SNS COLLEGE OF TECHNOLOGY

Coimbatore-35.

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

COURSE NAME : 19CSB201 – OPERATING SYSTEMS

II YEAR/ IV SEMESTER

UNIT – II Process Scheduling And Synchronization

Topic: CPU Scheduling : Real-Time CPU Scheduling

Real-Time CPU Scheduling

- **Soft real-time systems** provide no guarantee as to **when a critical real-time process will be scheduled**. They guarantee only that the process will be given preference over noncritical processes.
- **Hard real-time systems** have stricter requirements. A task must be **serviced by its deadline**; service after the deadline has expired is the same as no service at all.

Minimizing Latency

- **Event latency -** the amount of time that elapses from when an event occurs to when it is serviced.

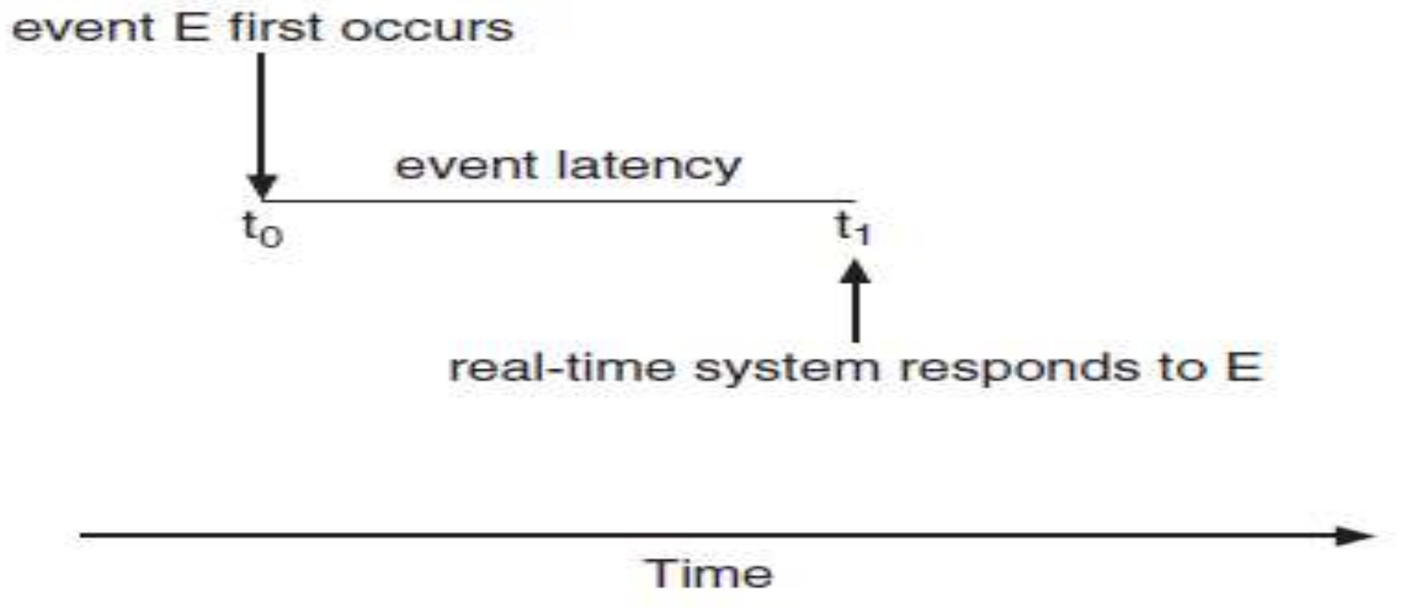

Figure 6.12 Event latency.

Two types of latencies affect the performance of real-time systems:

1. Interrupt latency

2. Dispatch latency

- **Interrupt latency** refers to the period of time from the arrival of an interrupt at the CPU to the start of the routine that services the interrupt.
- The amount of time required for the scheduling dispatcher to stop one process and start another is known as **dispatch latency**.

Interrupt latency

- When an interrupt occurs, the operating system must first complete the instruction it is executing and determine the type of interrupt that occurred. It must then save the state of the current process before servicing the interrupt using the specific interrupt service routine (ISR). The total time required to perform these tasks is the interrupt latency.

