

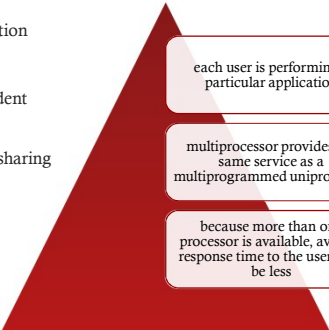
*Operating
Systems:
Internals
and
Design
Principles*



Chapter 10 Multiprocessor, Multicore and Real-Time Scheduling

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

Design Issues

Scheduling on a multiprocessor involves three interrelated issues:

```
graph TD; A([Scheduling on a multiprocessor involves three interrelated issues:]); A --> B([actual dispatching of a process]); A --> C([use of multiprogramming on individual processors]); A --> D([assignment of processes to processors]);
```

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

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

Approaches to Thread Scheduling

processes are not assigned to a particular processor

Load Sharing

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

Gang Scheduling

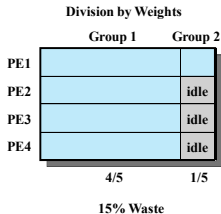
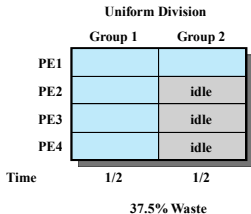
Four approaches for multiprocessor thread scheduling and processor assignment are:

provides implicit scheduling defined by the assignment of threads to processors

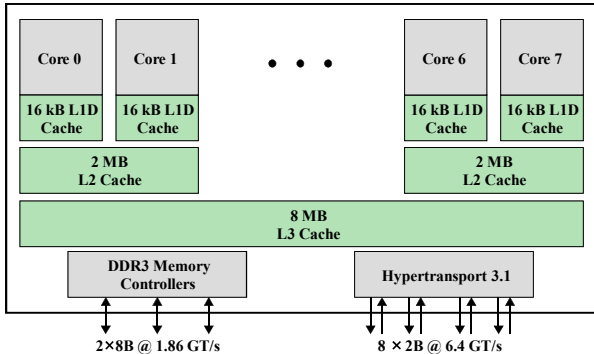
Dedicated Processor Assignment

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

Dynamic Scheduling



Example of Scheduling Groups with Four and One Threads [FEIT90b]



AMD Bulldozer Architecture

Hard and Soft Real-Time Tasks

Hard real-time task

- one that must meet its deadline
- otherwise it will cause unacceptable damage or a fatal error to the system

Soft real-time task

- has an associated deadline that is desirable but not mandatory
- it still makes sense to schedule and complete the task even if it has passed its deadline

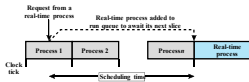


Responsiveness

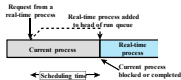
- Together with determinism make up the response time to external events
 - critical for real-time systems that must meet timing requirements imposed by individuals, devices, and data flows external to the system
- Concerned with how long, after acknowledgment, it takes an operating system to service the interrupt

Responsiveness includes:

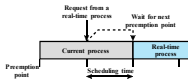
- amount of time required to initially handle the interrupt and begin execution of the interrupt service routine (ISR)
- amount of time required to perform the ISR
- effect of interrupt nesting



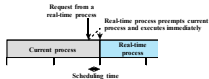
(a) Round-robin Preemptive Scheduler



(b) Priority-Driven Nonpreemptive Scheduler



(c) Priority-Driven Preemptive Scheduler on Preemption Points



(d) Immediate Preemptive Scheduler

Scheduling of Real-Time Process

Information Used for Deadline Scheduling

Ready time

- time task becomes ready for execution

Starting deadline

- time task must begin

Completion deadline

- time task must be completed

Processing time

- time required to execute the task to completion

Resource requirements

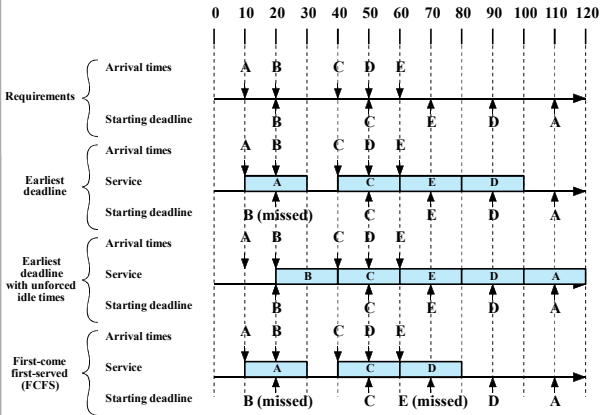
- resources required by the task while it is executing

Priority

- measures relative importance of the task

Subtask scheduler

- a task may be decomposed into a mandatory subtask and an optional subtask



Scheduling of Aperiodic Real-time Tasks with Starting Deadlines

Priority Inversion

- Can occur in any priority-based preemptive scheduling scheme
- Particularly relevant in the context of real-time scheduling
- Best-known instance involved the Mars Pathfinder mission
- Occurs when circumstances within the system force a higher priority task to wait for a lower priority task

Unbounded Priority Inversion

- the duration of a priority inversion depends not only on the time required to handle a shared resource, but also on the unpredictable actions of other unrelated tasks