Multiprocessor and Real-Time Scheduling

Athar Mahboob
MIS & CS Department
Institute of Business Administration
athar@atharmahboob.com
http://www.atharmahboob.com

Classifications of Multiprocessor Systems

- 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 operating system

Independent Parallelism

- Separate application or job
- No synchronization
- More than one processor is available
  - Average response time to users is less

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

- Parallel processing or multitasking within a single application
- Single application is a collection of threads
- Threads usually interact frequently

Fine-Grained Parallelism

- Highly parallel applications
- Specialized and fragmented area

Scheduling

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

Assignment of Processes to Processors

- Treat processors as a pooled resource and assign process to processors on demand
- Permanently assign process to a processor
  - Dedicate short-term queue for each processor
  - Less overhead
  - Processor could be idle while another processor has a backlog

Assignment of Processes to Processors

- Global queue
  - Schedule to any available processor
- 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

Assignment of Processes to Processors

- Peer architecture
  - Operating system 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

- Single queue for all processes
- Multiple queues are used for priorities
- All queues feed to the common pool of processors
- Specific scheduling disciplines is less important with more than on processor

Threads

- Execution separate from the rest of the process
- An application can be a set of threads that cooperate and execute concurrent-ly in the same address space
- Threads running on separate proces-sors yields a dramatic gain in perfor-mance

Multiprocessor Thread Scheduling

- Load sharing
  - Processes are not assigned to a particular processor
- Gang scheduling
  - A set of related threads is scheduled to run on a set of processors at the same time

Multiprocessor Thread Scheduling

- Dedicated processor assignment
  - Threads are assigned to a specific processor
- Dynamic scheduling
  - Number of threads can be altered during course of execution

Load Sharing

- Load is distributed evenly across the processors
- No centralized scheduler required
- Use global queues

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
- Preemptive threads are unlikely resume execution on the same processor
  - Cache use is less efficient
- 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

- 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

Scheduling Groups

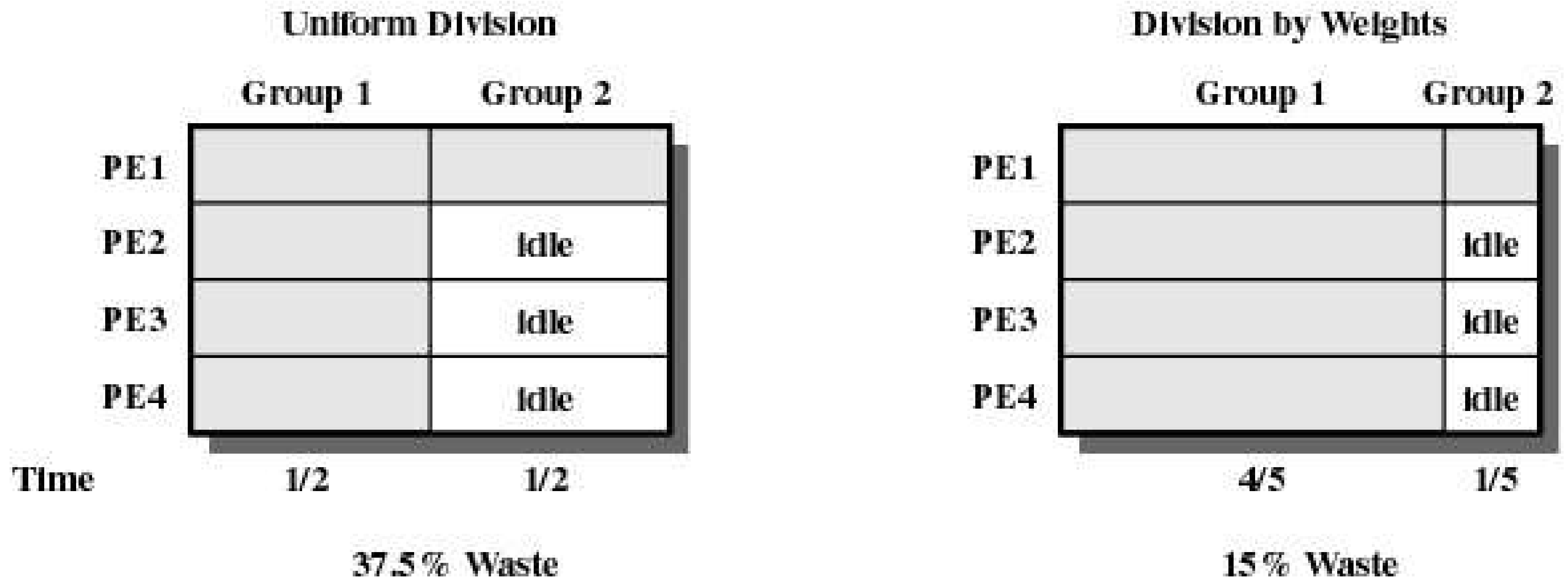


Figure 10.2 Example of Scheduling Groups with Four and One Threads [FEIT90]

Dedicated Processor Assignment

- When application is scheduled, its threads are assigned to a processor
- Some processors may be idle
- No multiprogramming of processors

Dynamic Scheduling

- Number of threads in a process are altered dynamically by the application
- Operating system adjusts the load to improve use
  - Assign idle processors
  - New arrivals may be assigned to a processor that is used by a job currently using more than one processor
  - Hold request until processor is available
  - New arrivals 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
- 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

