Multiprocessor and Real-Time Scheduling

Chapter 10

Contents

- Multiprocessor scheduling
- Real-Time scheduling
  - real-time systems and real-time OS
  - real-time scheduling
- Linux scheduling
- Unix SVR4 scheduling
- Windows 2000 scheduling

Classifications of Multiprocessor

- Loosely coupled multiprocessor
  - each processor has its own memory and I/O channels
- Functionally specialized processors
  - such as I/O processor
  - controlled by a master processor
- Tightly coupled multiprocessing
  - processors share main memory
  - controlled by an operating system

Synchronization Granularity

- Frequency of synchronization between processes
- Degree of granularity
  - Independent Parallelism
  - Very Coarse Parallelism
  - Coarse Parallelism
  - Medium Parallelism
  - Fine-Grained Parallelism

Independent Parallelism

- No synchronization among processes
- Multiple unrelated processes
- Typical example in a time sharing system
  - one application is word processing
  - the other one is using a spreadsheet
- Average response time to the users will be less than that of the uniprocessor system

Very Coarse Parallelism

- Distributed processing across network nodes to form a single computing environment
- Good when there is infrequent interaction among processes
  - overhead of network would slow down communications

Coarse Parallelism

- Multiprocessing of concurrent processes in a multiprogramming environment
  - similar to running many processes on one processor except it is spread to more processors

Medium Parallelism

- Parallel processing or multitasking within a single application
  - single application is a collection of threads
  - threads in a process usually interact so frequently

Fine-Grained Parallelism

- Much more complex use of parallelism than is found in the use of threads
  - parallelism inherent in a single instruction stream
- Highly parallel applications

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	(N/A)

Design Issues of Multiprocessor Scheduling

- Assignment of processes to processors
- Use of multiprogramming on individual processors
- Actual dispatching of a process

Assignment of Processes to Processors

- Assign processes to processors on demand
  - could be static or dynamic
  - assigned to one processor from activation until its completion
- Means of assigning processes to processors
  - master/slave architecture
  - peer architecture

Assignment of Processes to Processors

- Master/slave architecture
  - Key kernel functions always run on a particular processor
    - Master is responsible for scheduling
  - Slaves only execute user programs
    - Slave sends service request to the master
  - Disadvantages
    - Failure of master brings down whole system
    - Master can become a performance bottleneck

Assignment of Processes to Processors

- Peer architecture
  - Operating system can execute on any processor
    - Each processor does self-scheduling
  - Complicates the operating system
    - must ensure that two processors do not choose the same process
    - need to resolve and synchronize competing claims to resources

Use of Multiprogramming on Individual Processors

- In the environment of coarse-grained or independent synchronization granularity, use of multiprogramming is natural
- For medium-grained applications running on a multiprocessor, situation is less clear
  - it is no longer paramount that every single processor be busy as much as possible
  - an application that consists of a number of threads may run poorly unless all of its threads are available to run simultaneously

Dispatching a Process

- Actual selection of a process to run
  - uniprocessor system
    - use sophisticated algorithms to improve performance
  - multiprocessor system
    - simpler approaches may be more effective with less overhead

Process Scheduling

- Traditional multiprocessor system
  - processes are not dedicated to processors
  - single queue for all processors
  - multiple queues are used for the case of using priorities
    - all queues feed to the common pool of processors

Multiprocessor Thread Scheduling

- An application can be a set of threads that cooperate and execute concurrently in the same address space
- Threads running on separate processors yields a dramatic gain in performance

Multiprocessor Thread Scheduling

- General approaches for scheduling
  - Load sharing
    - processes are not assigned to a particular processor
    - a global queue is maintained and each processor selects a thread from the queue
  - Gang scheduling
    - a set of related threads is scheduled to run on a set of processors at the same time
  - Dedicated processor assignment
    - threads are assigned to a specific processor
  - Dynamic scheduling
    - number of threads in a process can be altered during course of execution

Load Sharing

- A global queue of ready threads is maintained
  - load is distributed evenly across the processors
    - assures no processor is idle
  - no centralized scheduler required
    - when a processor is available, scheduling routine is run on that processor

Disadvantages of Load Sharing

- Central queue needs mutual exclusion
  - may be a bottleneck when more than one processor looks for work at the same time
- Preempted threads are unlikely to resume execution on the same processor
  - cache use is less efficient
- It is unlikely that all threads of a program will gain access to the processors at the same time

Gang Scheduling

- Simultaneous scheduling of threads that make up a single process
- Useful for applications where performance severely degrades when any part of the application is not running
- Threads often need to synchronize with each other

