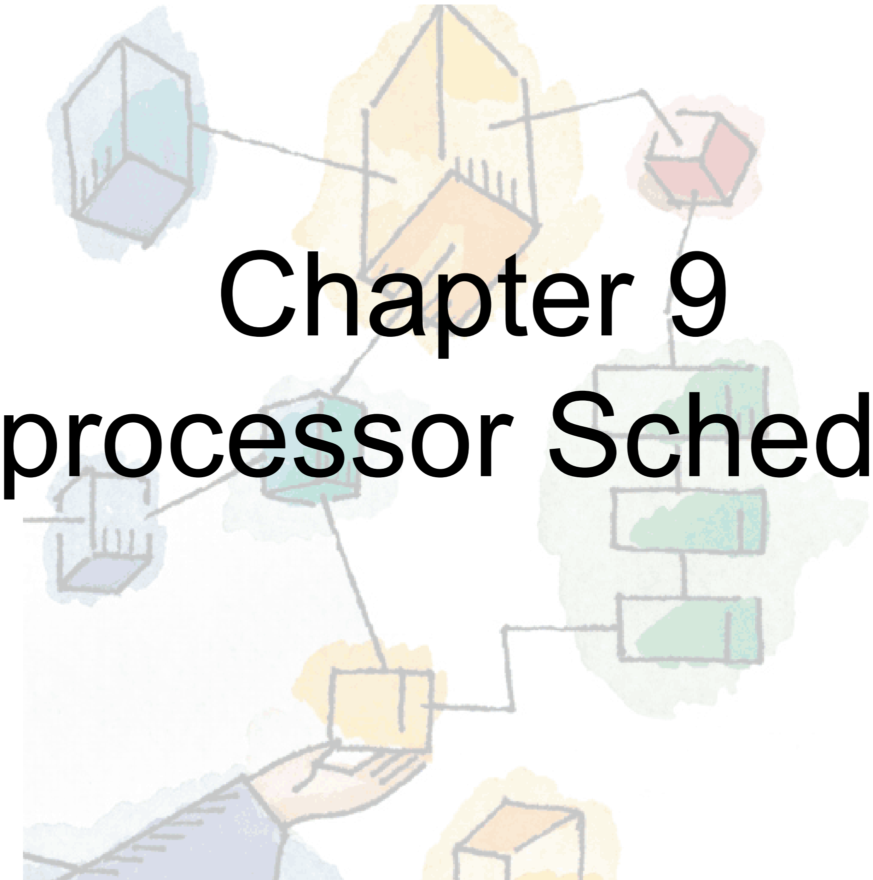
Operating Systems:
Internals and Design Principles
William Stallings



Chapter 9
Uniprocessor Scheduling

Outline

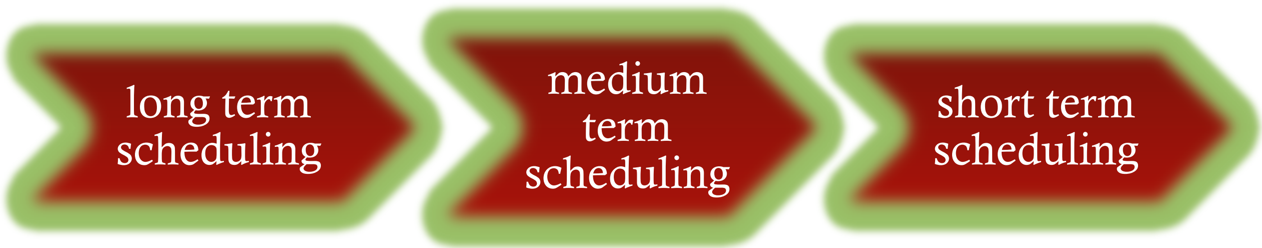
- Types of scheduling
  - Long term, medium term and short term
- Scheduling criteria
  - User-oriented, system-oriented, performance-related, non-performance-related
- Scheduling policies/algorithms
  - Preemptive and non-preemptive
  - Priority
  - FCFS, RR, SJF, SRT, FB, HRRN
- Priority inversion
- Thread/multicore CPU scheduling
- Algorithm evaluation

Processor Scheduling

- The resource provided by a processor is execution time
  - The resource is allocated by means of a schedule
- Aim is to assign processes to be executed by the processor in a way that meets system objectives, such as response time, throughput, and processor efficiency