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

Characteristics of Real-Time Operating Systems

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

Characteristics of Real-Time Operating Systems

- Responsiveness
  - How long, after acknowledgment, it takes the operating system to service the interrupt
  - Includes amount of time to begin execution of the interrupt
  - Includes the amount of time to perform the interrupt

Characteristics of Real-Time Operating Systems

- User control
  - User specifies priority
  - Specify paging
  - What processes must always reside in main memory
  - Disks algorithms to use
  - Rights of processes

Characteristics of Real-Time Operating Systems

- Reliability
  - Degradation of performance may have catas-trophic consequences
  - Attempt either to correct the problem or min-imize its effects while continuing to run
  - Most critical, high priority tasks execute

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

- 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

Scheduling of a Real-Time Process

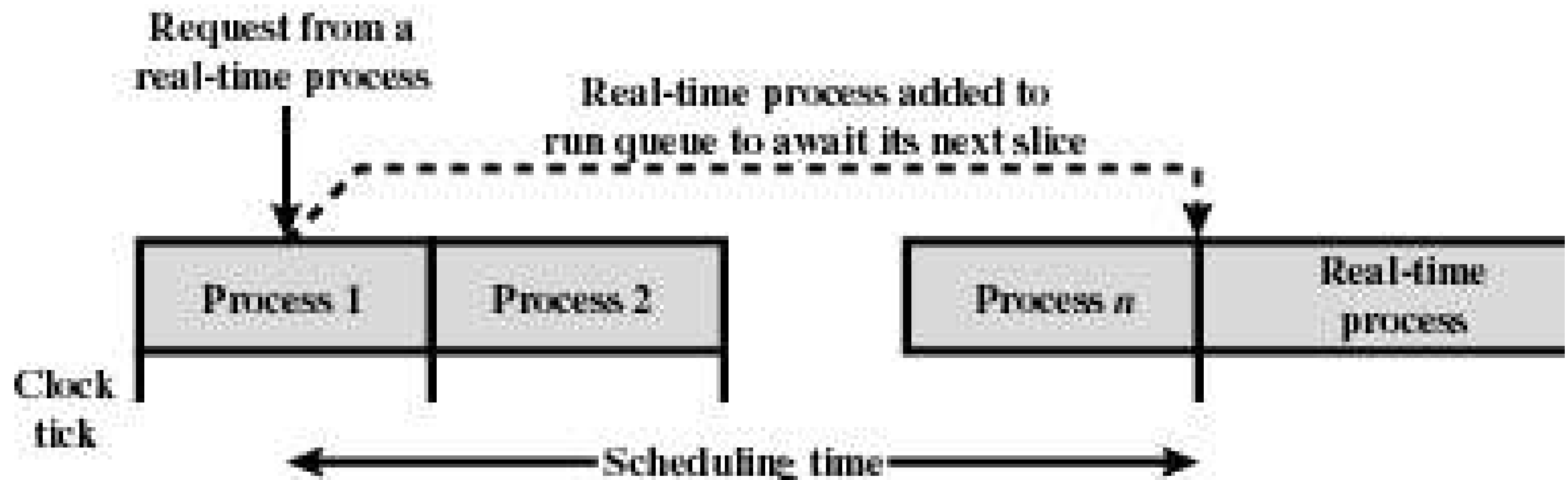


Figure 10.4 Scheduling of Real-Time Process

Scheduling of a Real-Time Process

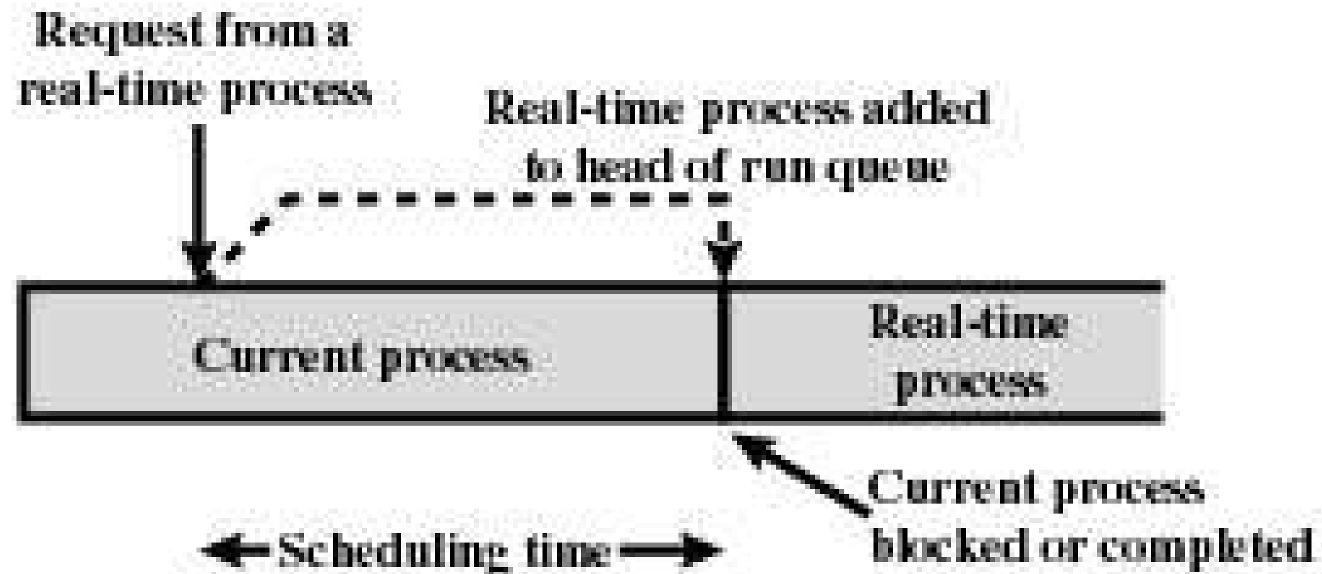


Figure 10.4 Scheduling of Real-Time Process

Scheduling of a Real-Time Process

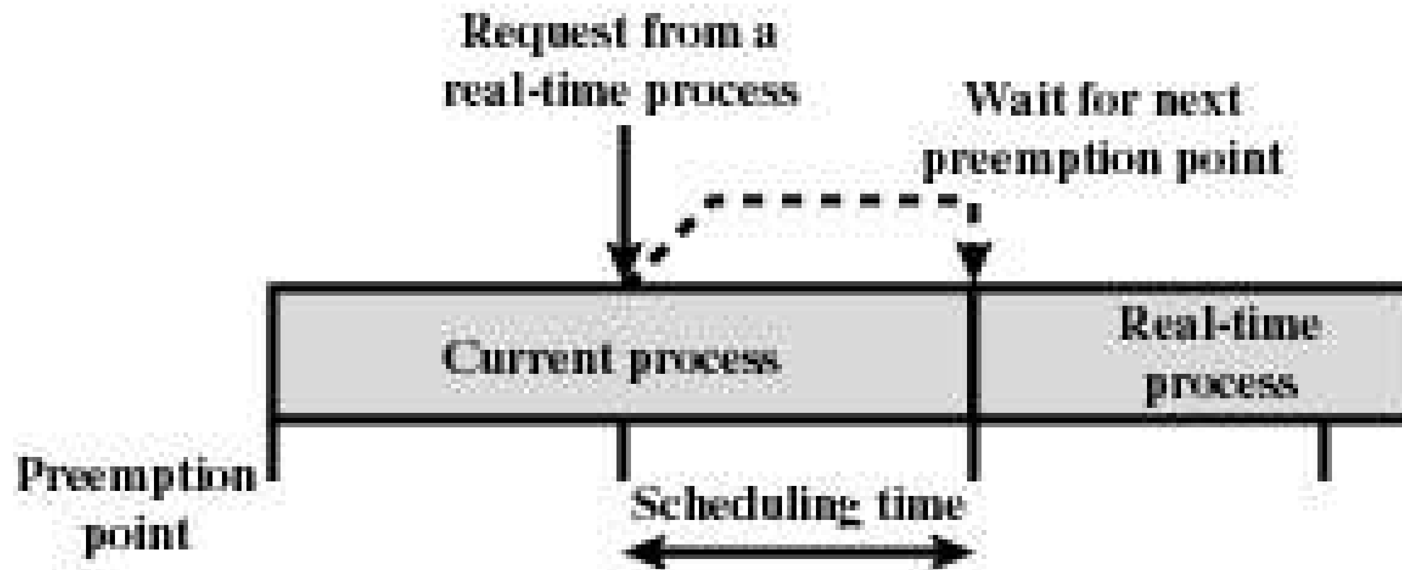


Figure 10.4 Scheduling of Real-Time Process

Scheduling of a Real-Time Process

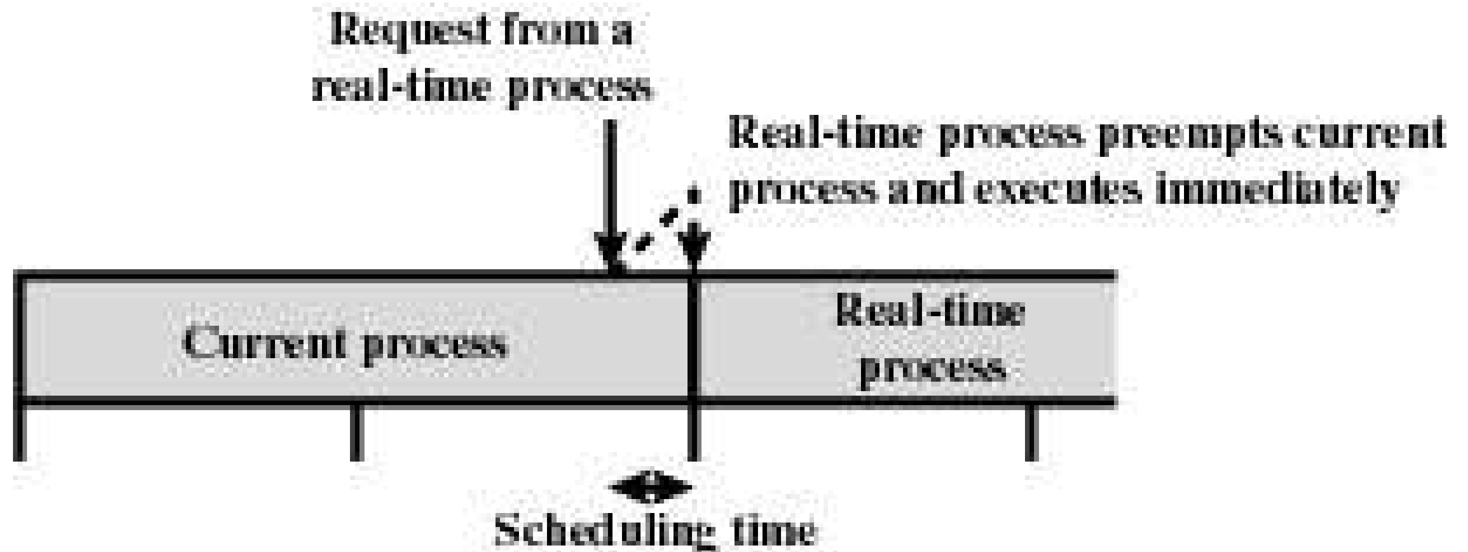


Figure 10.4 Scheduling of Real-Time Process

Real-Time Scheduling

- Static table-driven