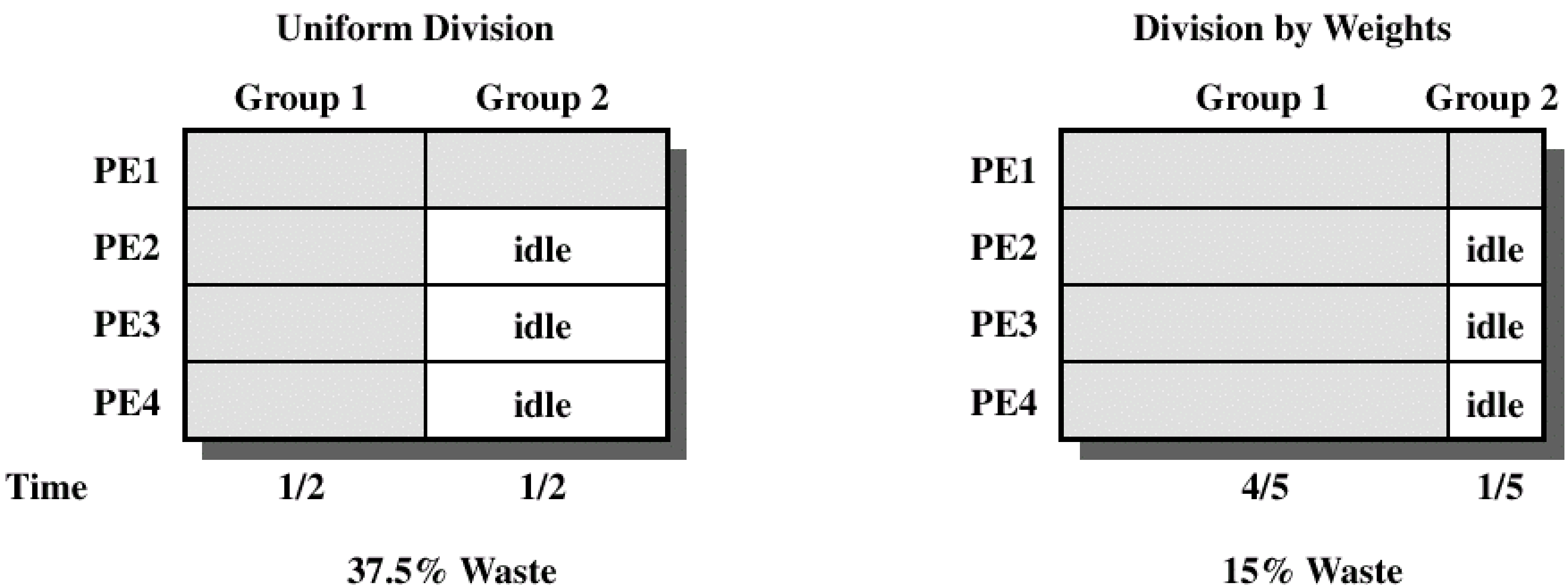


Figure 10.2 Example of scheduling groups with four and one threads

Dedicated Processor Assignment

- An extreme form of gang scheduling
  - dedicate a group of processors to an application for the duration of it
  - no multiprogramming
    - some processors may be idle
  - can it be efficient?
    - in a highly parallel system, processor utilization is no longer so important
    - avoidance of process switching should result in a substantial speedup of that program

Dynamic Scheduling

- Number of threads in a process may be altered dynamically by the application
- When a job requests processors
  - if there are idle processors, use them
  - new job may be assigned to a processor that is used by a job currently using more than one processor
  - hold request until processor is available
- Upon release of processors
  - new job will be given a processor before existing running applications

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
  - each task has a deadline
- Tasks or processes attempt to control or react to events that take place in the outside world
  - these events occur in “real-time” and process must be able to keep up with them

Real-Time Systems

- Hard real-time task
  - a task that must meet its deadline
- Soft real-time task
  - has an associated deadline that is desirable but not mandatory
- Periodic task
  - a task that must be executed periodically
  - once per period T
- Aperiodic task

Real-Time Systems

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

Characteristics of Real-Time Operating Systems

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

Characteristics of Real-Time Operating Systems

- 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
    - most of the requests for service are dictated by external events and timings

Characteristics of Real-Time Operating Systems

- Responsiveness
  - how long, after acknowledgment, it takes the operating system to service the interrupt
  - aspects of responsiveness
    - the amount of time to begin execution of the interrupt
    - the amount of time to perform the interrupt
    - effect of interrupt nesting

Characteristics of Real-Time Operating Systems

- User control
  - user has much broader control over the system
    - user specify priority
    - user specify paging and swapping
      - what processes must always reside in main memory
      - what disks algorithms to use
    - specify what rights the processes have

Characteristics of Real-Time Operating Systems

- Reliability
  - degradation of performance may have catastrophic consequences
    - financial loss
    - equipment damage
    - loss of life

Characteristics of Real-Time Operating Systems

- Fail-soft operation
  - ability of a system to fail in such a way as to preserve as much capability and data as possible
  - stability
    - a real-time system is stable if, in cases where it is impossible to meet all task deadlines, the system will meet the deadlines of its most critical, highest-priority tasks

