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

Real-Time Systems

- The operating system, and in particular the scheduler, is perhaps the most important component

Examples:

- control of laboratory experiments
- process control in industrial plants
- robotics
- air traffic control
- telecommunications
- military command and control systems

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.
- 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.

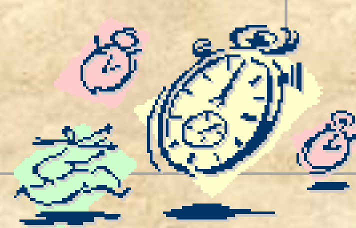
Hard and Soft Real-Time Tasks

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.



Periodic and Aperiodic Tasks

- **Periodic tasks**
  - requirement may be stated as:
    - once per period *T*
    - exactly *T* units apart
- **Aperiodic tasks**
  - has a deadline by which it must finish or start
  - may have a **constraint** on both start and finish time

Characteristics of Real Time Systems

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

- **Determinism**
- **Responsiveness**
- **User control**
- **Reliability**
- **Fail-soft operation**

Determinism

- Concerned with how long an operating system delays before acknowledging an interrupt.
- Operations are performed at fixed, **predetermined** times or within predetermined time **intervals**

Determinism

The extent to which an operating system can **deterministically** satisfy requests depends on:

- the **speed** with which it can respond to interrupts.

- whether the system **has sufficient capacity** to handle all requests within the required time.

Responsiveness

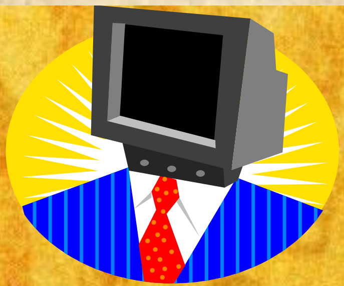
- Together with determinism make up the response time to external events
  - <u>critical for real-time</u> systems that **<u>must meet</u>** timing requirements imposed by individuals, devices, and data flows external to the system.
- Concerned with how long, after acknowledgment, it takes an operating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

Responsiveness

Responsiveness includes:

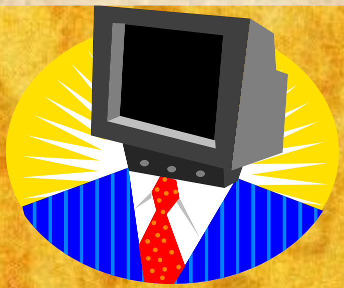
- amount of time required to initially handle the interrupt and **begin execution** of the interrupt service routine (ISR)
- amount of time required to **perform** the ISR
- effect of interrupt nesting

User Control



- Generally **<u>much broader in a real-time</u>** operating system than in ordinary operating systems
- It is essential to allow the **<u>user fine-grained</u>** control over task priority
- User should be able to **<u>distinguish between hard and soft tasks</u>** and to specify relative priorities within each class

User Control

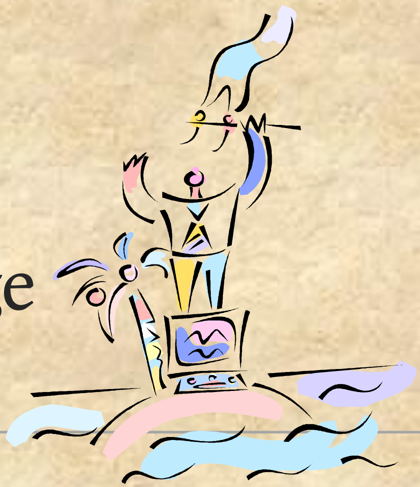


May allow user to specify such characteristics as:

- paging or process swapping
- what processes must always be resident in main memory
- what disk transfer algorithms are to be used
- what rights the processes in various priority bands have