  - Determines at run time when a task begins execution
- Static priority-driven preemptive
  - Traditional priority-driven scheduler is used
- Dynamic planning-based
- Dynamic best effort

Deadline Scheduling

- Real-time applications are not con-cerned with speed but with completing tasks
- Scheduling tasks with the earliest dead-line minimized the fraction of tasks that miss their deadlines

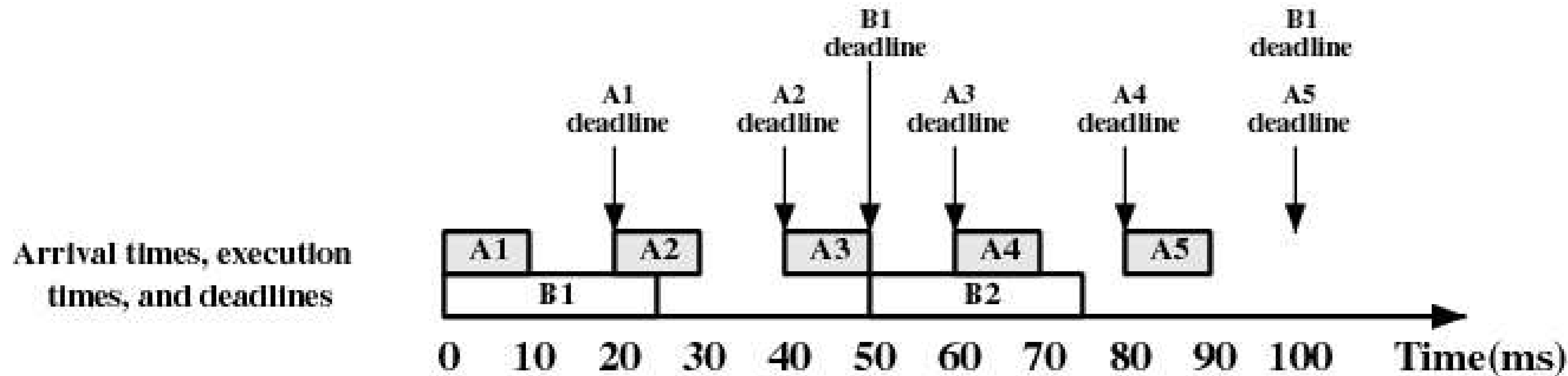
Deadline Scheduling

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

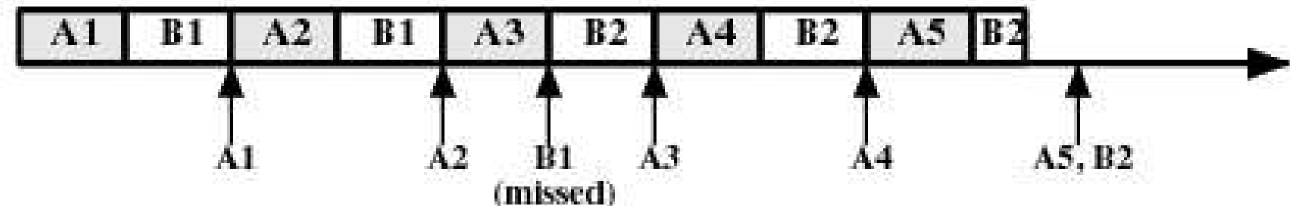
Two Tasks

Table 10.2 Execution Profile of Two Periodic Tasks

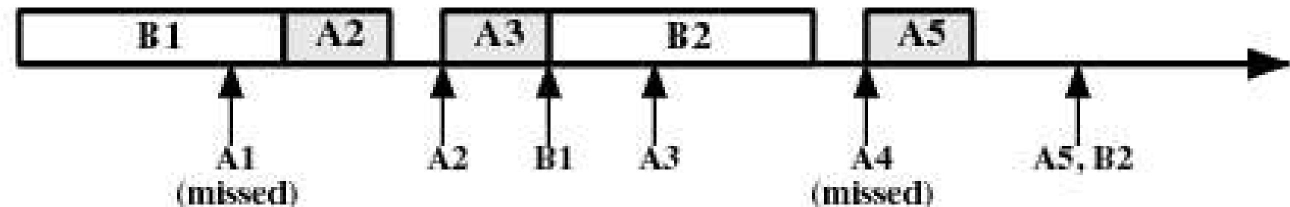
Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
• • •	• • •	• • •	• • •
B(1)	0	25	50
B(2)	50	25	100
• • •	• • •	• • •	• • •