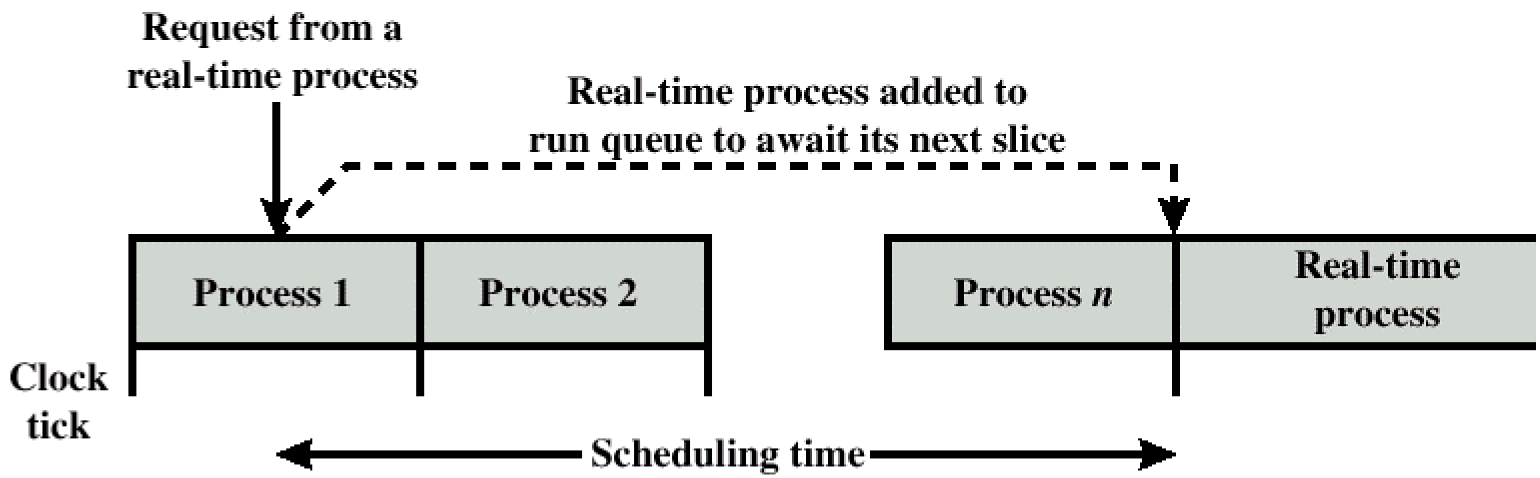
Features of Real-Time Operating Systems

- Fast context switch
- Small size
- Ability to respond to external interrupts quickly
- Multitasking with interprocess communication tools such as semaphores, signals, and events
- Files that accumulate data at a fast rate

Features of Real-Time Operating Systems

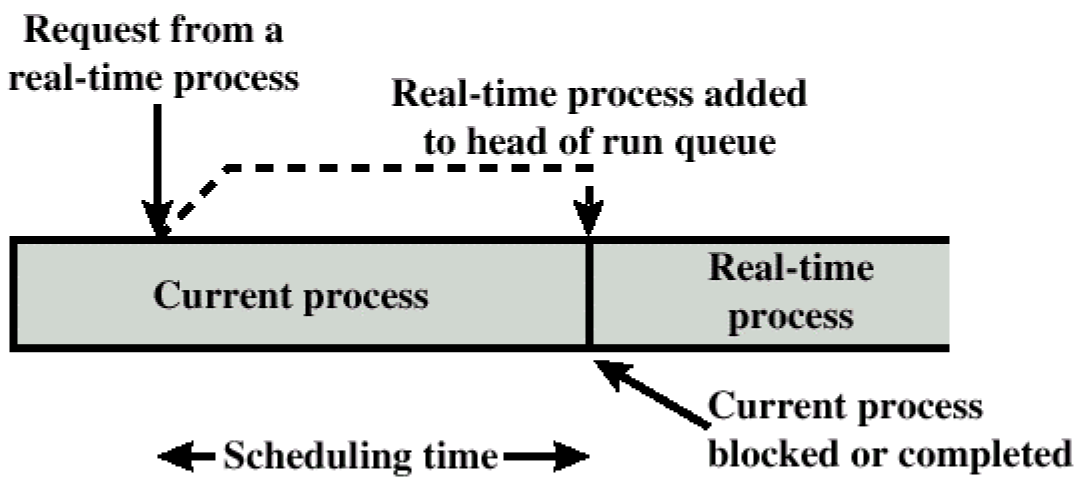
- Preemptive scheduling based on priority
  - immediate preemption allows operating system to respond to an interrupt quickly
- Minimization of intervals during which interrupts are disabled
- Delay tasks for fixed amount of time
- Special alarms and timeouts