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



Fail-Soft Operation

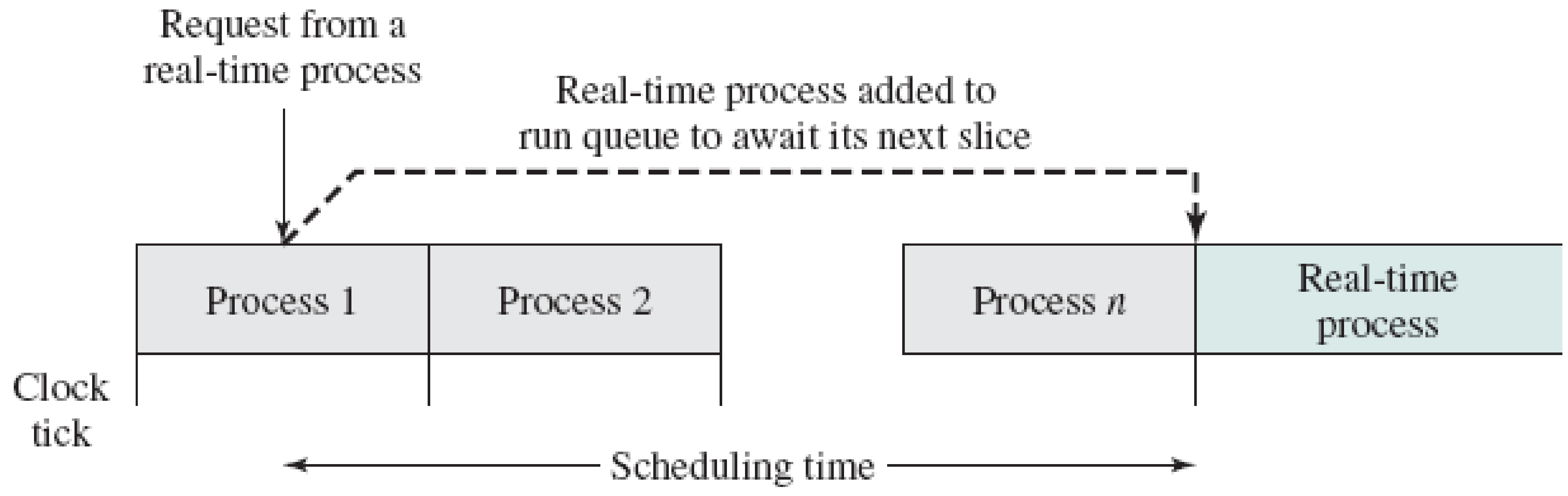
- A characteristic that refers to the **ability of a system to fail** in such a way as to preserve as much capability and data as possible
- Important aspect is **stability**
  - a real-time system is stable if the system will meet the deadlines of its **most critical**, highest-priority tasks even if some less critical task deadlines are not always met

The heart of a real-time system is the <u>short-term task scheduler</u>.

In designing such a scheduler, fairness and minimizing average response time are <u>not important</u>.