Fixed-priority scheduling;
A has priority



Fixed-priority scheduling;
B has priority



Earliest deadline scheduling
using completion deadlines

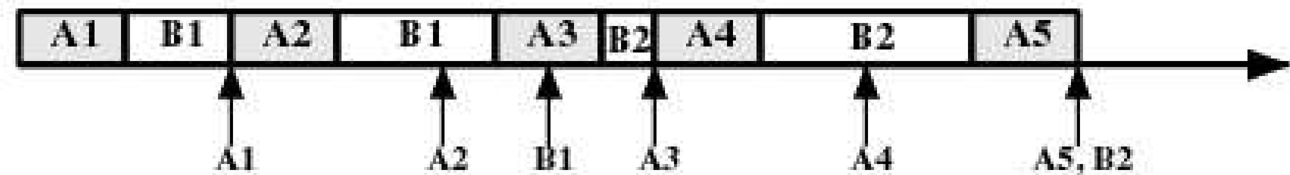


Figure 10.5 Scheduling of Periodic Real-time Tasks with Completion Deadlines

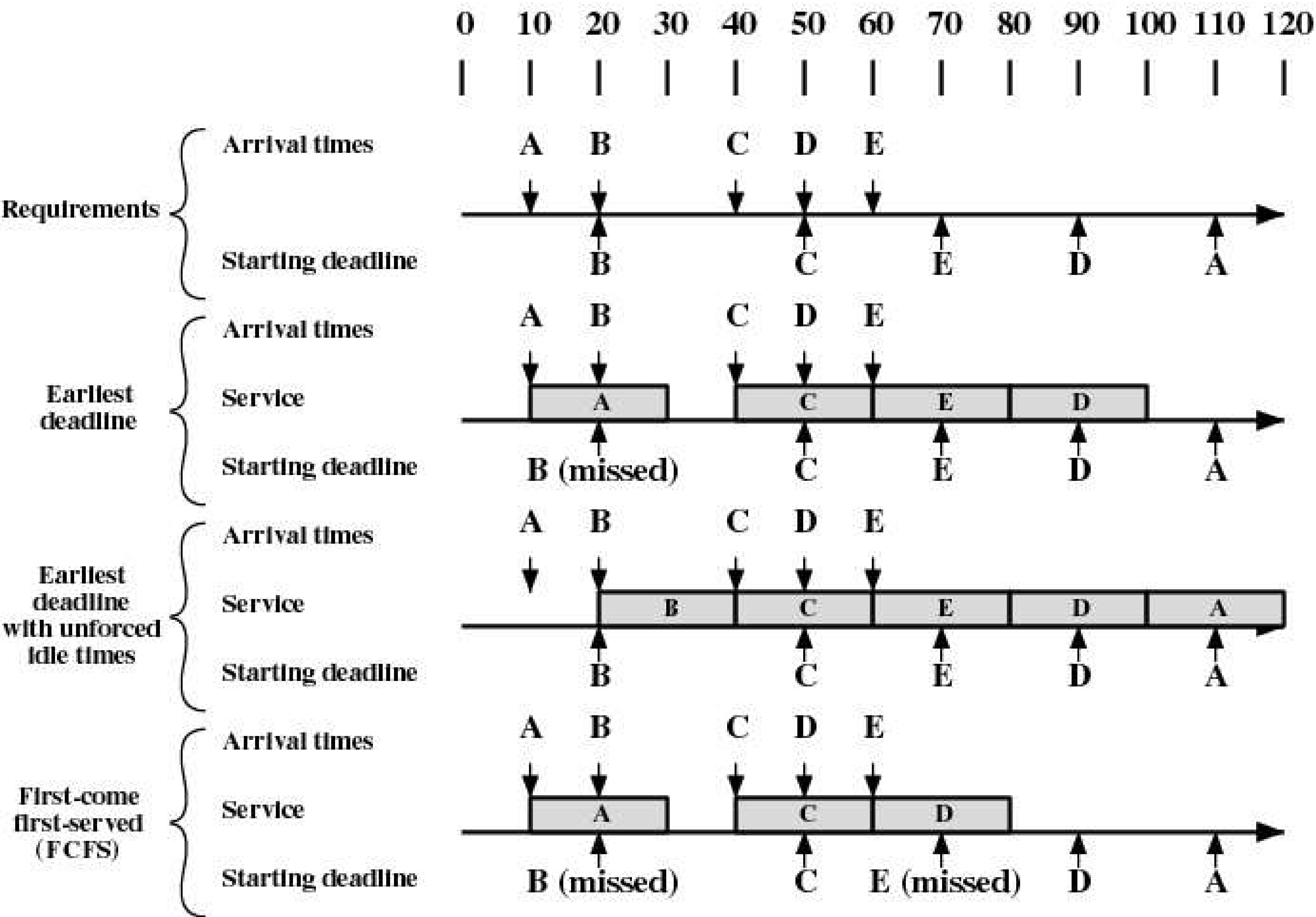