Scheduling of a Real-Time Process



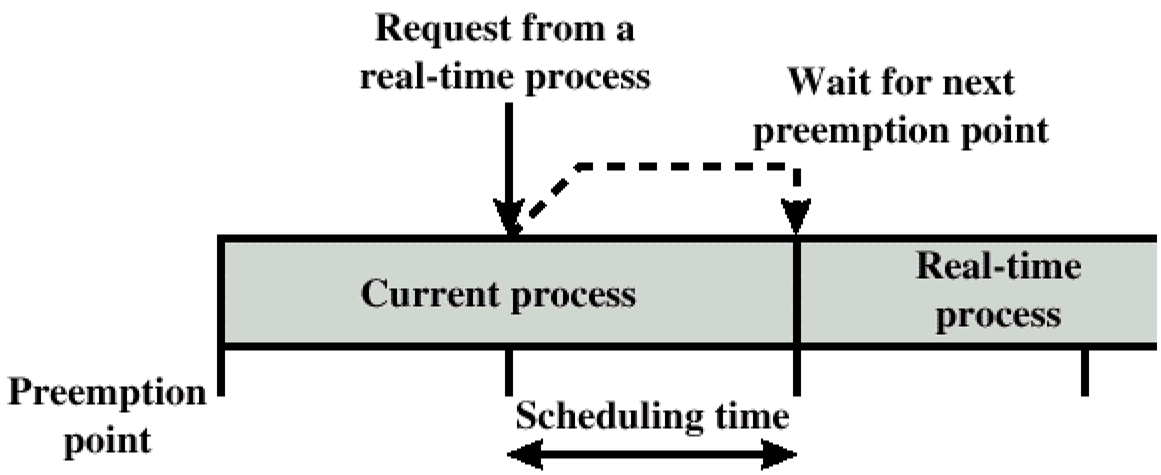
(a) Round-robin Preemptive Scheduler

Scheduling of a Real-Time Process



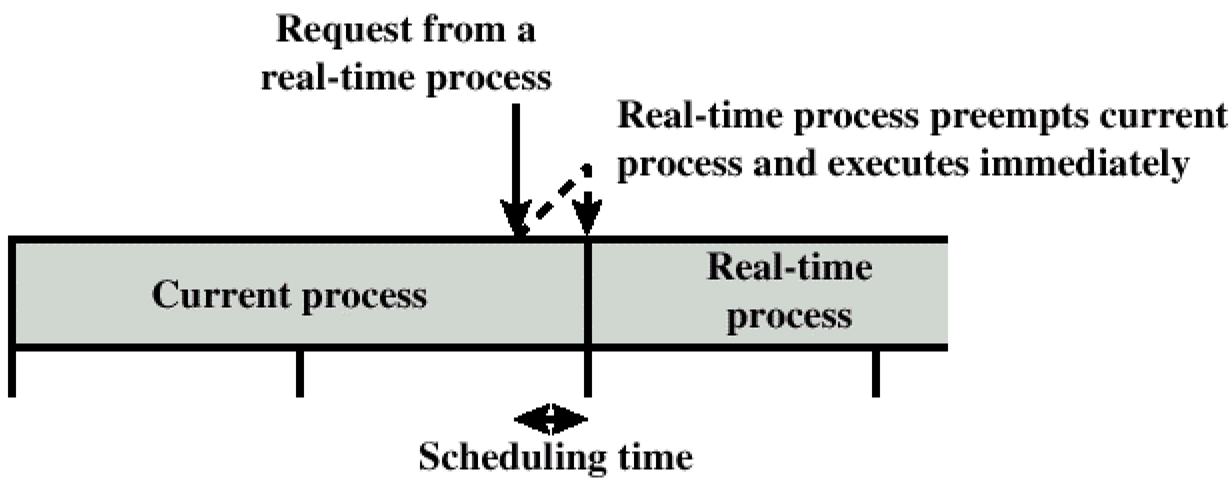
(b) Priority-Driven Nonpreemptive Scheduler

Scheduling of a Real-Time Process



(c) Priority-Driven Preemptive Scheduler on Preemption Points

Scheduling of a Real-Time Process



(d) Immediate Preemptive Scheduler

Figure 10.4 Scheduling of Real-Time Process

Quiz 3(20 points)

- Most operating systems have two schemes for memory management. One is for user processes and the other is for kernel. Why is that?

- Explain the usage of ‘copy on write’ bit in page table entry of Unix SVR4.

Real-Time Scheduling

- Classes of algorithms
  - Static table-driven
    - try to develop the complete schedule
    - determines when a task must begin execution
  - Static priority-driven preemptive
    - traditional priority-driven scheduler can be used
  - Dynamic planning-based
    - an attempt is made to create a schedule that contains the previously scheduled tasks as well as the new arrival
  - Dynamic best effort
    - when a task arrives, system assigns a priority based on the characteristics of the task

Deadline Scheduling

- Information about a task
  - ready time
  - starting deadline : a time by which a task must begin
  - completion deadline
  - processing time
  - resource requirements
  - priority
  - subtask structure
    - mandatory and optional subtask

Schedulability

- Let ?? ?? ??$_?$??$_?$????????$_n$?} be a task set
- ?$_i$ is said to be schedulable if it meets its deadline all the time
- ? is said to be schedulable if each task in ? is schedulable

Earliest Deadline Scheduling

- At each scheduling points, the task with the earliest deadline is selected to be run next
  - dynamic, priority-based preemptive scheduling
  - applicable to both periodic and aperiodic tasks
  - scheduling tasks with the earliest deadline minimized the fraction of tasks that miss their deadlines

Two Tasks

Table 10.2 Execution Profile of Two Periodic Tasks

Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
• • •	• • •	• • •	• • •
B(1)	0	25	50
B(2)	50	25	100
• • •	• • •	• • •	• • •