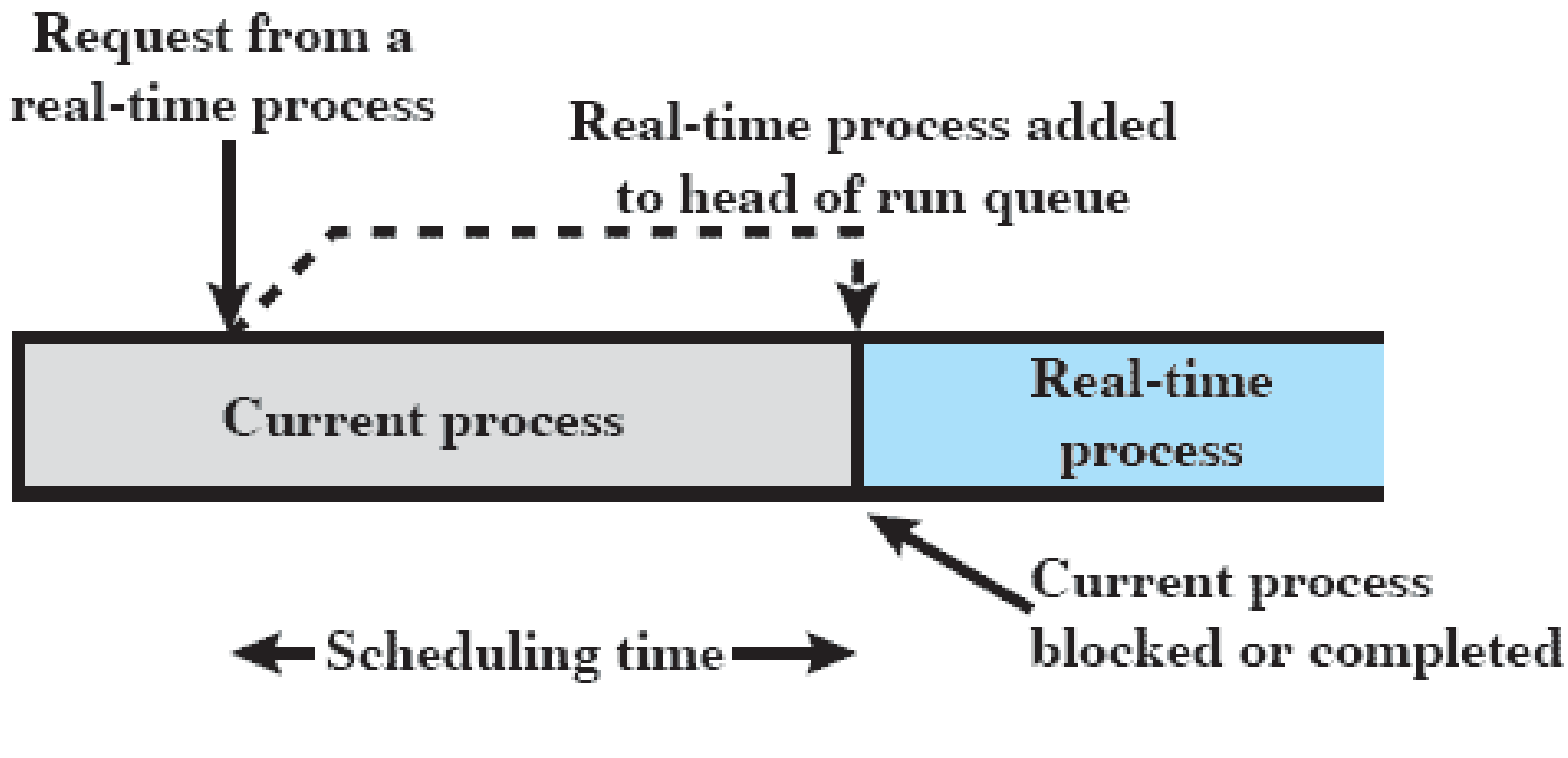
What is important: is that all hard real-time tasks complete (or start) by their deadline and that as many as possible soft real-time tasks also complete (or start) by their deadline.