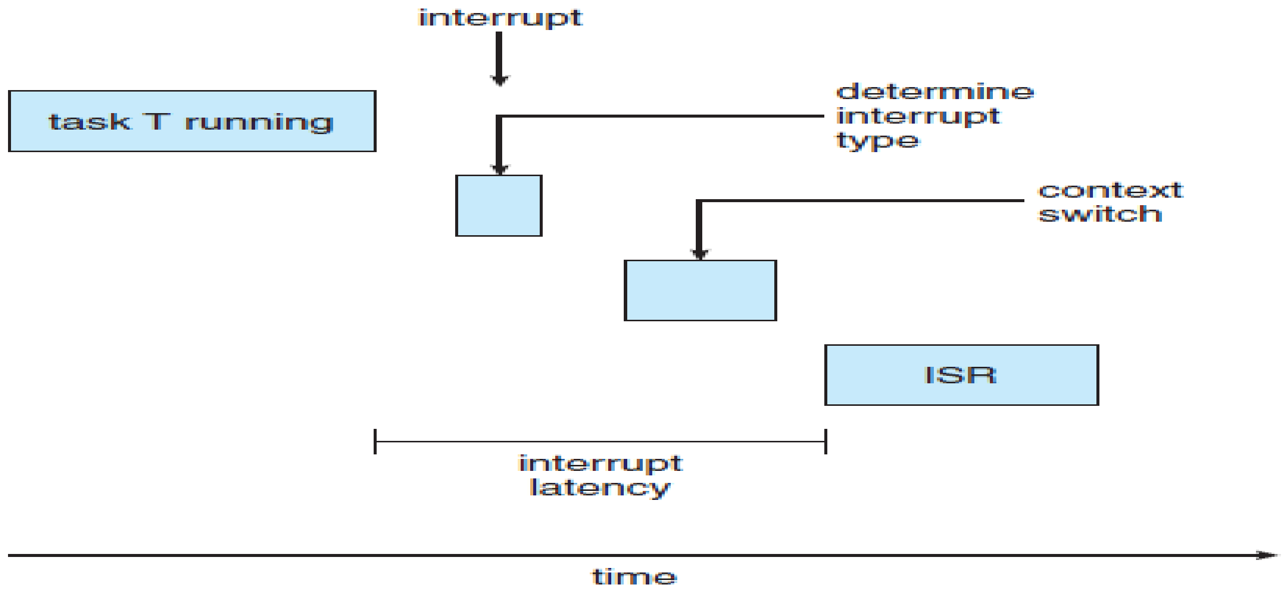

Figure 6.13 Interrupt latency.