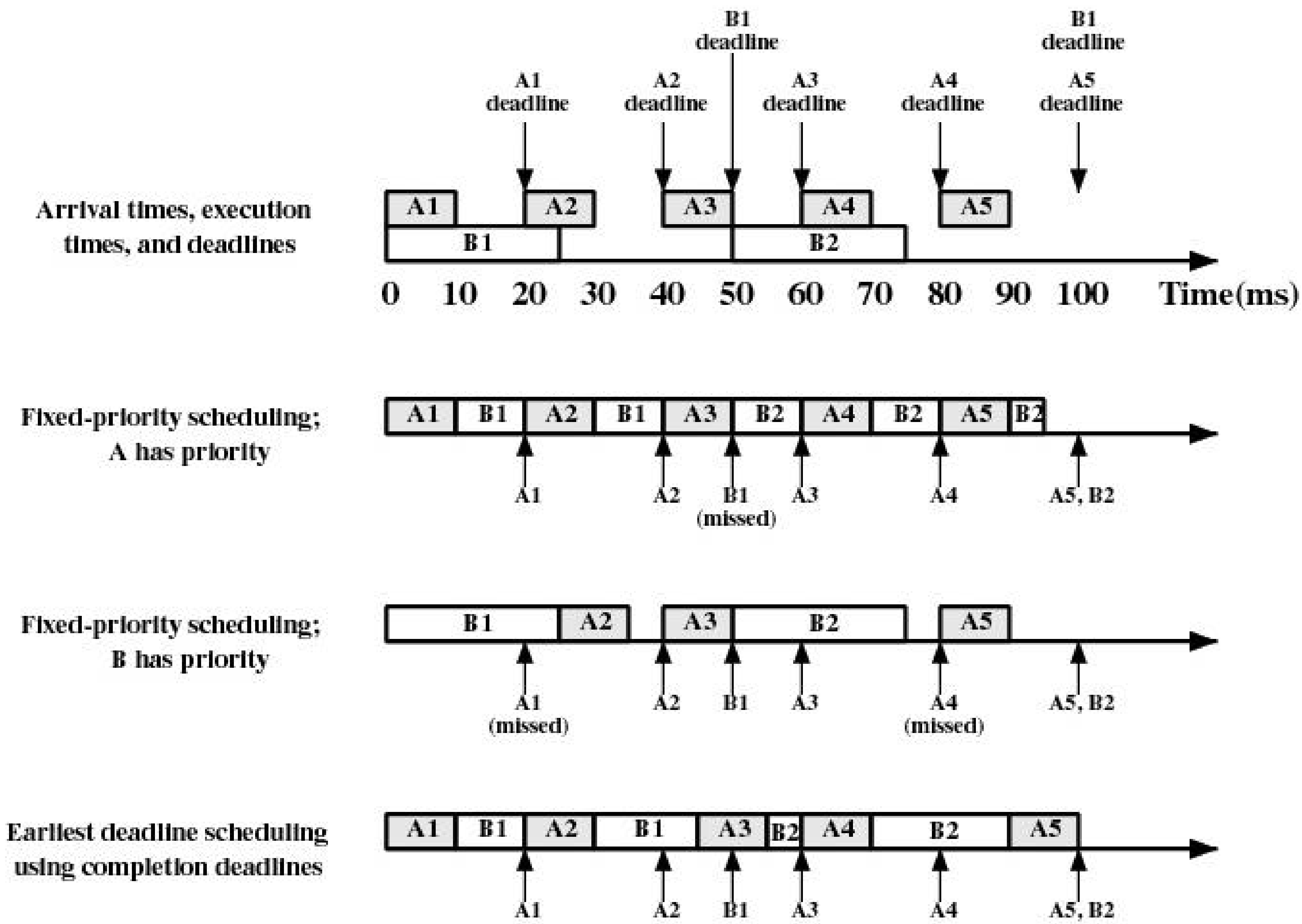


Figure 10.5 Scheduling of Periodic Real-time Tasks with Completion Deadlines

Rate Monotonic Scheduling

- Assigns priorities to tasks on the basis of their periods
  - highest-priority task is the one with the shortest period
  - applicable only to periodic tasks
  - static, priority-based preemptive scheduling