Scheduling of Real-Time Process



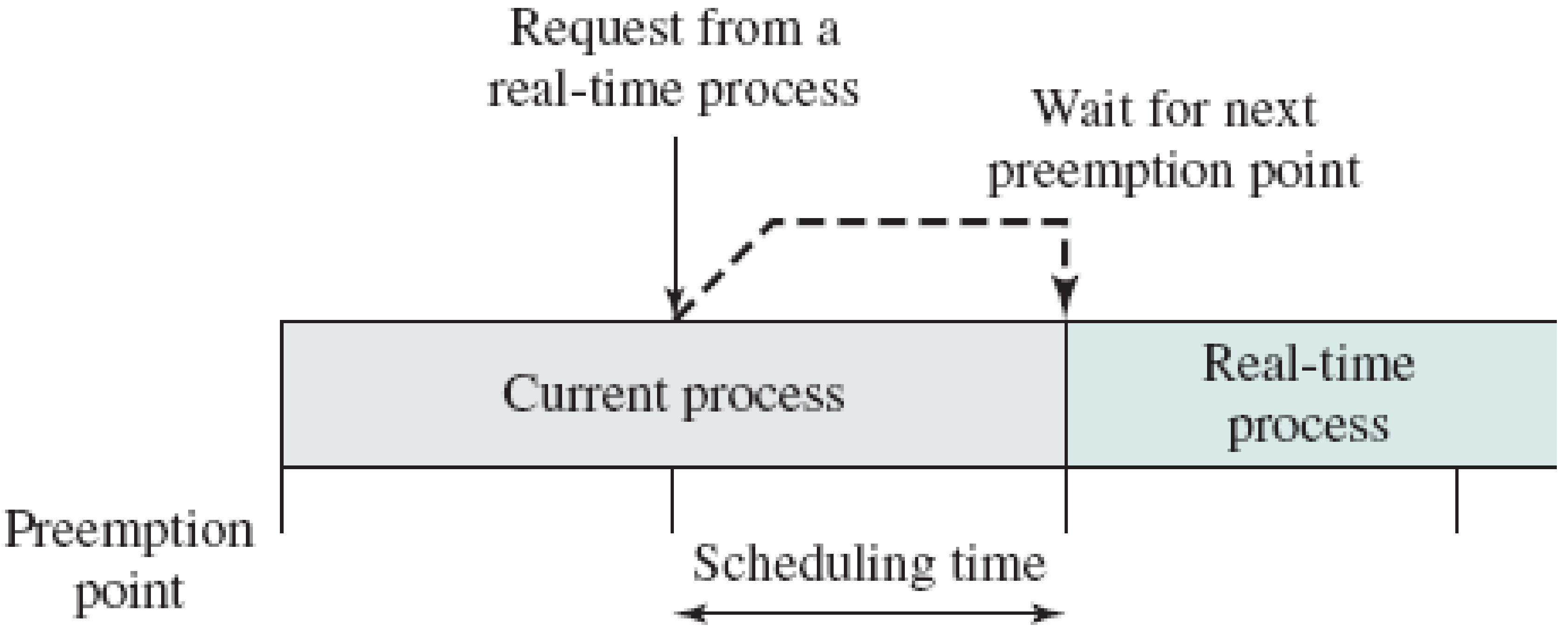
(a) Round-robin preemptive scheduler

In this case, the scheduling time will generally be unacceptable for real-time applications.



(b) Priority-Driven Nonpreemptive Scheduler

This could lead to a delay of several seconds if a slow, low-priority task were executing at a critical time.
<u>Again, this approach is not acceptable.</u>



(c) Priority-driven preemptive scheduler on preemption points

A more promising approach is to combine priorities with clock-based interrupts. Preemption points occur at regular intervals. When a preemption point occurs, the currently running task is preempted if a higher-priority task is waiting.

This would include the preemption of tasks that are part of the operating system kernel.