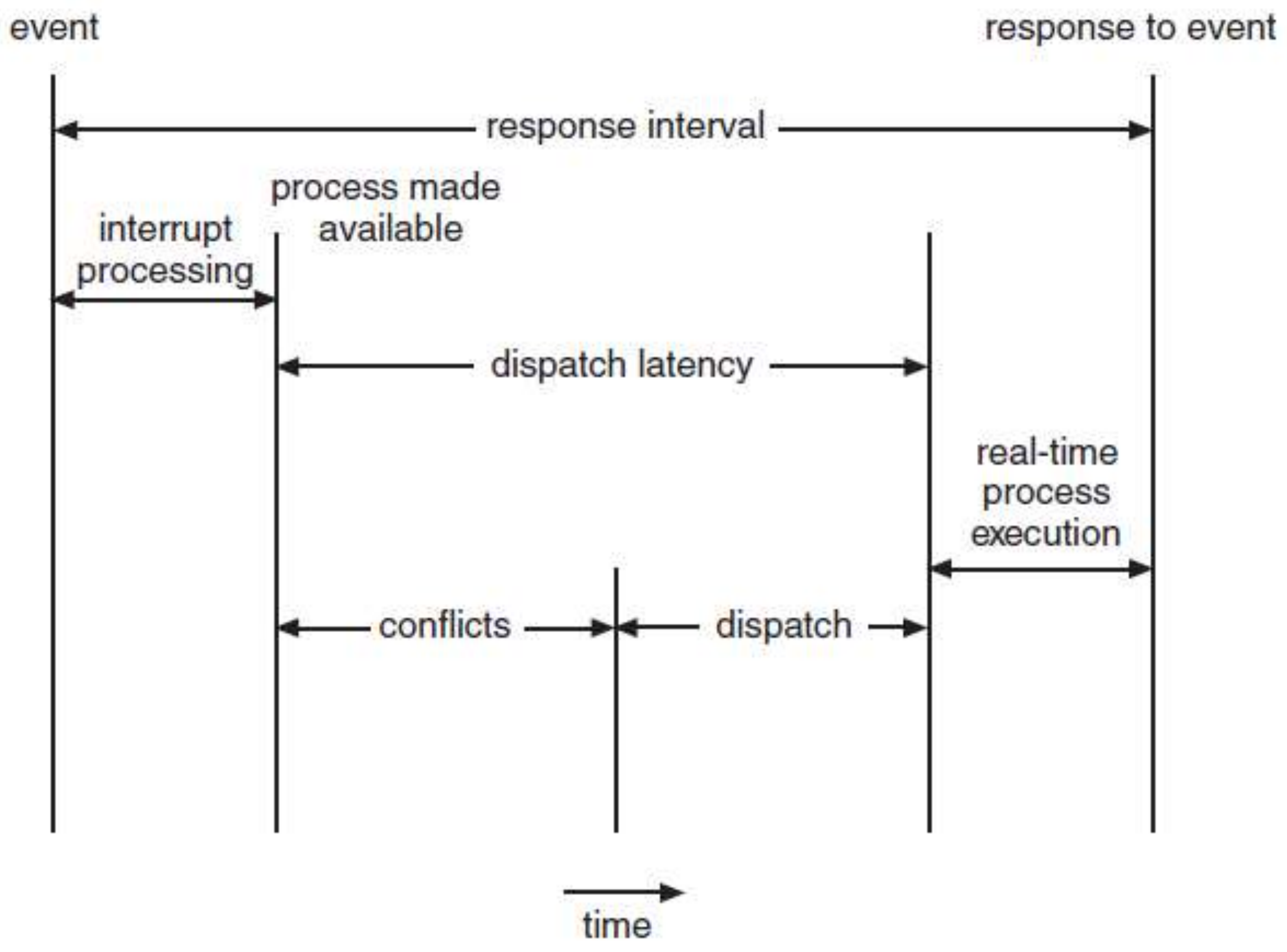

Figure 6.14 Dispatch latency.

Dispatch latency

The most effective technique for keeping dispatch latency low is to provide preemptive kernels.

The **conflict phase** of dispatch latency has two components:

1. Preemption of any process running in the kernel

2. Release by low-priority processes of resources needed by a high-priority process

Priority-Based Scheduling

- the processes are considered **periodic**
- The **rate** of a periodic task is 1/*p*.

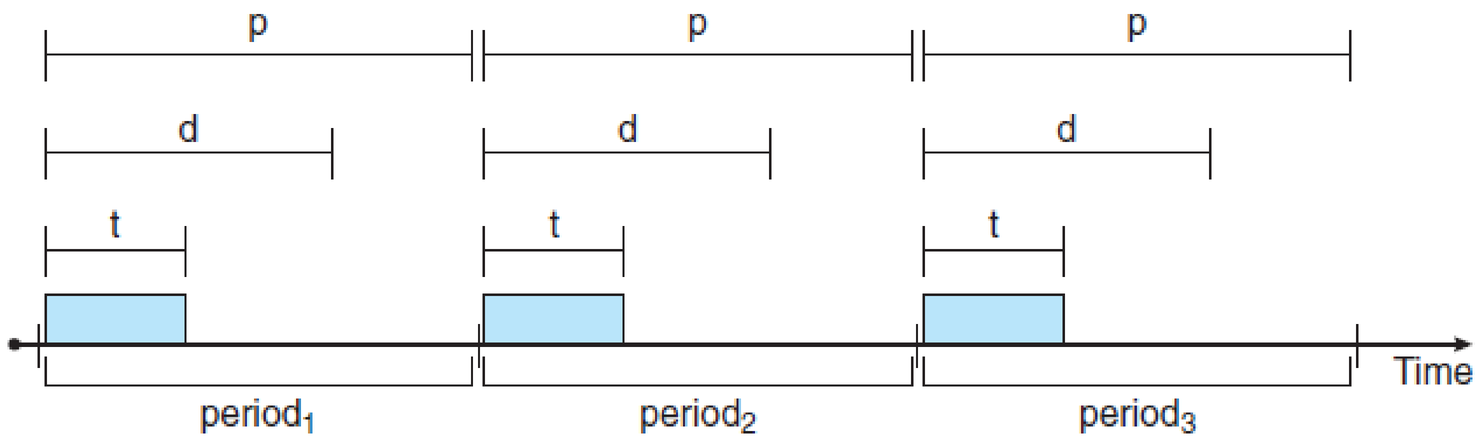

Figure 6.15 Periodic task.

