UTN FRD – Sistemas Operativos
Unidad IV Planificación de Procesos

Unidad IV Planificación Uniprocesador

Scheduling

- An OS must allocate resources amongst competing processes.
- The resource provided by a processor is execution time
  - The resource is allocated by means of a schedule

Overall Aim of Scheduling

- The aim of processor scheduling is to assign processes to be executed by the processor over time,
  - in a way that meets system objectives, such as response time, throughput, and processor efficiency.

Scheduling Objectives

- The scheduling function should
  - Share time ***fairly*** among processes
  - Prevent starvation of a process
  - Use the processor efficiently
  - Have low overhead
  - Prioritise processes when necessary (e.g. real time deadlines)

Types of Scheduling

Table 9.1 Types of Scheduling

Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device

Long-Term Scheduling

- Determines which programs are admitted to the system for processing
  - May be first-come-first-served
  - Or according to criteria such as priority, I/O requirements or expected execution time
- Controls the degree of multiprogramming
- More processes, smaller percentage of time each process is executed

Medium-Term Scheduling

- Part of the swapping function
- Swapping-in decisions are based on the need to manage the degree of multiprogramming

Short-Term Scheduling

- Known as the dispatcher
- Executes most frequently
- Invoked when an event occurs
  - Clock interrupts
  - I/O interrupts
  - Operating system calls
  - Signals

Aim of Short Term Scheduling

- Main objective is to allocate processor time to optimize certain aspects of system behaviour.
- A set of criteria is needed to evaluate the scheduling policy.

Short-Term Scheduling Criteria: User vs System

- We can differentiate between user and system criteria
- User-oriented
  - Response Time
    - Elapsed time between the submission of a request until there is output.
- System-oriented
  - Effective and efficient utilization of the processor

Short-Term Scheduling Criteria: Performance

- We could differentiate between performance related criteria, and those unrelated to performance
- Performance-related
  - Quantitative, easily measured
  - E.g. response time and throughput
- Non-performance related
  - Qualitative
  - Hard to measure

Interdependent Scheduling Criteria

User Oriented, Performance Related

Turnaround time This is the interval of time between the submission of a process and its completion. Includes actual execution time plus time spent waiting for resources, including the processor. This is an appropriate measure for a batch job.

Response time For an interactive process, this is the time from the submission of a request until the response begins to be received. Often a process can begin producing some output to the user while continuing to process the request. Thus, this is a better measure than turnaround time from the user's point of view. The scheduling discipline should attempt to achieve low response time and to maximize the number of interactive users receiving acceptable response time.

Deadlines When process completion deadlines can be specified, the scheduling discipline should subordinate other goals to that of maximizing the percentage of deadlines met.

User Oriented, Other

Predictability A given job should run in about the same amount of time and at about the same cost regardless of the load on the system. A wide variation in response time or turnaround time is distracting to users. It may signal a wide swing in system workloads or the need for system tuning to cure instabilities.

Interdependent Scheduling Criteria cont.

System Oriented, Performance Related

Throughput The scheduling policy should attempt to maximize the number of processes completed per unit of time. This is a measure of how much work is being performed. This clearly depends on the average length of a process but is also influenced by the scheduling policy, which may affect utilization.

Processor utilization This is the percentage of time that the processor is busy. For an expensive shared system, this is a significant criterion. In single-user systems and in some other systems, such as real-time systems, this criterion is less important than some of the others.

System Oriented, Other

Fairness In the absence of guidance from the user or other system-supplied guidance, processes should be treated the same, and no process should suffer starvation.

Enforcing priorities When processes are assigned priorities, the scheduling policy should favor higher-priority processes.

Balancing resources The scheduling policy should keep the resources of the system busy. Processes that will underutilize stressed resources should be favored. This criterion also involves medium-term and long-term scheduling.

Priorities

- Scheduler will always choose a process of higher priority over one of lower priority
- Have multiple ready queues to represent each level of priority

Priority Queuing

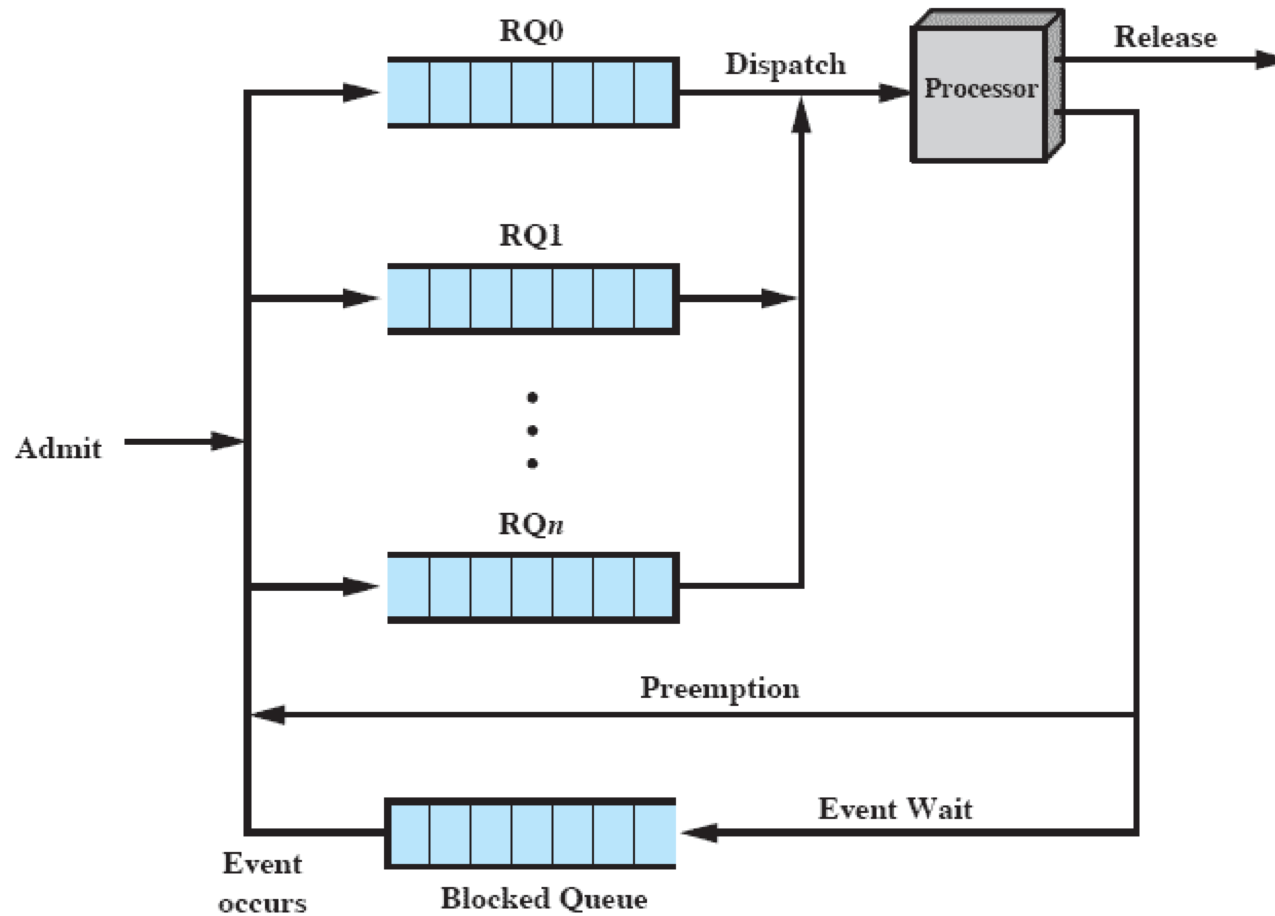


Figure 9.4 Priority Queuing

Starvation

- Problem:
  - Lower-priority may suffer starvation if there is a steady supply of high priority processes.
- Solution
  - Allow a process to change its priority based on its age or execution history