Figure 10.6 Scheduling of Aperiodic Real-time Tasks with Starting Deadlines

Rate Monotonic Scheduling

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

Periodic Task Timing Diagram

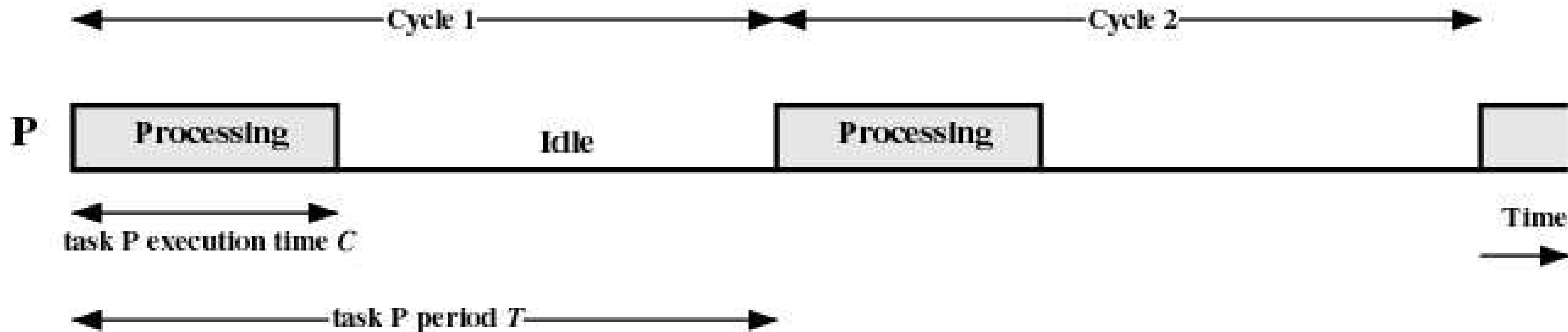


Figure 10.7 Periodic Task Timing Diagram

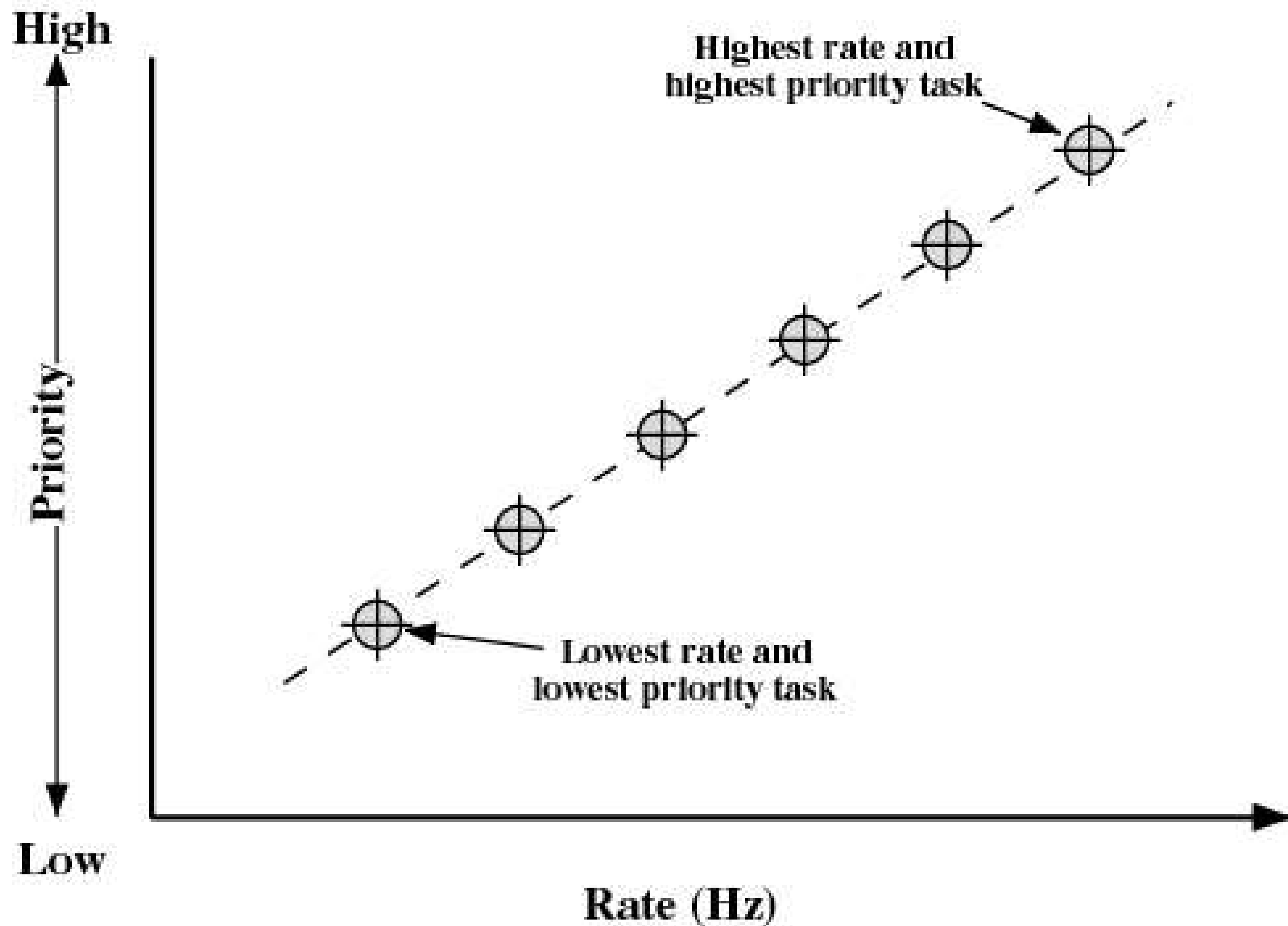


Figure 10.8 A Task Set with RMS [WARR91]