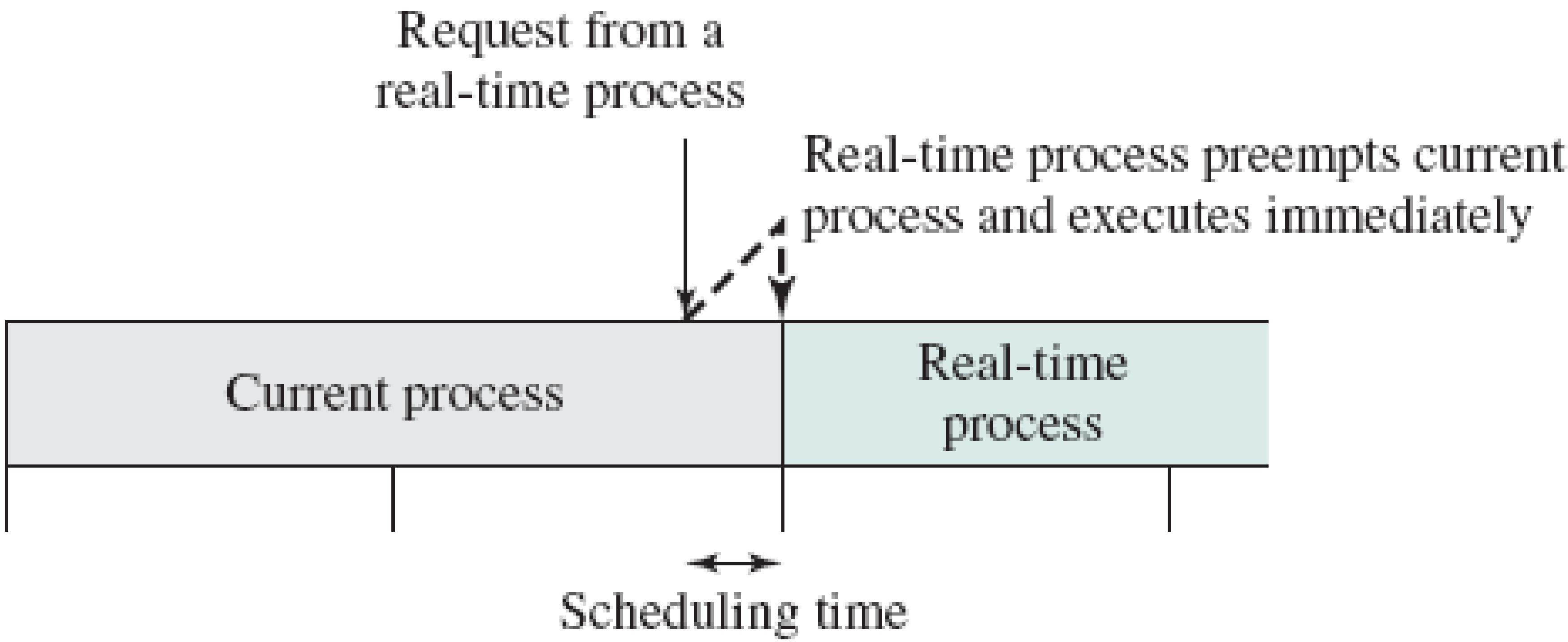
Such a delay may be on the order of several milliseconds (Figure 10.4c).

While this last approach may be adequate for <u>some real-time </u>applications, it will not be enough for more demanding applications.

In those cases, the approach that has been taken is sometimes referred to as <u>immediate preemption.</u>

In this case, the operating system responds to an interrupt almost immediately, unless the system is in a critical-code lockout section.

Scheduling delays for a real-time task can then be reduced to <u>100 μs or less</u>.



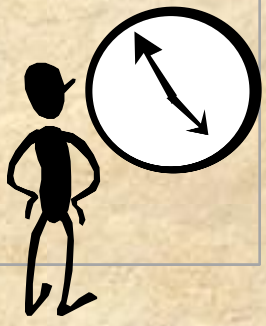
(d) Immediate preemptive scheduler

Real-Time Scheduling

<u>**Scheduling approaches depend on:**</u>

•whether a system performs <u>schedulability analysis</u>
(if it does, whether it is done statically or dynamically)

•whether the result of the analysis itself produces a <u>scheduler plan</u> according to which tasks are dispatched at run time



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

Classes of Real-Time Scheduling Algorithms

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 <u>tries to meet all deadlines and aborts any started process whose deadline is missed</u>

Deadline Scheduling

- Real-time operating systems are designed with the objective <u>**of starting real-time tasks** </u>as <u>rapidly as possible </u>and emphasize <u>**rapid**</u> interrupt handling and task dispatching
- Real-time applications are generally not concerned with sheer speed but rather with <u>completing</u> (or starting<u>) **tasks at the most valuable times**</u>
- Priorities provide a basic tool and do not capture the requirement of completion (or initiation) at the most valuable time