Alternative Scheduling Policies

Table 9.3 Characteristics of Various Scheduling Policies

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s − e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

Selection Function

- Determines which process is selected for execution
- If based on execution characteristics then important quantities are:
  - ***w*** = time spent in system so far, waiting
  - ***e*** = time spent in execution so far
  - ***s*** = total service time required by the process, including *e*;

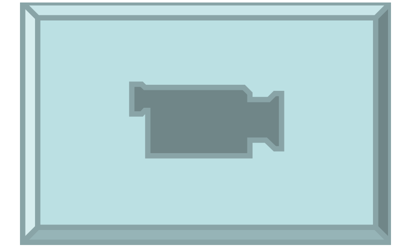
Decision Mode

- Specifies the instants in time at which the selection function is exercised.
- Two categories:
  - Nonpreemptive
  - Preemptive

Nonpreemptive vs Premeptive

- Non-preemptive
  - Once a process is in the running state, it will continue until it terminates or blocks itself for I/O
- Preemptive
  - Currently running process may be interrupted and moved to ready state by the OS
  - Preemption may occur when new process arrives, on an interrupt, or periodically.

Process Scheduling Example



- Example set of processes, consider each a batch job

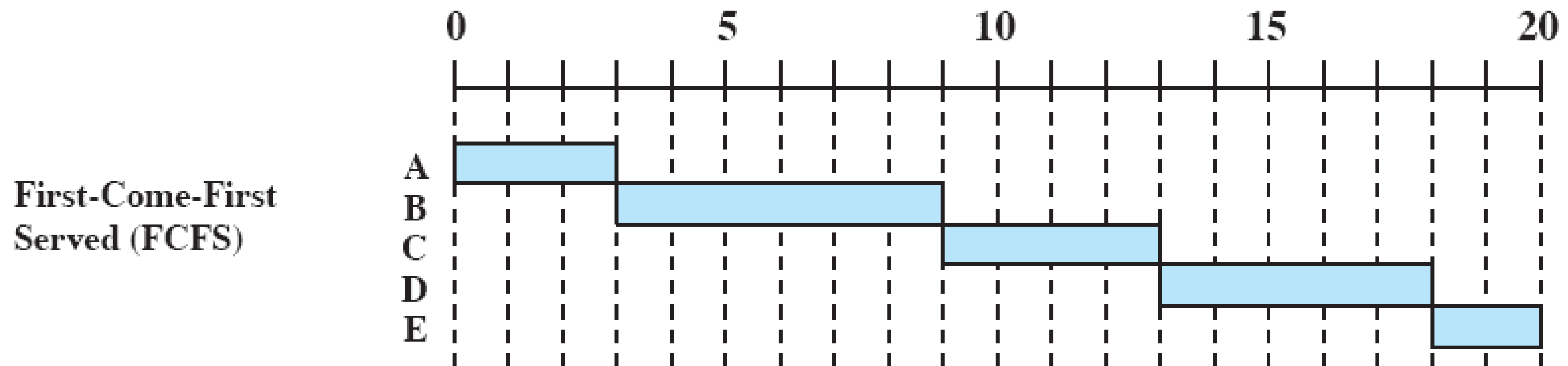
Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

  – Service time represents total execution time

First-Come-First-Served

- Each process joins the Ready queue
- When the current process ceases to execute, the longest process in the Ready queue is selected



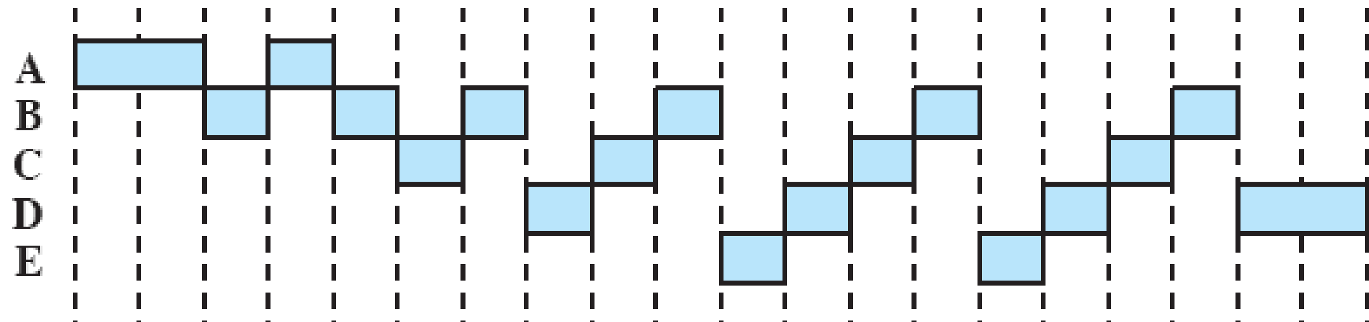
First-Come-
First-Served

- A short process may have to wait a very long time before it can execute
- Favors CPU-bound processes
  - I/O processes have to wait until CPU-bound process completes

Round Robin

- Uses preemption based on a clock
  - also known as time slicing, because each process is given a slice of time before being preempted.