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
- Within each class multiple priorities may be used

A	minimum
B	middle
C	middle
D	maximum

(a) Relative thread priorities

(b) Flow with FIFO scheduling

(c) Flow with RR scheduling

Figure 10.9 Example of Linux Scheduling

UNIX SVR4 Scheduling

- Highest preference to real-time processes
- Next-highest to kernel-mode processes
- Lowest preference to other user-mode processes

SVR4 Dispatch Queues

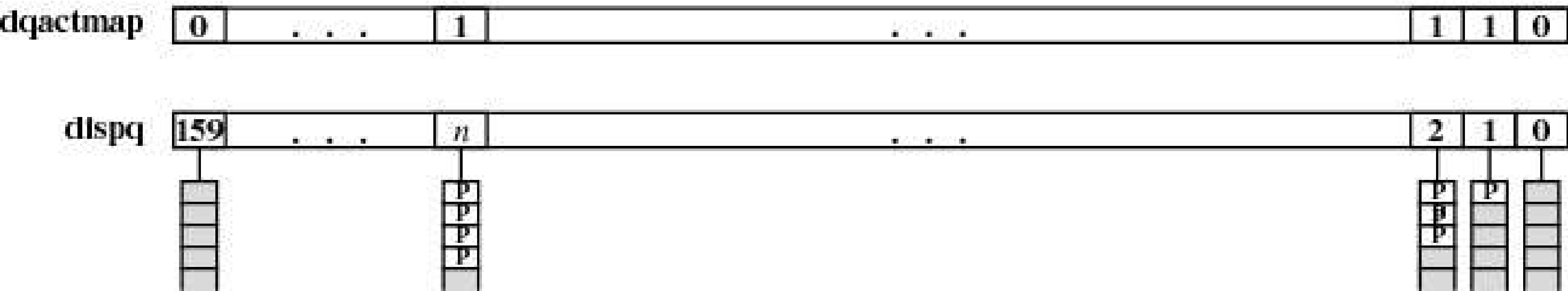


Figure 10.11 SVR4 Dispatch Queues

Windows 2000 Scheduling

- Priorities organized into two bands or classes
  - Real-time
  - Variable
- Priority-driven preemptive scheduler