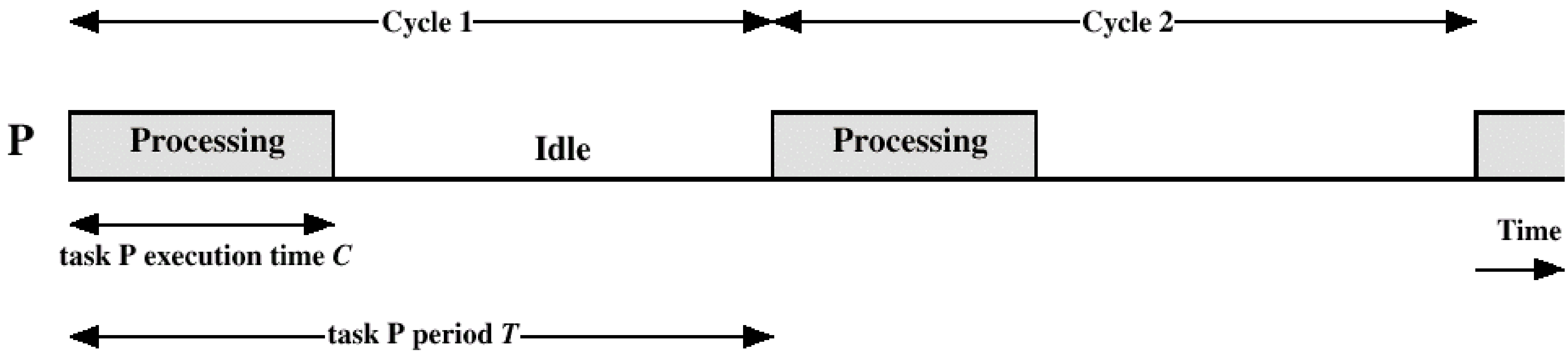


Figure 10.7 Periodic Task Timing Diagram

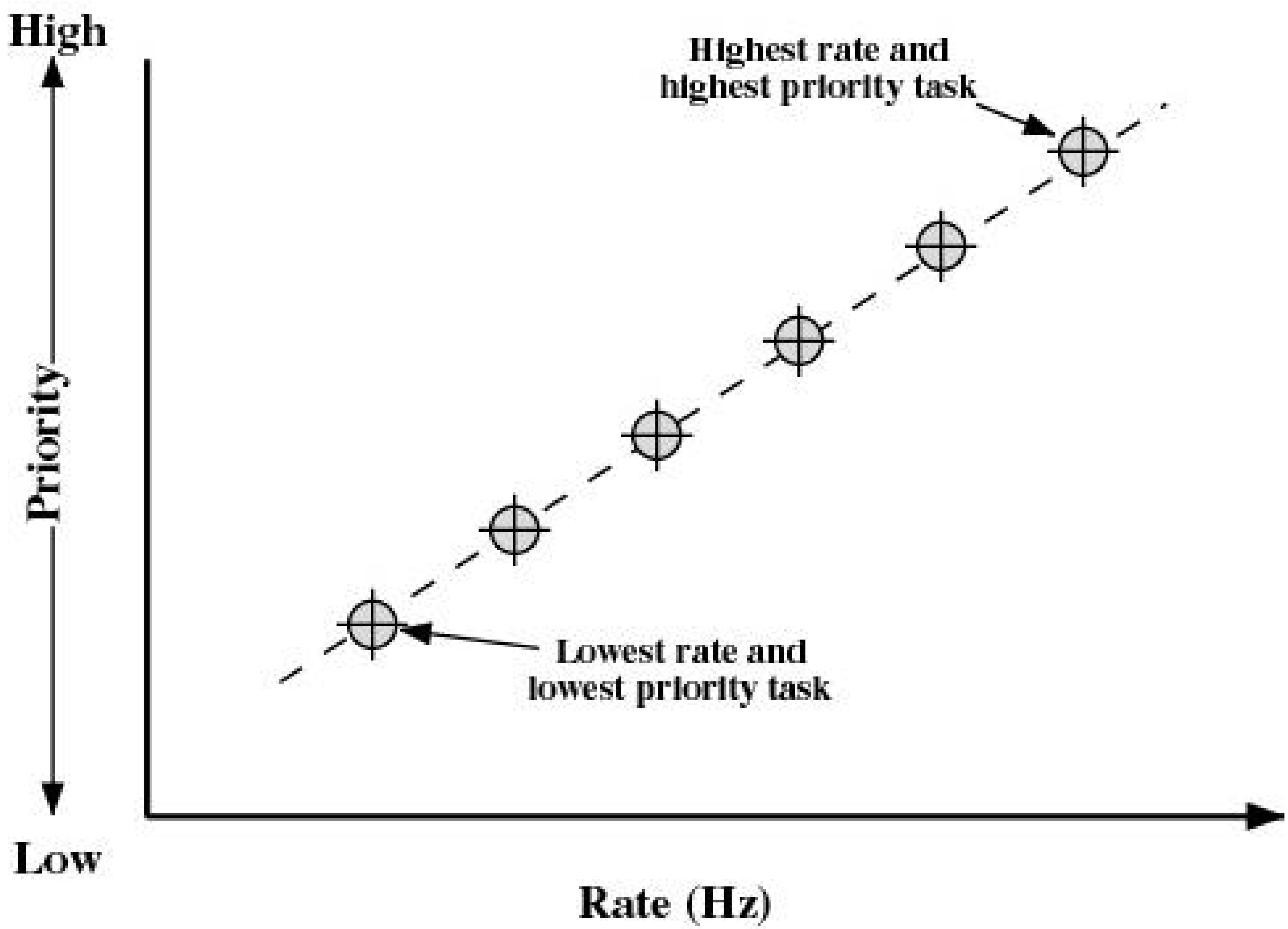


Figure 10.8 A Task Set with RMS [WARR91]