Types of Scheduling

- Broken down into three separate functions:



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
  - may limit to provide satisfactory service to the current set of processes

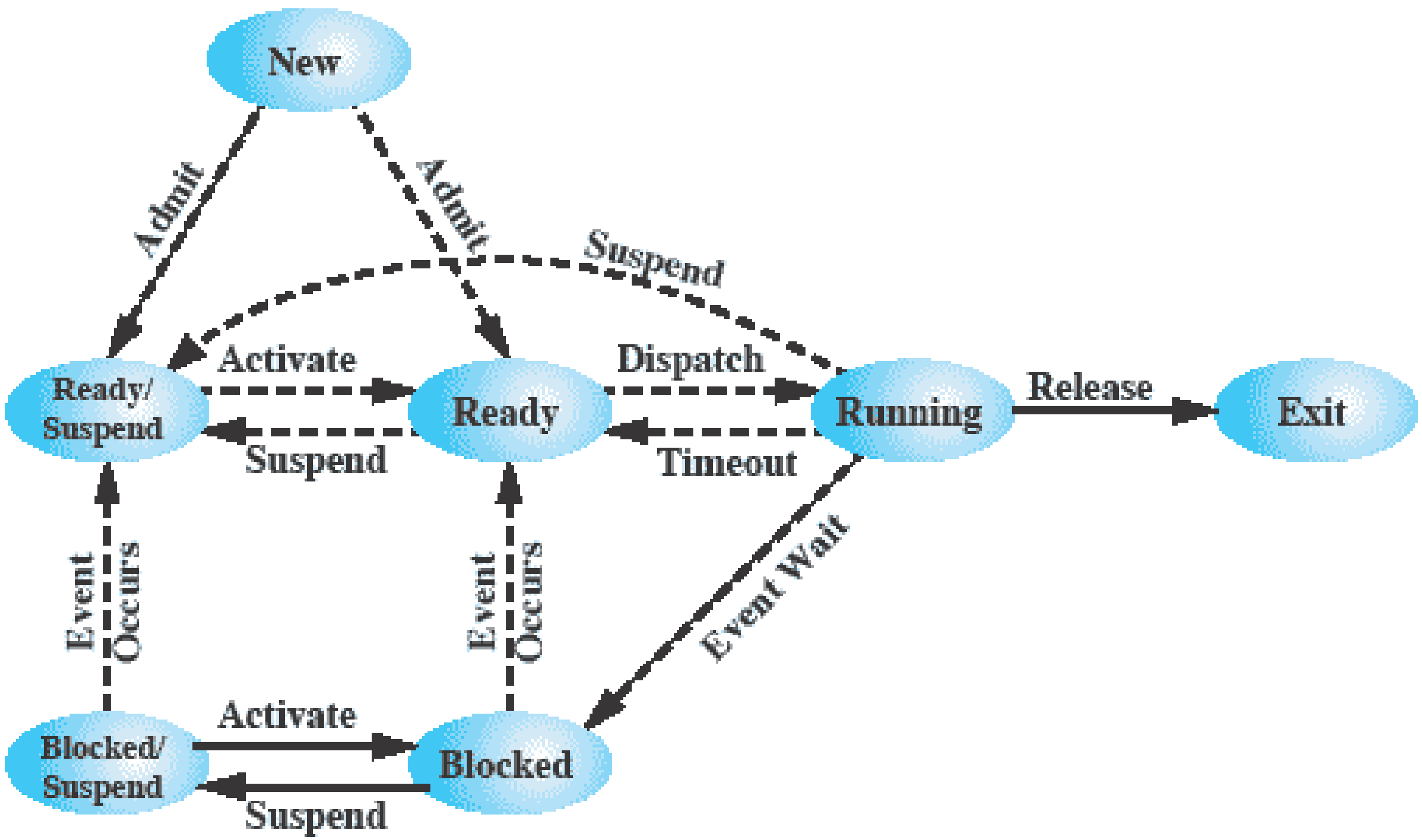
Medium-Term Scheduling

- Part of the swapping function
- Swapping-in decisions are based on the need to manage the degree of multiprogramming
- considers the memory requirements of the swapped-out processes