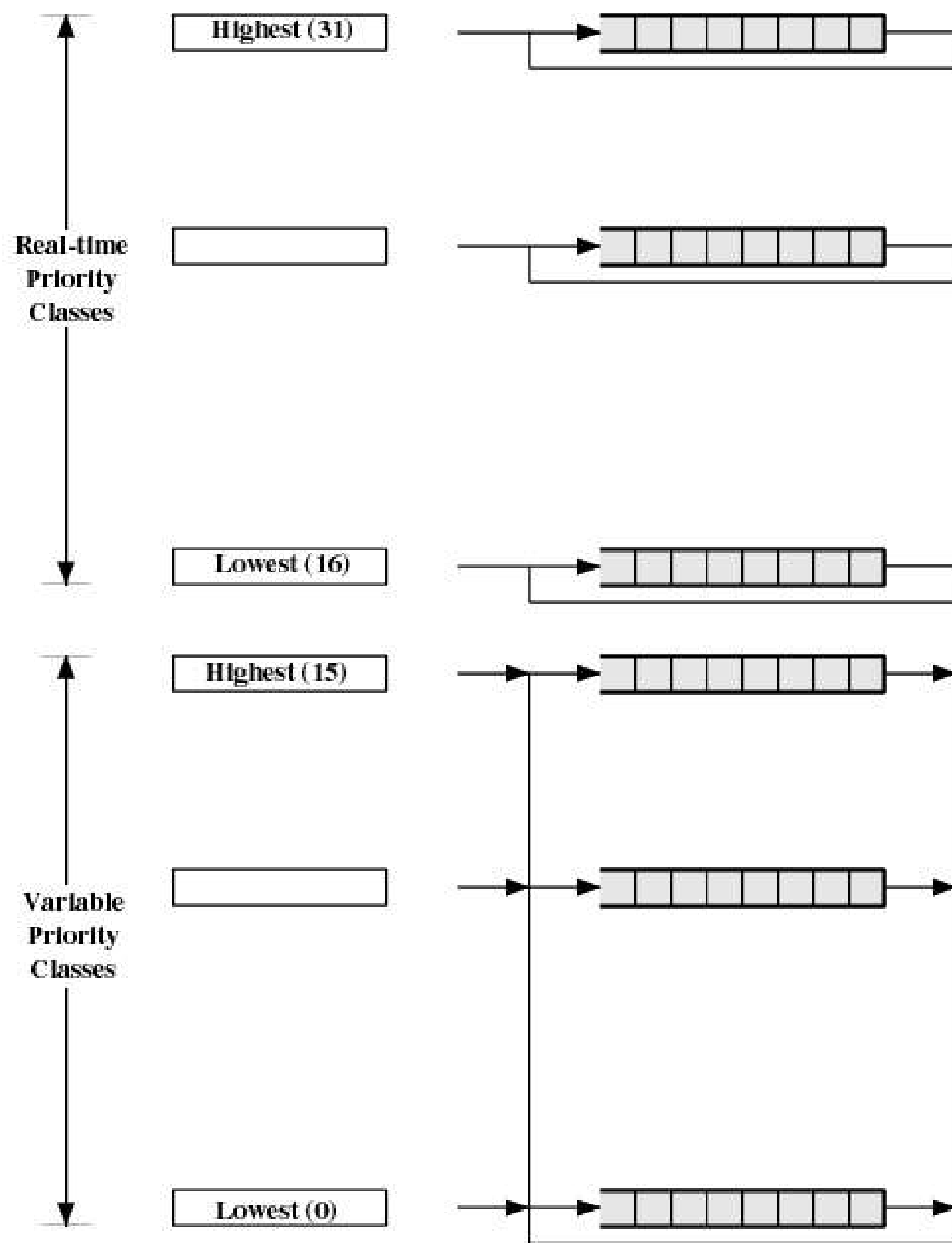


Figure 10.11 Windows 2000 Thread Dispatching Priorities

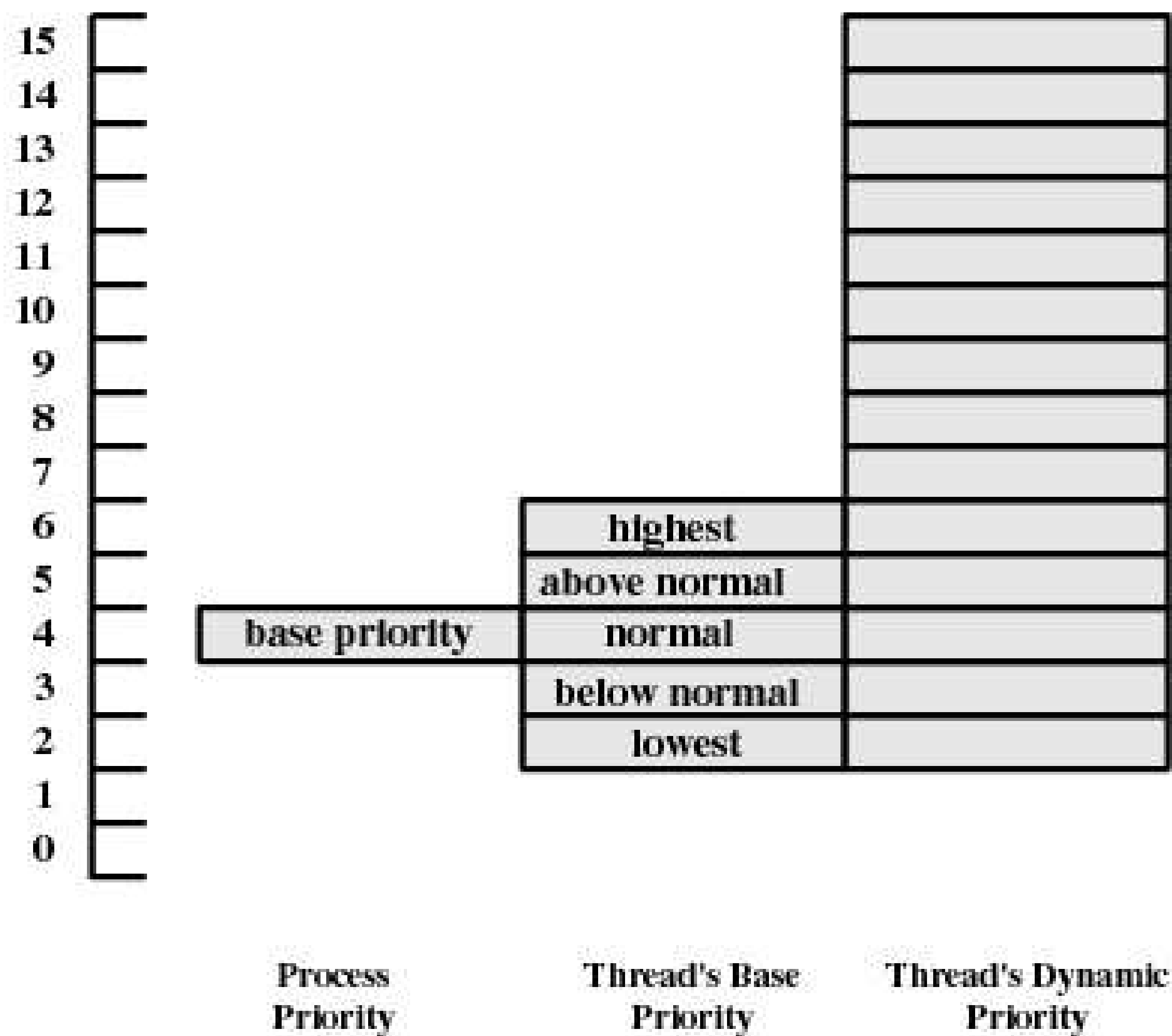


Figure 10.13 Example of Windows 2000 Priority Relationship