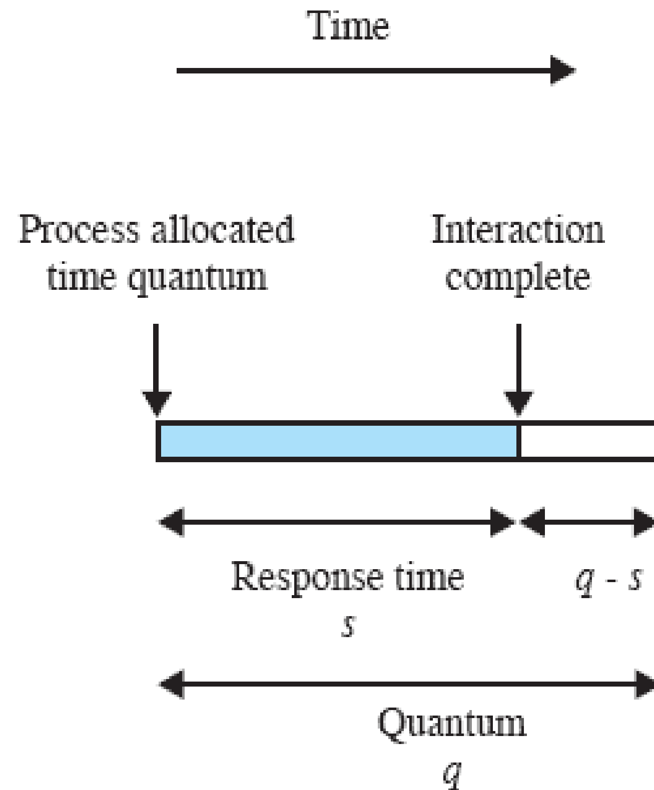
Round-Robin (RR), $q = 1$



Round Robin

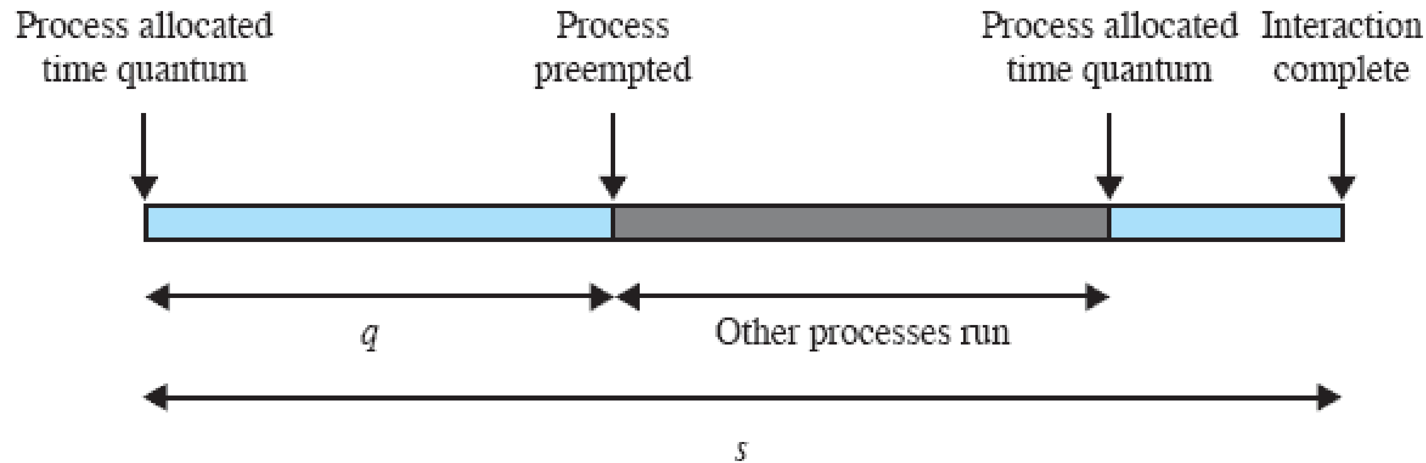
- Clock interrupt is generated at periodic intervals
- When an interrupt occurs, the currently running process is placed in the ready queue
  - Next ready job is selected

Effect of Size of Preemption Time Quantum



(a) Time quantum greater than typical interaction

Effect of Size of Preemption Time Quantum



(b) Time quantum less than typical interaction

Figure 9.6 Effect of Size of Preemption Time Quantum

‘Virtual Round Robin’

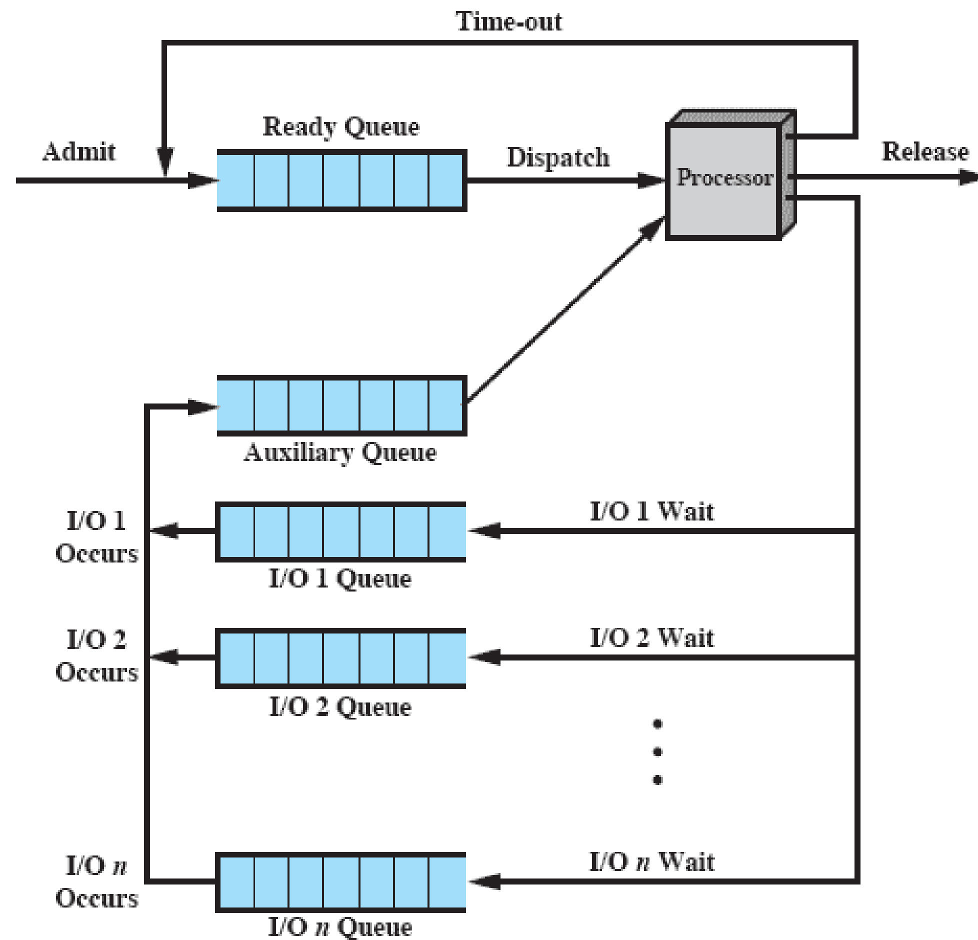
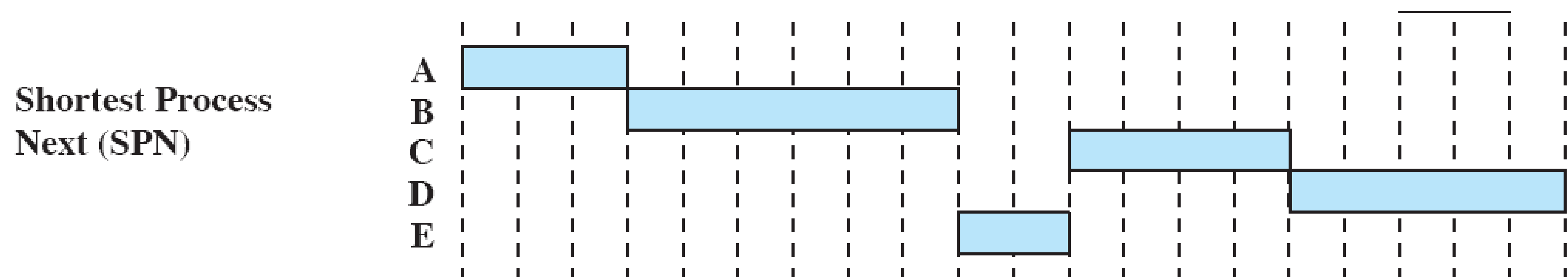