Short-Term Scheduling

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

Two Suspend States



(b) With Two Suspend States

Scheduling and Process State Transitions

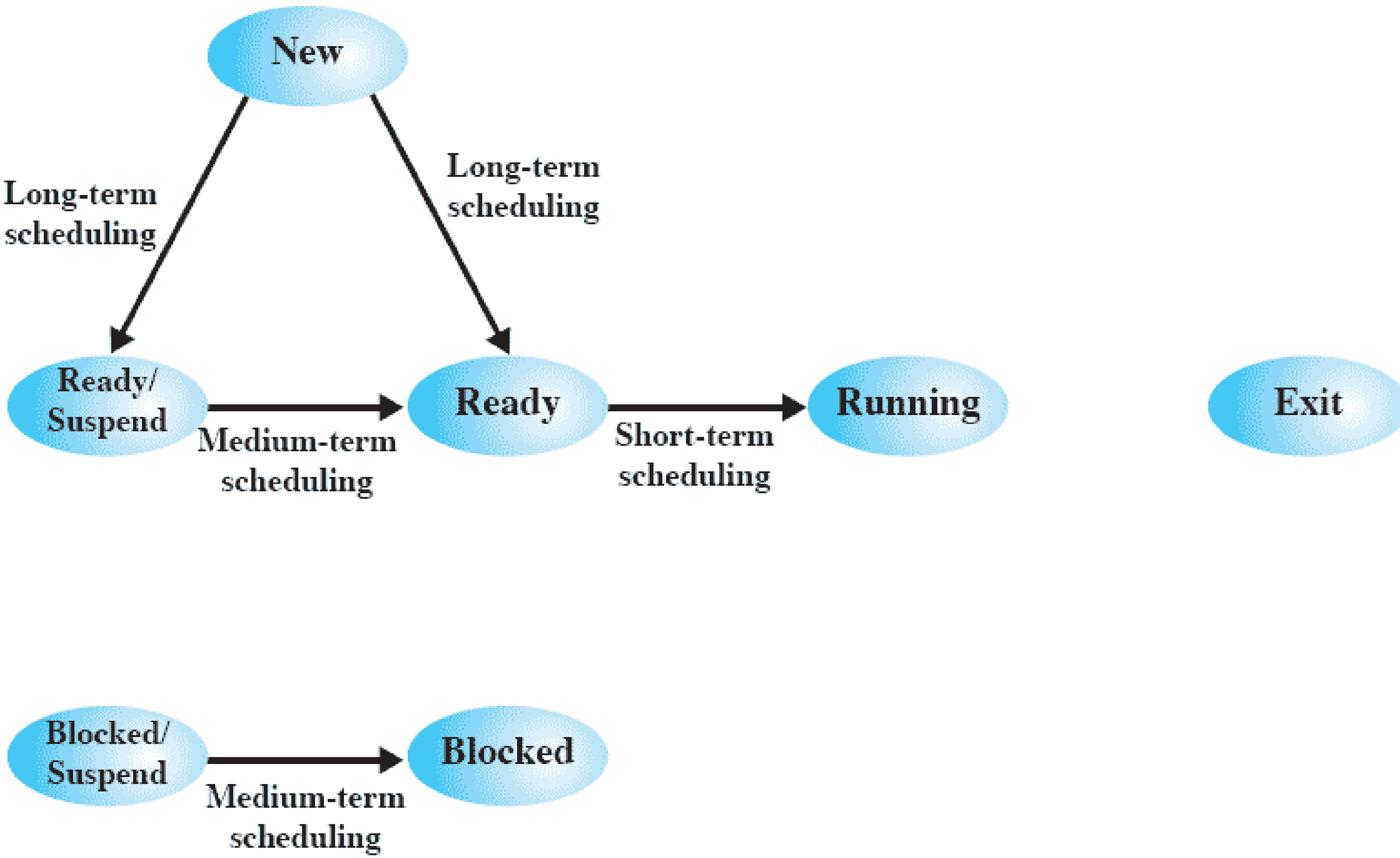


Figure 9.1 Scheduling and Process State Transitions

Queuing Diagram

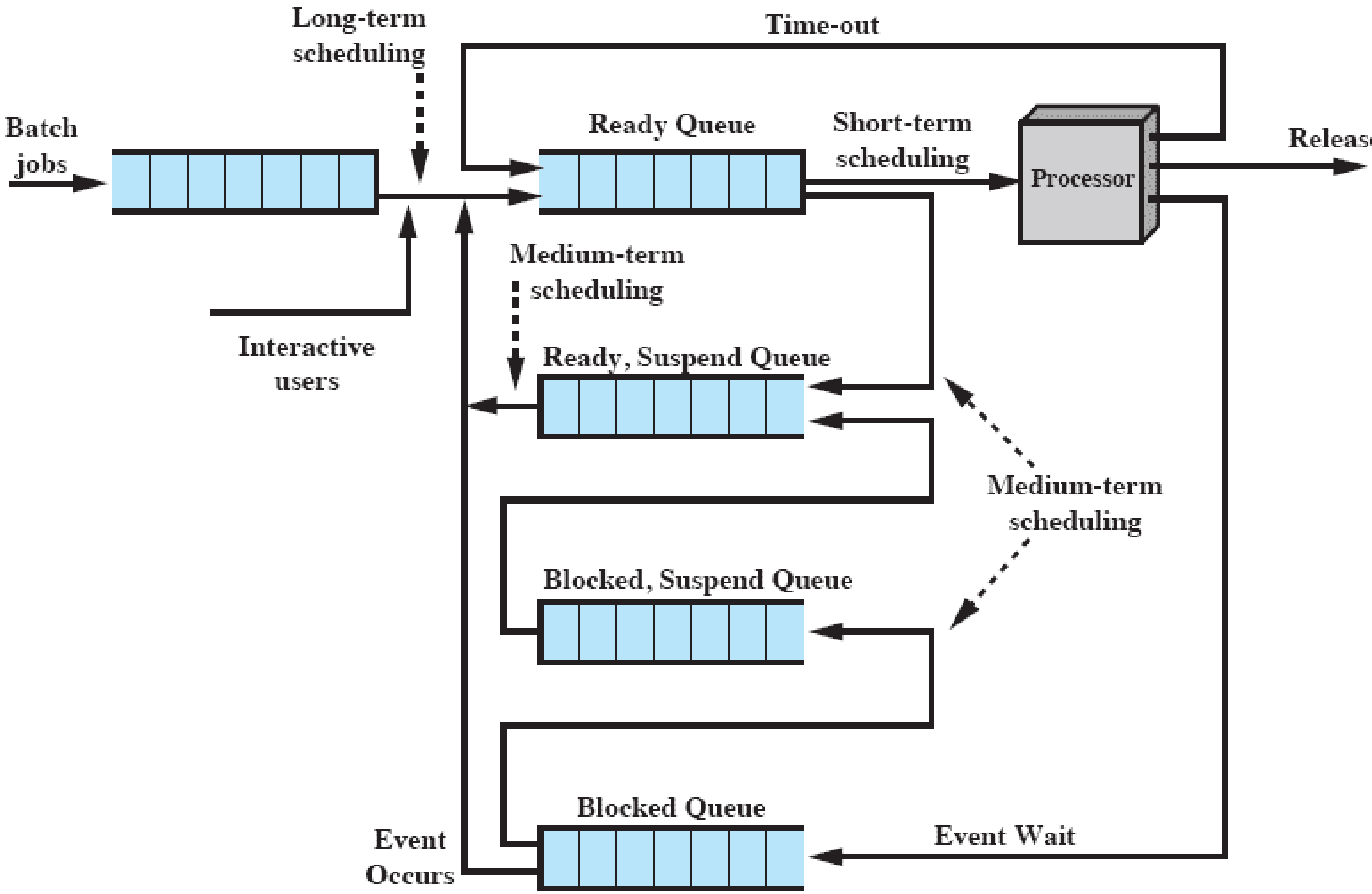


Figure 9.3 Queuing Diagram for Scheduling

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.

Scheduling Criteria: User vs. System

- User-oriented criteria
  - relate to the behavior of the system as perceived by the individual user or process (such as response time in an interactive system)
  - important on virtually all systems
- System-oriented criteria
  - focus on effective and efficient utilization of the processor (rate at which processes are completed)
  - generally of minor importance on single-user systems

Scheduling Criteria: Performance

examples:
- response time
- throughput

example:
- predictability

Criteria can be classified into:

- Performance-related
  - quantitative
  - easily measured
- Non-performance related
  - qualitative
  - hard to measure

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.