- Then, using a technique known as an **admission-control** algorithm, the scheduler does one of two things. It either admits the process, guaranteeing that the process will complete on time, or rejects the request as impossible if it cannot guarantee that the task will be serviced by its deadline.

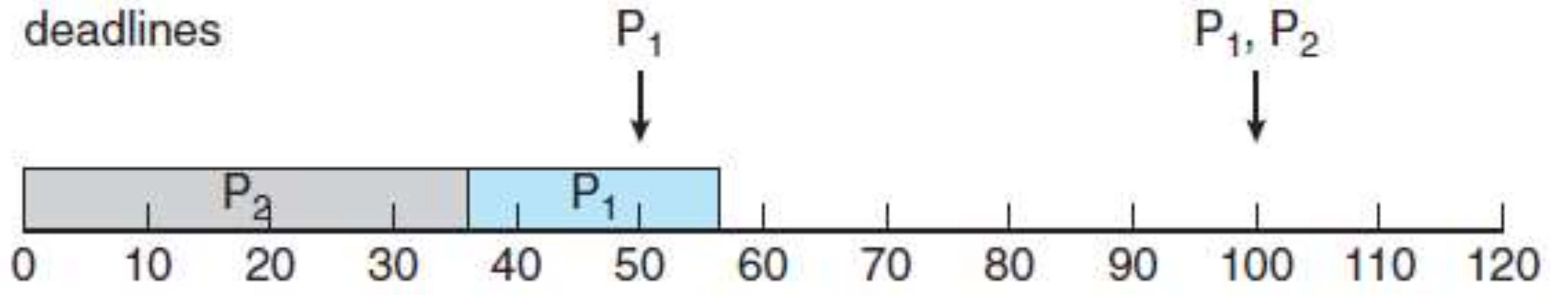

Figure 6.16 Scheduling of tasks when P_2 has a higher priority than P_1.

Rate-Monotonic Scheduling

- The **rate-monotonic** scheduling algorithm schedules periodic tasks using a static priority policy with preemption.
- If a lower-priority process is running and a higher-priority process becomes available to run, it will preempt the lower-priority process.Uponentering the system, each periodic task is assigned a priority inversely based on its period. The shorter the period, the higher the priority; the longer the period, the lower the priority. The rationale behind this policy is to assign a higher priority to tasks that require the CPU more often

•

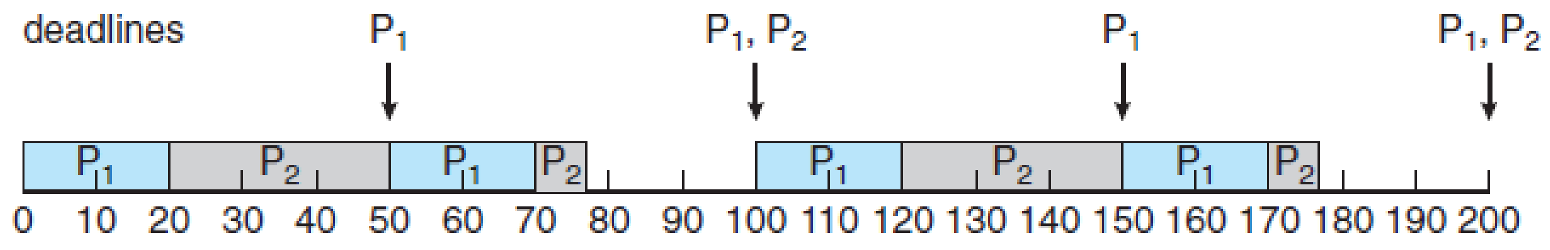

Figure 6.17 Rate-monotonic scheduling.

