

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

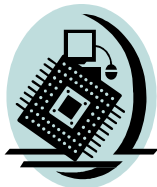
Medium-Grained Parallelism

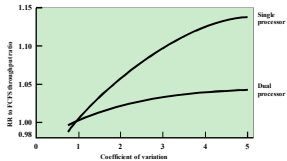
- Single application can be effectively implemented as a collection of threads within a single process
 - programmer must explicitly specify the potential parallelism of an application
 - there needs to be a high degree of coordination and interaction among the threads of an application, leading to a 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



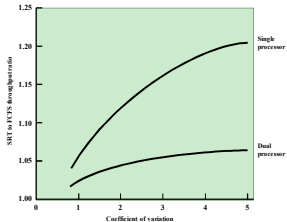
Assignment of Processes to Processors

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





(a) Comparison of RR and FCFS



(b) Comparison of SRT and FCFS

Comparison of Scheduling Performance for One and Two Processors


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



Number of threads per application	Matrix multiplication	FFT
1	1	1
2	1.8	1.8
4	3.8	3.8
8	6.5	6.1
12	5.2	5.1
16	3.9	3.8
20	3.3	3
24	2.8	2.4

Application Speedup as a Function of Number of Threads





Real-Time Scheduling

Characteristics of Real Time Systems

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

Determinism

Responsiveness

User control

Reliability

Fail-soft operation

Reliability

- More important for real-time systems than non-real time systems
- Real-time systems respond to and control events in real time so loss or degradation of performance may have catastrophic consequences such as:
 - financial loss
 - major equipment damage
 - loss of life



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

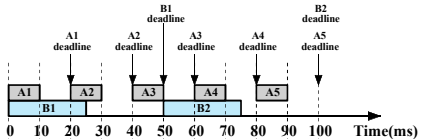
Dynamic planning-based approaches

- 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

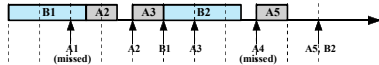
Arrival times, execution times, and deadlines



Fixed-priority scheduling;
A has priority



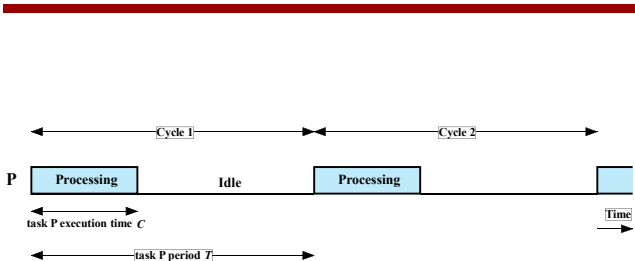
Fixed-priority scheduling;
B has priority



Earliest deadline scheduling
using completion deadlines

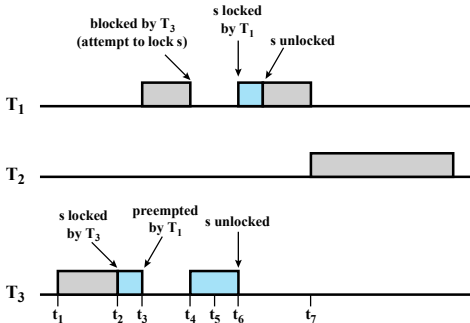


Scheduling of Periodic Real-time Tasks with Completion Deadlines (based on previous Table)



Periodic Task Timing Diagram

Priority Inheritance



(b) Use of priority inheritance