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

Shortest Process Next

- Nonpreemptive policy
- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes

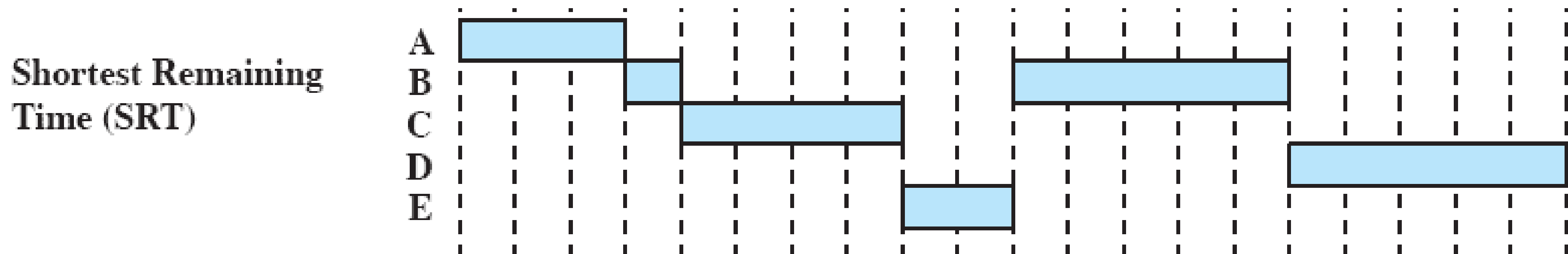


Shortest Process Next

- Predictability of longer processes is reduced
- If estimated time for process not correct, the operating system may abort it
- Possibility of starvation for longer processes

Shortest Remaining Time

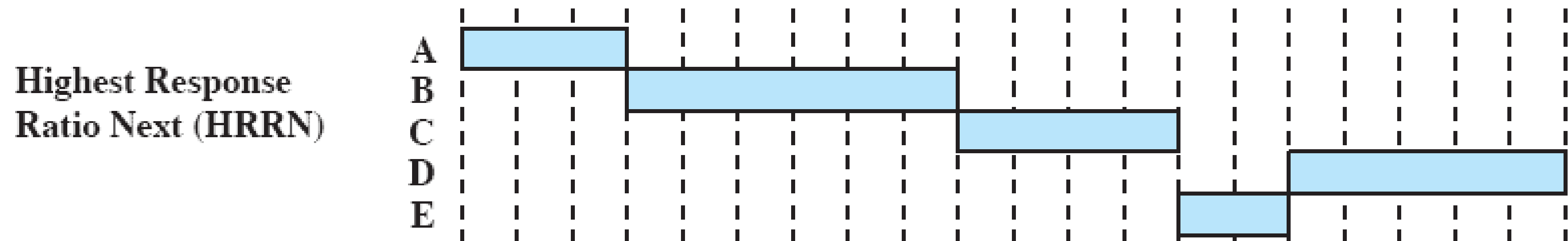
- Preemptive version of shortest process next policy
- Must estimate processing time and choose the shortest



Highest Response Ratio Next

- Choose next process with the greatest ratio

$$Ratio = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time}$$



Feedback Scheduling

- Penalize jobs that have been running longer
- Don't know remaining time process needs to execute

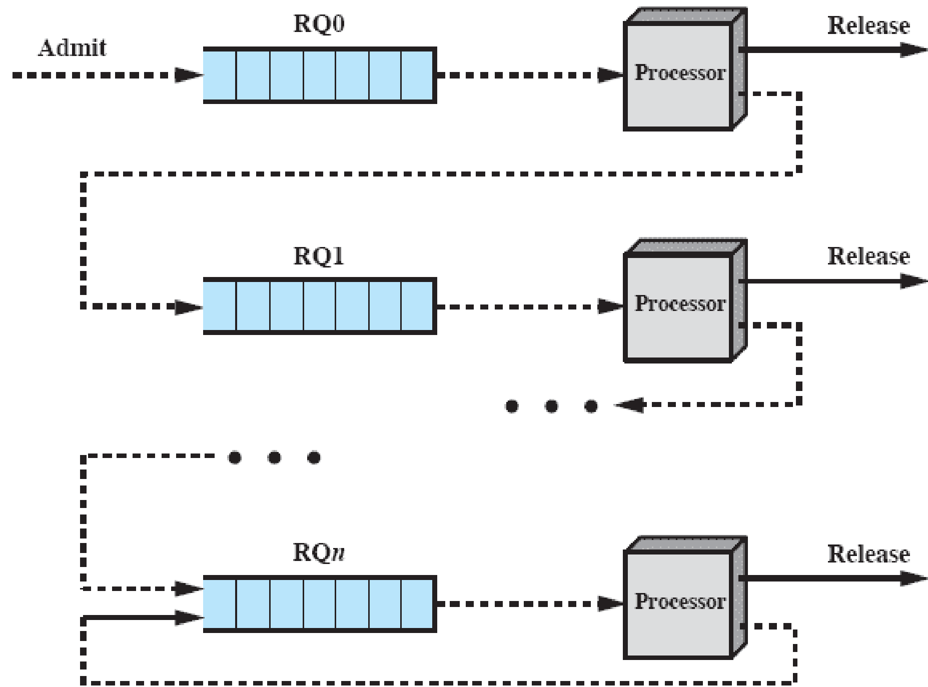
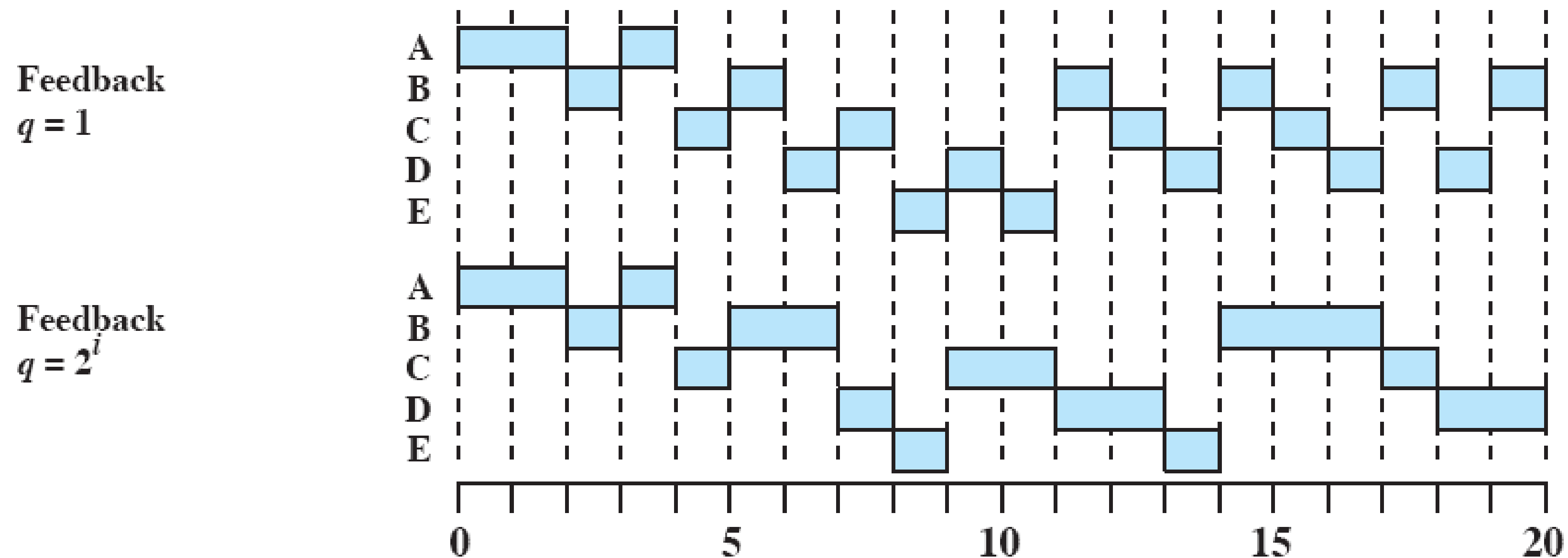