Table 10.4 Value of the RMS Upper Bound

n	$n(2^{1/n} - 1)$
1	1.0
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
•	•
•	•
•	•
∞	$\ln 2 \approx 0.693$

Linux Scheduling

- Scheduling classes
  - SCHED_FIFO: First-in-first-out real-time threads
  - SCHED_RR: Round-robin real-time threads
  - SCHED_OTHER: Other, non-real-time threads
    - traditional Unix scheduling algorithm is used here
- Within each class multiple priorities may be used

A	minimum
B	middle
C	middle
D	maximum

(a) Relative thread priorities

D → B → C → A →

(b) Flow with FIFO scheduling

D → B → C → B → C → A →

(c) Flow with RR scheduling

Figure 10.9 Example of Linux scheduling

UNIX SVR4 Scheduling

- Set of 160 priority levels divided into three priority classes
  - real time(159 ~ 100)
  - kernel(99 ~ 60)
  - time-shared(59 ~ 0)

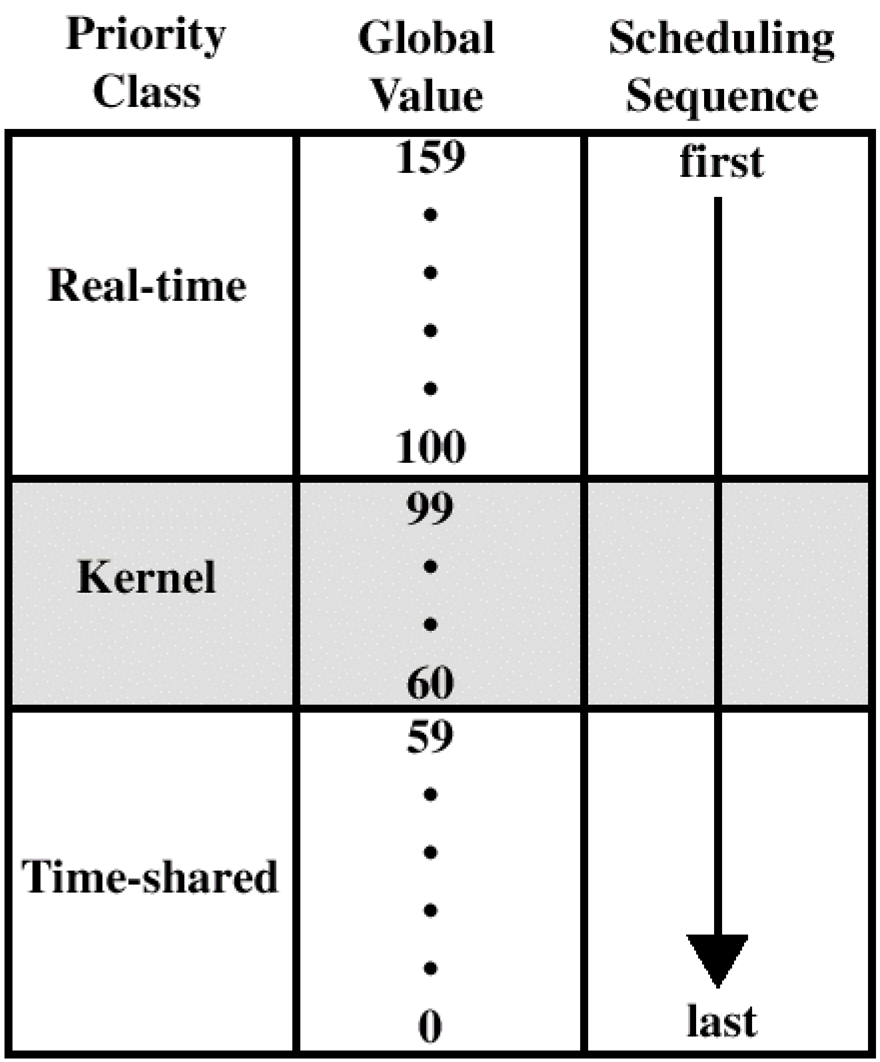


Figure 10.10 SVR4 priority classes

UNIX SVR4 Scheduling

- Scheduling
  - Highest preference to real-time processes
  - Next-highest to kernel-mode processes
  - Lowest preference to other user-mode processes
- processes at a given priority level are executed in round-robin fashion