Scheduling Criteria

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.

Non-preemptive vs Preemptive

- 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.

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

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 – dynamic proirity

Selection Function

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

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

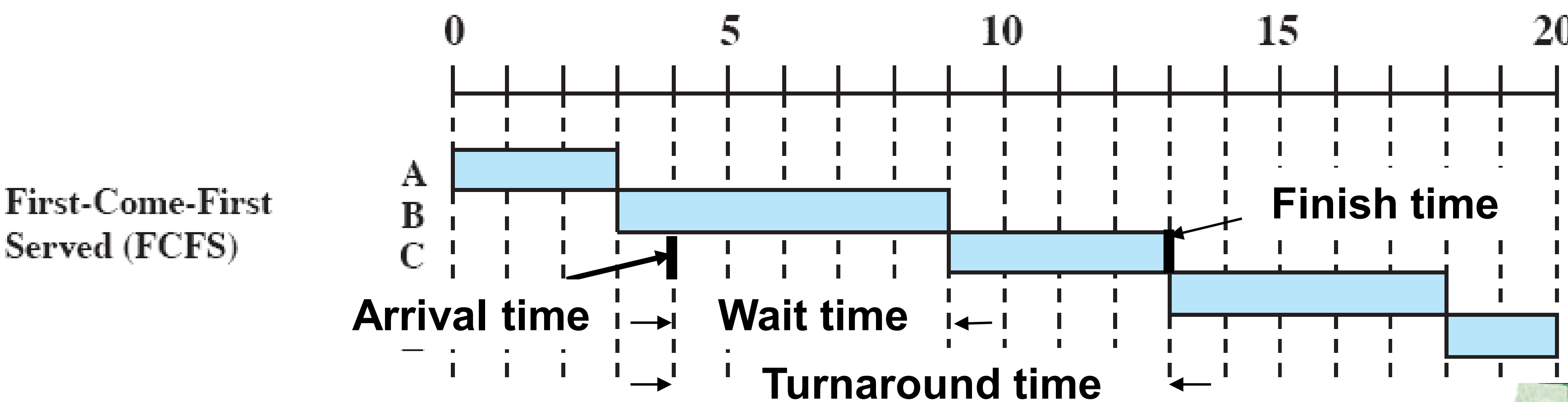
  - Service time represents the required total execution time
  - Calculate Turnaround time & Wait time

First-Come-First-Served

- Each process joins the Ready queue in a first-in-first-out manner
- When the current process ceases to execute, the process waiting the longest time in the Ready queue is selected