Figure 9.10 Feedback Scheduling

Feedback Performance

- Variations exist, simple version pre-empts periodically, similar to round robin
  - But can lead to starvation



Performance Comparison

- Any scheduling discipline that chooses the next item to be served independent of service time obeys the relationship:

$$\frac{T_r}{T_s} = \frac{1}{1 - \rho}$$

where

T_r = turnaround time or residence time; total time in system, waiting plus execution

T_s = average service time; average time spent in Running state

ρ = processor utilization

Unidad IV Planificación Multiprocesador

Classifications of Multiprocessor Systems

- Loosely coupled processors,
  - Each has their memory & I/O channels
- Functionally specialized processors
  - Controlled by a master processor
  - Such as I/O processor
- Tightly coupled multiprocessing
  - Processors share main memory
  - Controlled by operating system

Granularity

- Or frequency of synchronization, between processes in a system.
- Five categories, differing in granularity:
  - Independent Parallelism
  - Coarse Parallelism
  - Very Coarse-Grained Parallelism
  - Medium-Grained Parallelism
  - Fine-Grained Parallelism

Independent Parallelism

- No explicit synchronization among processes
- Separate application or job
- Example is time-sharing system

Coarse and Very Coarse-Grained Parallelism

- Synchronization among processes at a very gross level
- Good for concurrent processes running on a multiprogrammed uniprocessor
  - Can by supported on a multiprocessor with little change

Medium-Grained Parallelism

- Single application is a collection of threads
- Threads usually interact frequently, affecting the performance of the entire application

Fine-Grained Parallelism

- Highly parallel applications
- Specialized and fragmented area

Synchronization Granularity and Processes

Table 10.1 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

Scheduling Design Issues

- Scheduling on a multiprocessor involves three interrelated issues:
  - Assignment of processes to processors
  - Use of multiprogramming on individual processors
  - Actual dispatching of a process
- 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 process to processors on demand.
  - Should the assignment be static or dynamic though?
- Dynamic Assignment
  - threads are moved for a queue for one processor to a queue for another processor;

Static Assignment

- Permanently assign process to a processor
  - Dedicate short-term queue for each processor
  - Less overhead
  - Allows the use of ‘group’ or ‘gang’ scheduling (see later)
- But may leave a processor idle, while others have a backlog
  - Solution: use a common queue

Assignment of Processes to Processors

- Both dynamic and static methods require some way of assigning a process to a processor
- Two methods:
  - Master/Slave
  - Peer
- There are of course a spectrum of approaches between these two extremes.

Master / Slave Architecture

- Key kernel functions always run on a particular processor
- Master is responsible for scheduling
- Slave sends service request to the master
- 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
- Complicates the operating system
  - Make sure two processors do not choose the same process

Process Scheduling

- Usually processes are not dedicated to processors
- A single queue is used for all processes
- Or multiple queues are used for priorities
  - All queues feed to the common pool of processors

Thread Scheduling

- Threads execute separate from the rest of the process
- An application can be a set of threads that cooperate and execute concurrently in the same address space
- Dramatic gains in performance are possible in multi-processor systems
  - Compared to running in uniprocessor systems

Approaches to Thread Scheduling

- Many proposals exist but four general approaches stand out:
  - Load Sharing
  - Gang Scheduling
  - Dedicated processor assignment
  - Dynamic scheduling

Load Sharing

- Processes are not assigned to a particular processor
- 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 Chapter 9.

Disadvantages of Load Sharing

- Central queue needs mutual exclusion
  - Can lead to bottlenecks
- Preemptive threads are unlikely resume execution on the same processor
- If all threads are in the global queue, all threads of a program will not gain access to the processors at the same time

Gang Scheduling

- A set of related threads is scheduled to run on a set of processors at the same time
- Parallel execution of closely related processes may reduce overhead such as process switching and synchronization blocking.

Dedicated Processor Assignment

- When application is scheduled, its threads are assigned to a processor
- Some processors may be idle
  - No multiprogramming of processors
- ***But***
  - In *highly* parallel systems processor utilization is less important than effectiveness
  - Avoiding process switching speeds up programs

Dynamic Scheduling

- Number of threads in a process are altered dynamically by the application
  - This allows the OS to adjust the load to improve utilization

Unidad IV Planificación en Sistemas de Tiempo Real

Real-Time Scheduling

- 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 vs Soft

- “Hard “ real time task:
  - One that must meet a deadline
- “Soft” real time task
  - Has a deadline which is desirable but not mandatory

Periodic vs Aperiodic

- Periodic tasks
  - Are completed regularly, once per period *T* or *T* units apart
- Aperiodic tasks
  - have time constraints either for deadlines or start

Real-Time Systems

- Control of laboratory experiments
- Process control in industrial plants
- Robotics
- Air traffic control
- Telecommunications
- Military command and control systems

Characteristics of Real Time Systems

- Real time systems have requirements in five general areas:
  - Determinism
  - Responsiveness
  - User control
  - Reliability
  - Fail-soft operation

Determinism

- Operations are performed at fixed, predetermined times or within predetermined time intervals
- Concerned with how long the operating system delays before acknowledging an interrupt and there is sufficient capacity to handle all the requests within the required time

Responsiveness

- How long, after acknowledgment, it takes the operating system to service the interrupt
- Responsiveness includes:
  - Amount of time to begin execution of the interrupt
  - Amount of time to perform the interrupt
  - Effect of interrupt nesting

User control

- It is essential to allow the user fine-grained control over task priority.
- May allow user to specify things such as paging or process swapping
- Disks transfer algorithms to use
- Rights of processes

Characteristics

- Reliability
  - Degradation of performance may have catastrophic consequences
- Fail-soft operation
  - Ability of a system to fail in such a way as to preserve as much capability and data as possible
  - Stability is important – if all deadlines are impossible, critical deadlines still meet.

