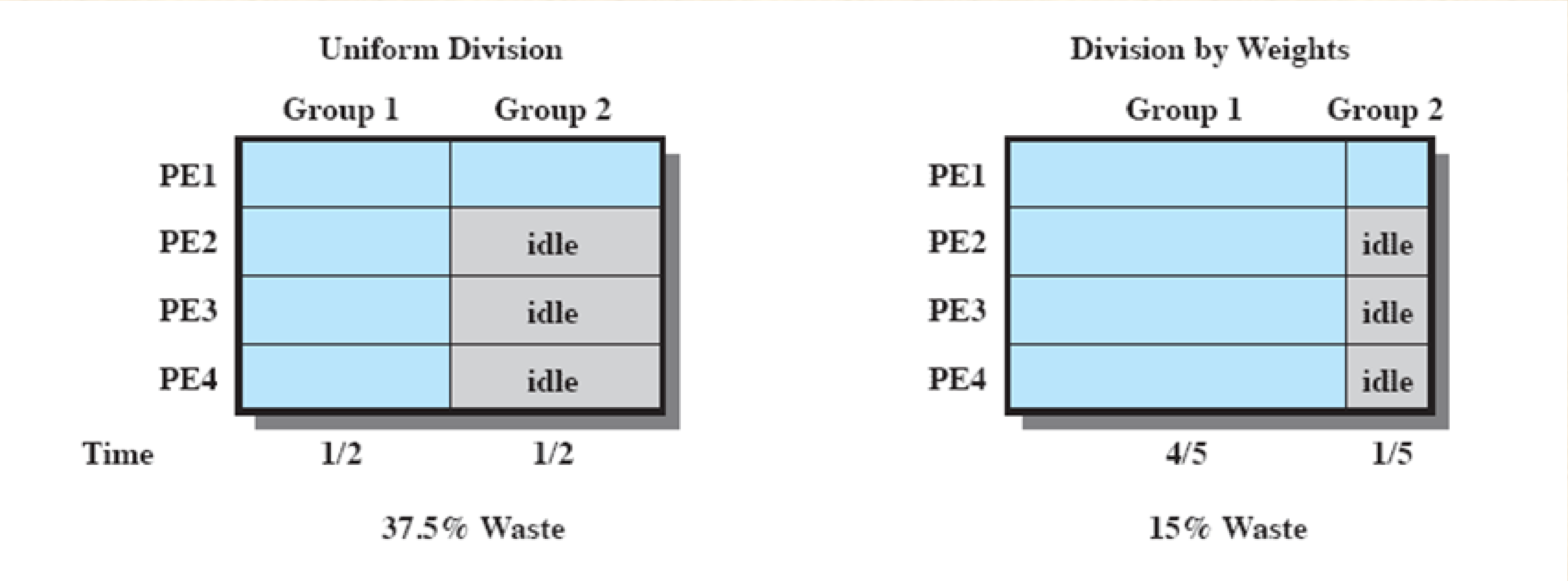
Gang Scheduling

- 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

- Useful for medium-grained to fine-grained parallel applications whose performance severely degrades when any part of the application is not running while other parts are ready to run
- Also beneficial for any parallel application

Figure 10.2
Example of Scheduling Groups With Four and One Threads



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

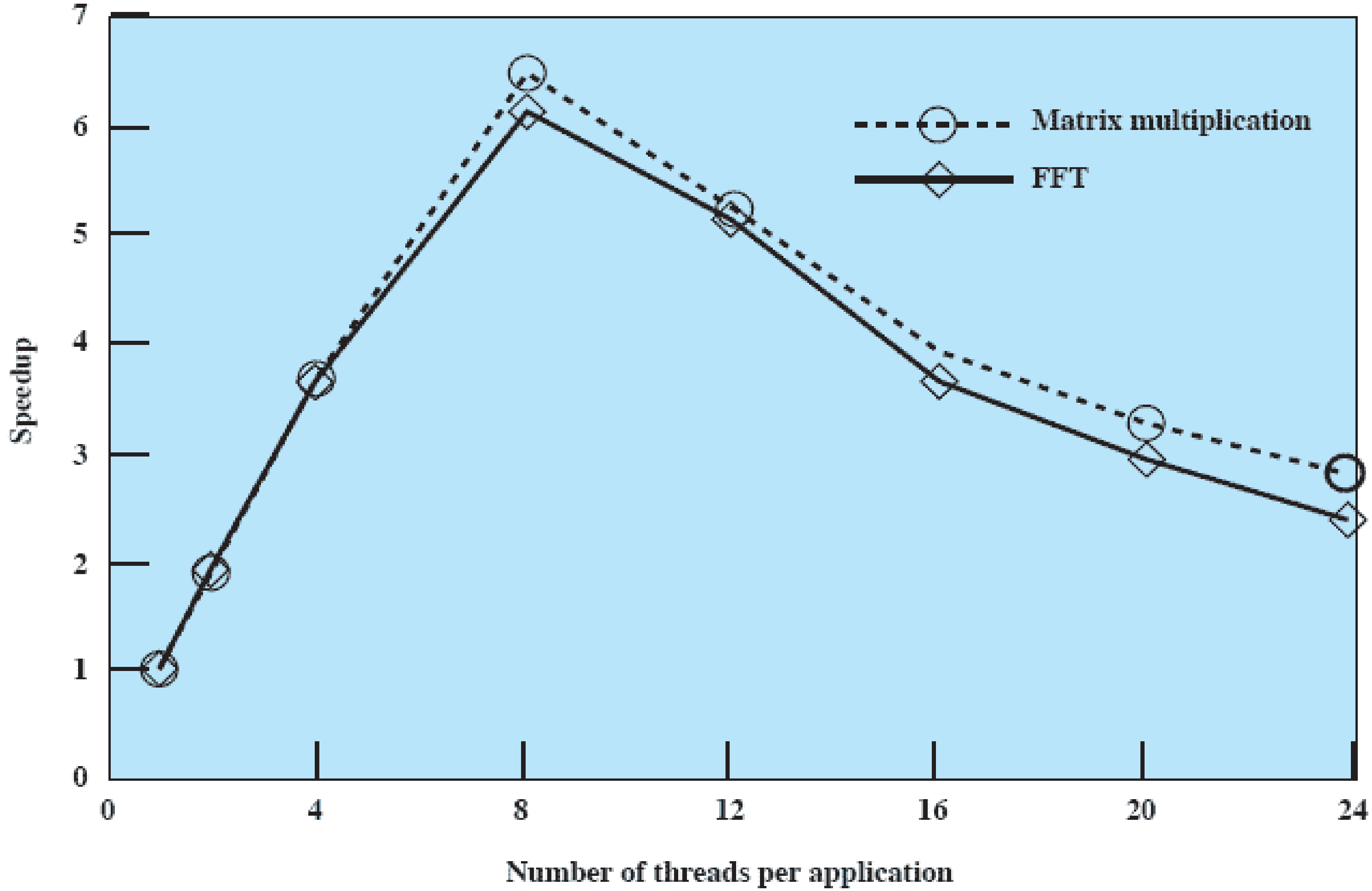


Figure 10.3

Application Speedup as a Function of Number of Threads

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