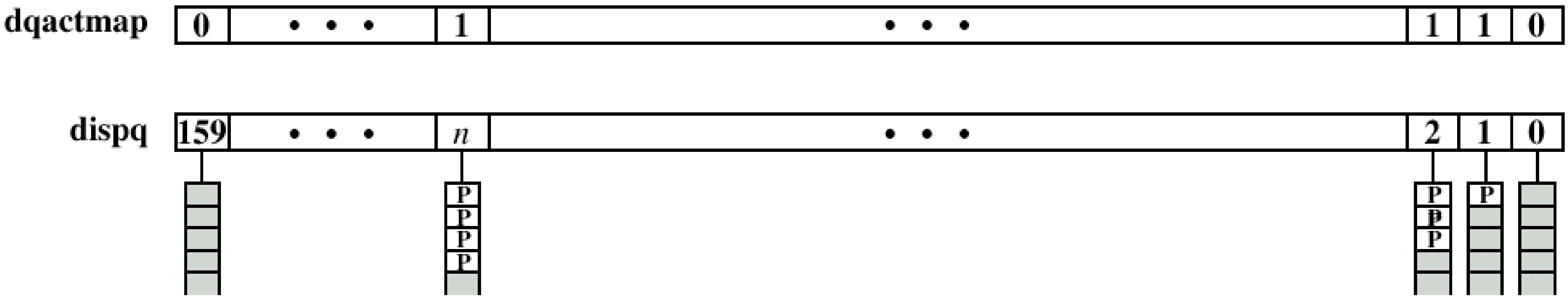


Figure 10.11 SVR4 Dispatch Queues

Windows 2000 Scheduling

- Priorities are organized into two bands or classes
  - Real time
    - all threads have a fixed priority that never changes
  - Variable
    - thread’s priority may change during it’s lifetime
- each band consists of 16 priority levels
- Priority-driven preemptive scheduler

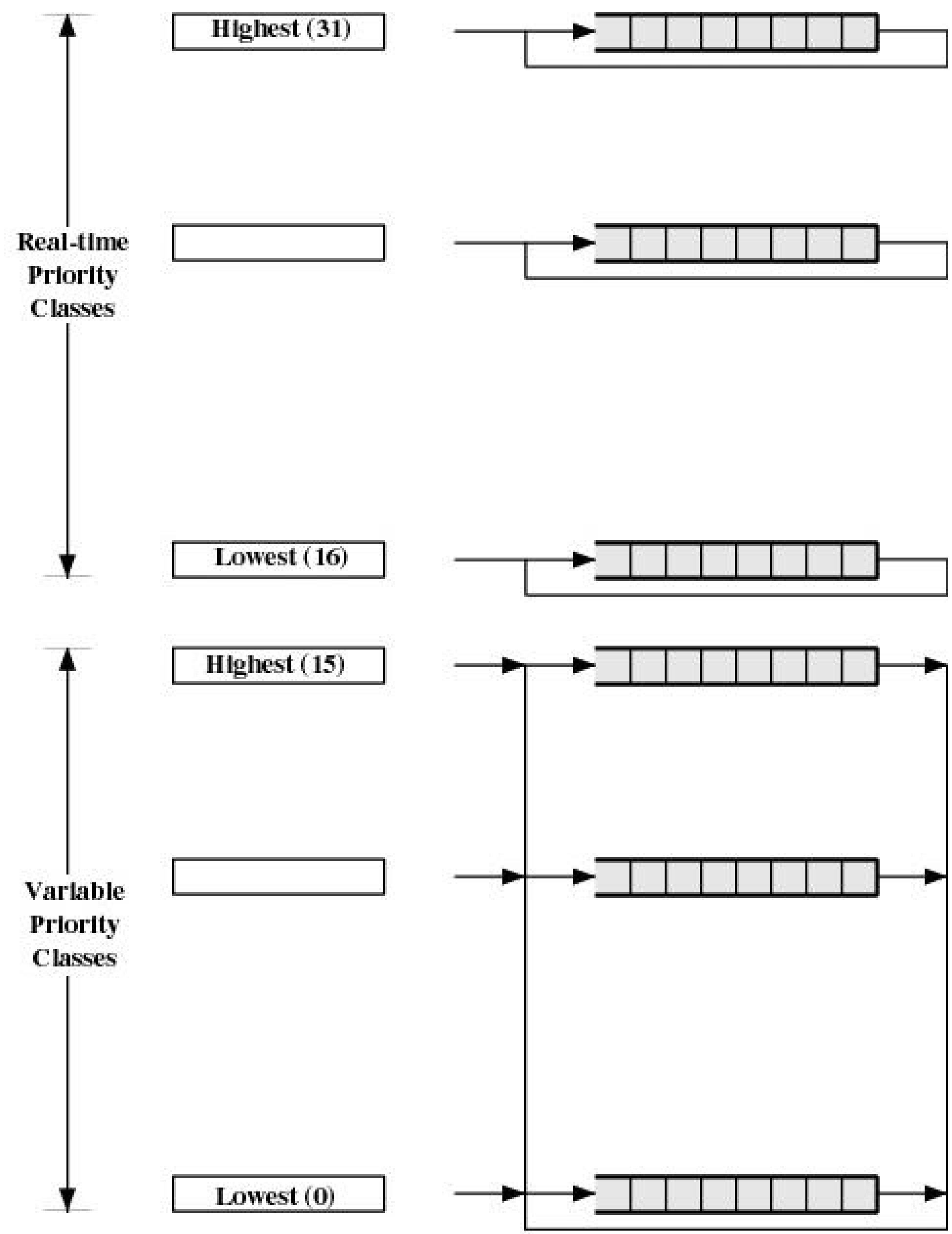


Figure 10.11 Windows 2000 Thread Dispatching Priorities