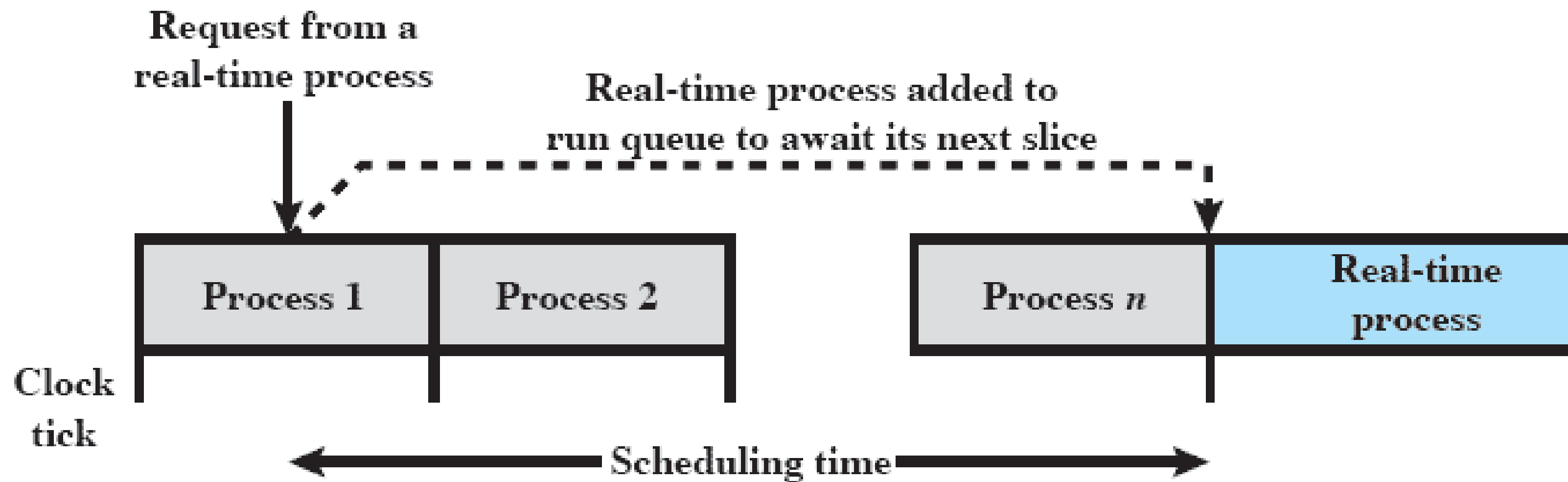
Features of Real-Time OS

- Fast process or thread switch
- Small size
- Ability to respond to external interrupts quickly
- Multitasking with interprocess communication tools such as semaphores, signals, and events

Features of Real-Time OS cont…

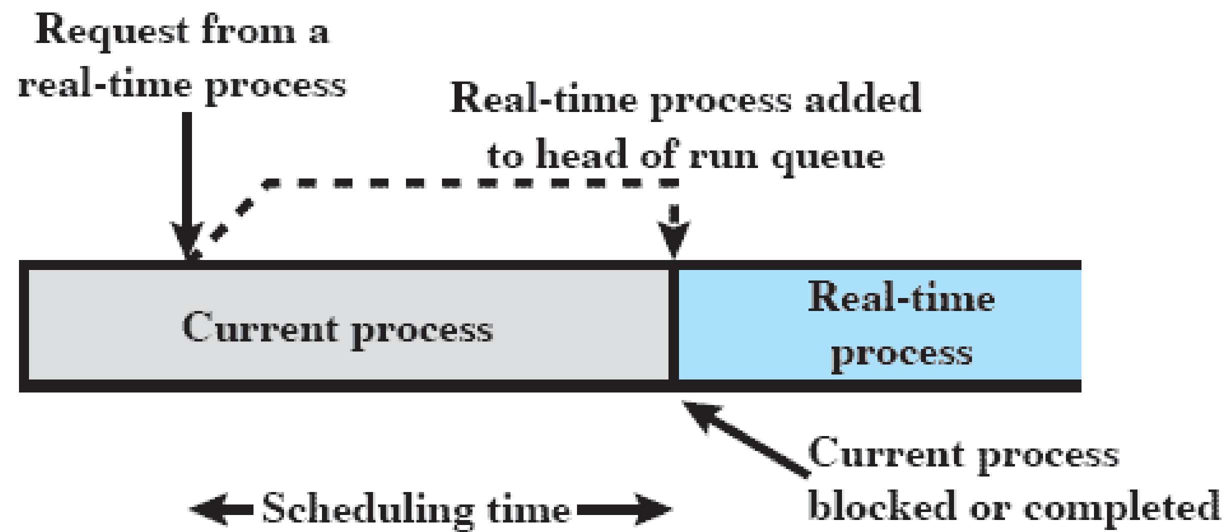
- Use of special sequential files that can accumulate data at a fast rate
- Preemptive scheduling base on priority
- Minimization of intervals during which interrupts are disabled
- Delay tasks for fixed amount of time
- Special alarms and timeouts