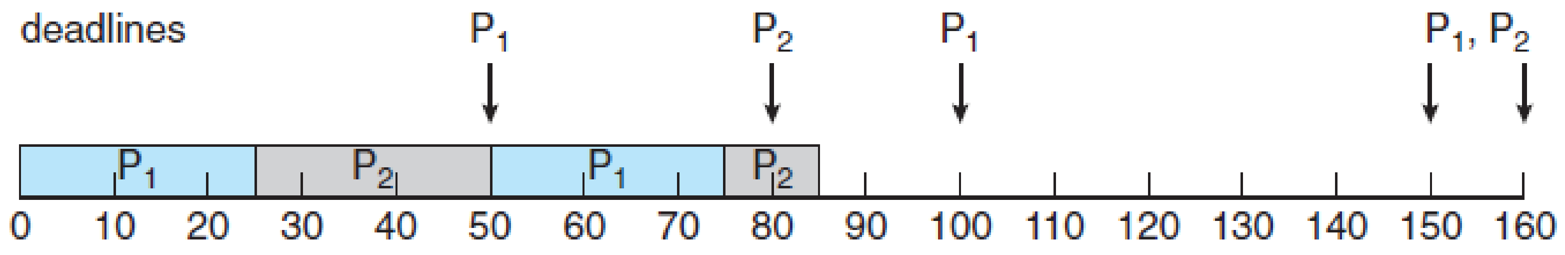

Figure 6.18 Missing deadlines with rate-monotonic scheduling.

Earliest-Deadline-First Scheduling

- **Earliest-deadline-first (EDF)** scheduling dynamically assigns priorities according to deadline. The earlier the deadline, the higher the priority; the later the deadline, the lower the priority.

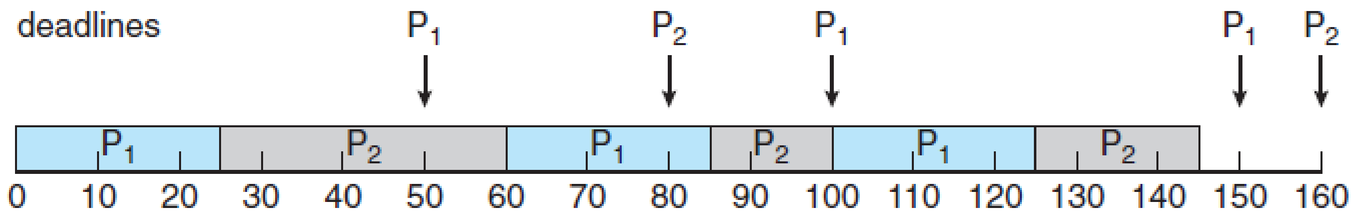

Figure 6.19 Earliest-deadline-first scheduling.

Proportional Share Scheduling

- **Proportional share** schedulers operate by allocating *T* shares among all applications. An application can receive *N* shares of time, thus ensuring that the application will have *N*/*T* of the total processor time.

POSIX Real-Time Scheduling

The POSIX standard also provides extensions for real-time computing—POSIX.1b.

Here, we cover some of the POSIX API related to scheduling real-time threads.

POSIX defines two scheduling classes for real-time threads:

- SCHED FIFO
- SCHED RR

The POSIX API specifies the following two functions for getting and setting the scheduling policy:

- pthread attr getsched policy(pthread attr t *attr, int *policy)
- pthread attr setsched policy(pthread attr t *attr, int policy)

REFERENCES

TEXT BOOKS:

T1 Silberschatz, Galvin, and Gagne, “Operating System Concepts”, Ninth Edition, Wiley India Pvt Ltd, 2009.)

T2. Andrew S. Tanenbaum, “Modern Operating Systems”, Fourth Edition, Pearson Education, 2010

REFERENCES:

R1 Gary Nutt, “Operating Systems”, Third Edition, Pearson Education, 2004.

R2 Harvey M. Deitel, “Operating Systems”, Third Edition, Pearson Education, 2004.

R3 Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, “Operating System Concepts”, 9th Edition, John Wiley and Sons Inc., 2012.

R4. William Stallings, “Operating Systems – Internals and Design Principles”, 7th Edition, Prentice Hall, 2011

THANK
YOU!