Round Robin scheduling unacceptable



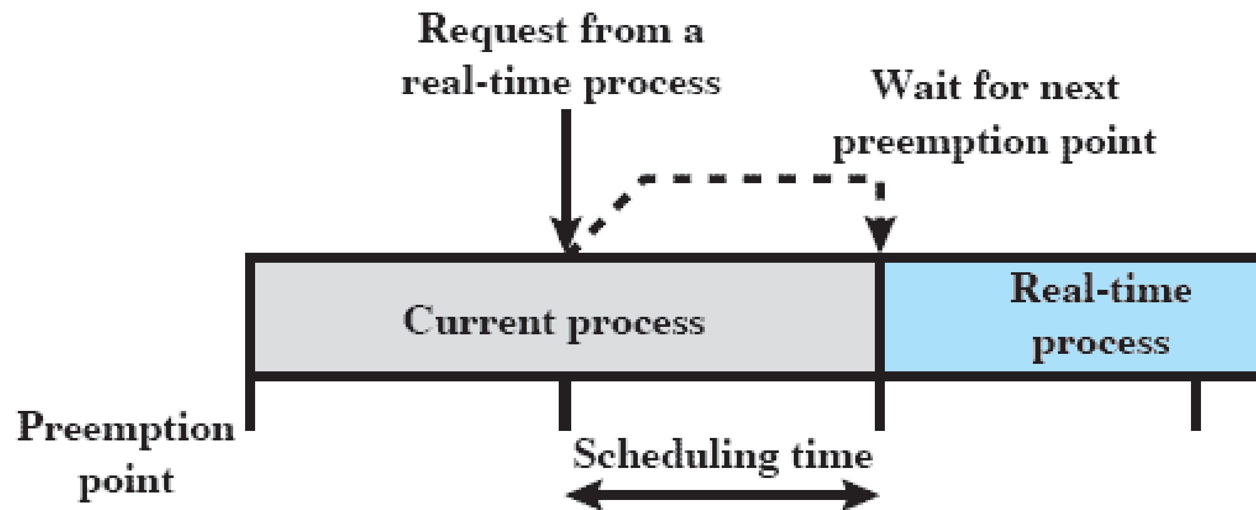
(a) Round-robin Preemptive Scheduler

Priority driven unacceptable



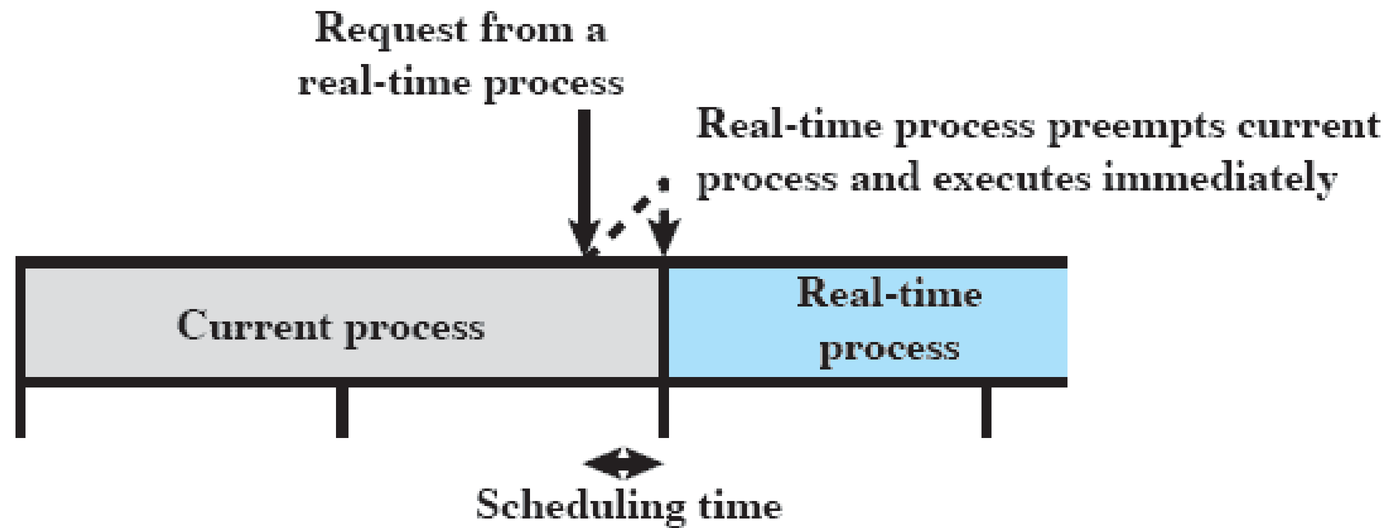
(b) Priority-Driven Nonpreemptive Scheduler

Combine priorities with clock-based interrupts



(c) Priority-Driven Preemptive Scheduler on Preemption Points

Immediate Preemption



(d) Immediate Preemptive Scheduler

Classes of Real-Time Scheduling Algorithms

- Static table-driven
  - Task execution determined at run time
- Static priority-driven preemptive
  - Traditional priority-driven scheduler is used
- Dynamic planning-based
  - Feasibility determined at run time
- Dynamic best effort
  - No feasibility analysis is performed

Deadline Scheduling

- Real-time applications are not concerned with speed but with completing tasks
- “Priorities” are a crude tool and may not capture the time-critical element of the tasks

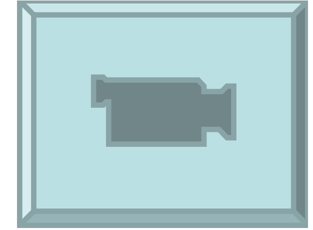
Deadline Scheduling

- Information used
  - Ready time
  - Starting deadline
  - Completion deadline
  - Processing time
  - Resource requirements
  - Priority
  - Subtask scheduler

Preemption

- When starting deadlines are specified, then a nonpreemptive scheduler makes sense.
- E.G. if task X is running and task Y is ready, there may be circumstances in which the only way to allow both X and Y to meet their completion deadlines is to preempt X, execute Y to completion, and then resume X to completion.

Rate Monotonic Scheduling



- Assigns priorities to tasks on the basis of their periods
- Highest-priority task is the one with the shortest period

Task Set

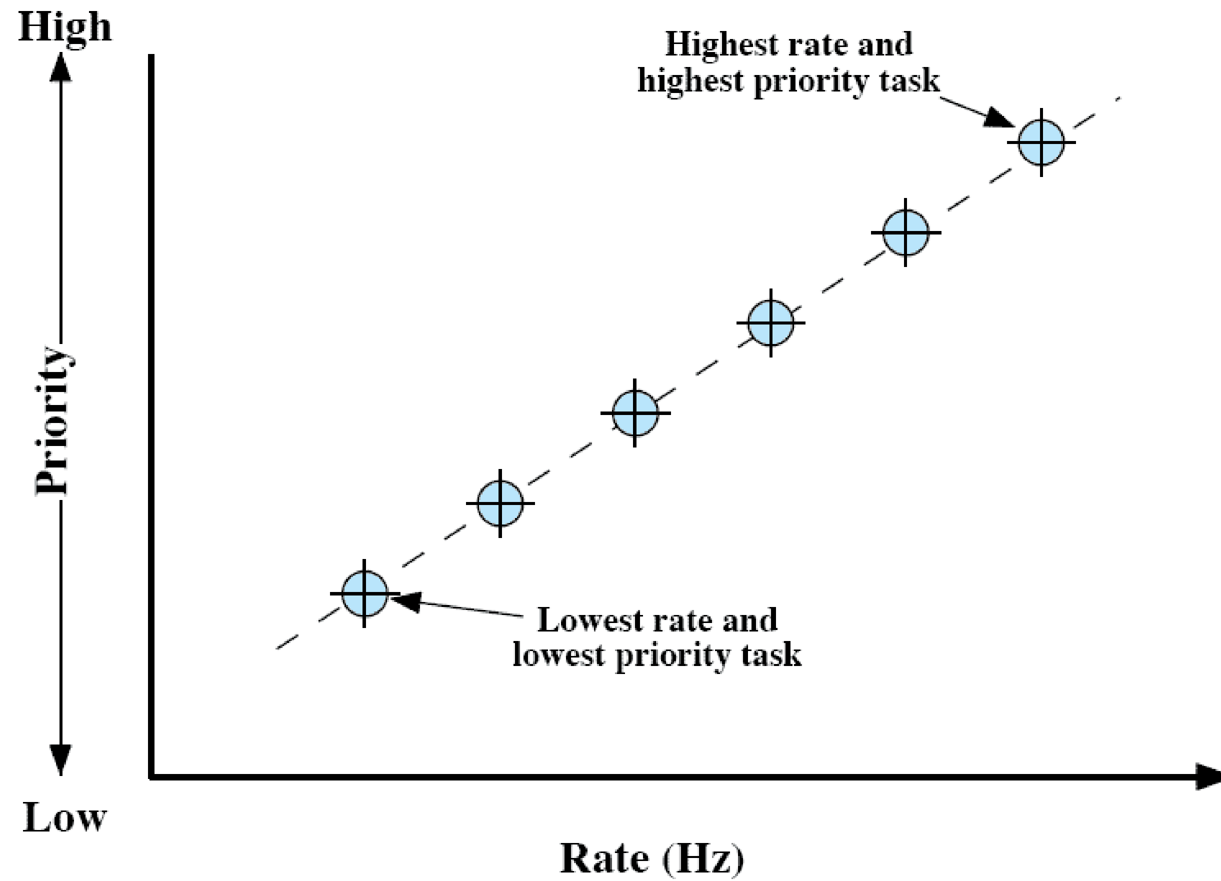


Figure 10.8 A Task Set with RMS [WARR91]

Periodic Task Timing Diagram

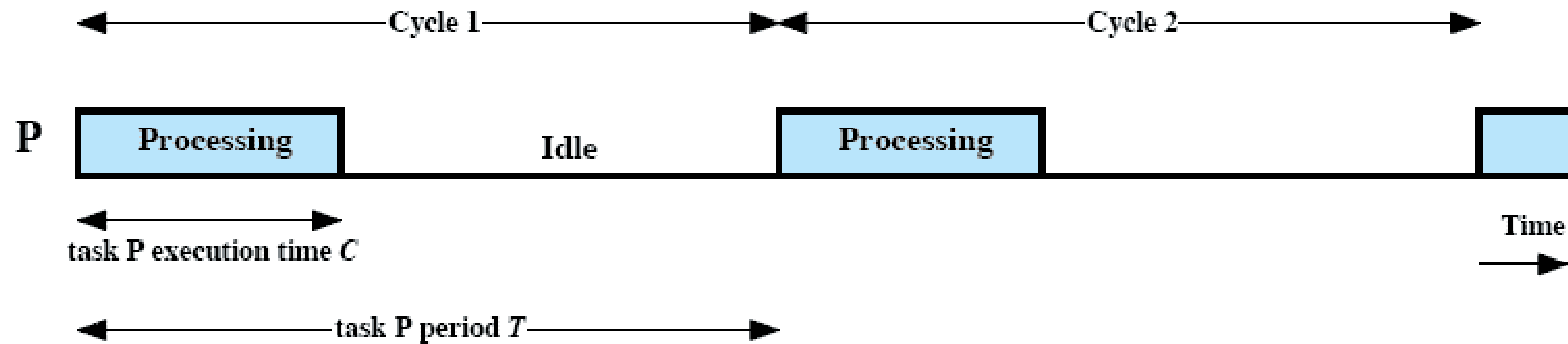


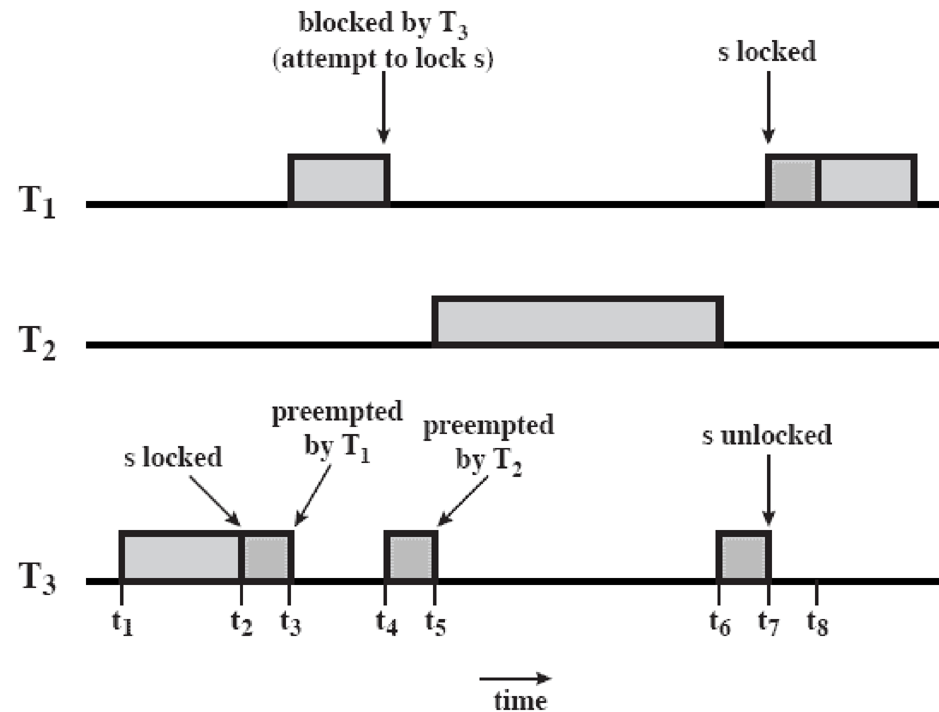
Figure 10.9 Periodic Task Timing Diagram

Priority Inversion

- Can occur in any priority-based preemptive scheduling scheme
- Occurs when circumstances within the system force a higher priority task to wait for a lower priority task

Unbounded Priority Inversion

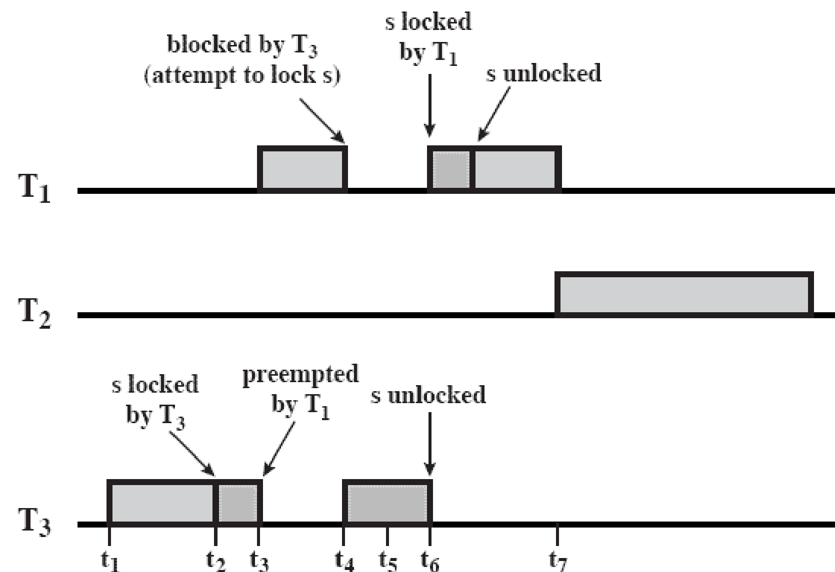
- Duration of a priority inversion depends on unpredictable actions of other unrelated tasks



(a) Unbounded priority inversion

Priority Inheritance

- Lower-priority task inherits the priority of any higher priority task pending on a resource they share



(b) Use of priority inheritance