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



Turnaround time = Finish time – Arrival time

Wait time = Turnaround time - Service time

First-Come-First-Served

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

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

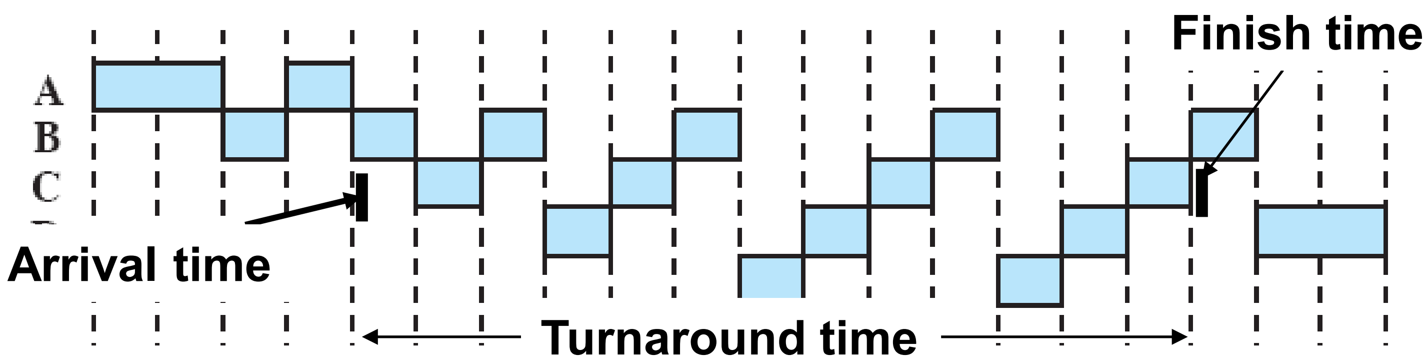
Round Robin

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

Table 9.4 Process Scheduling Example

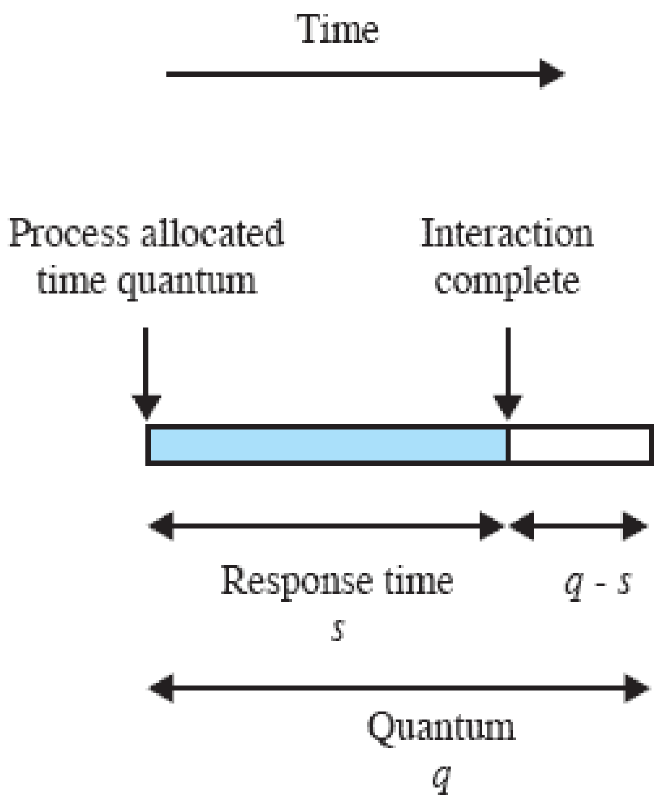
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

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



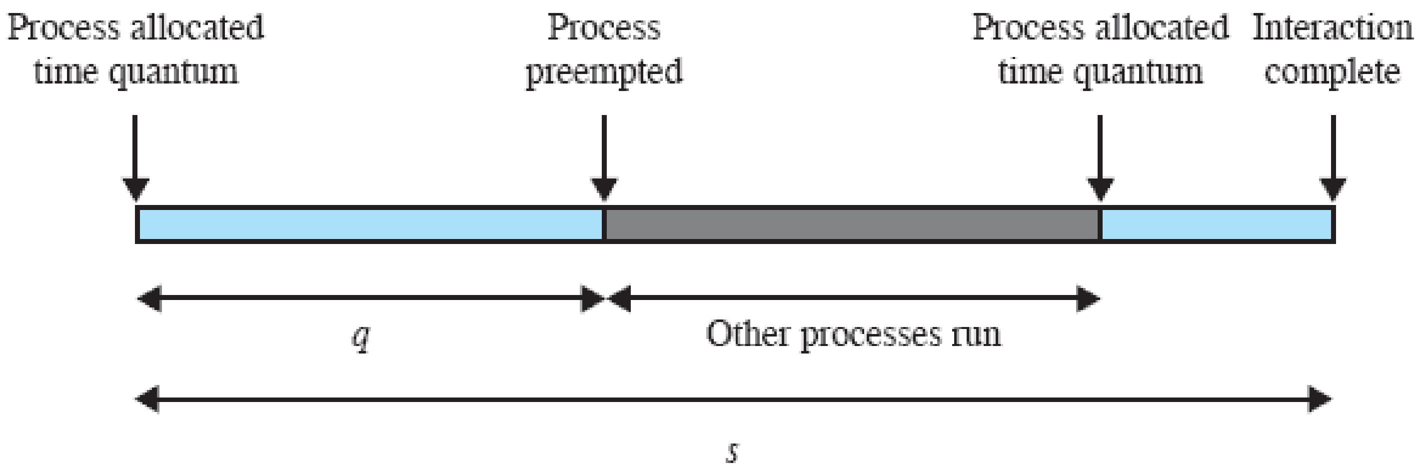
Wait time = ?

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’

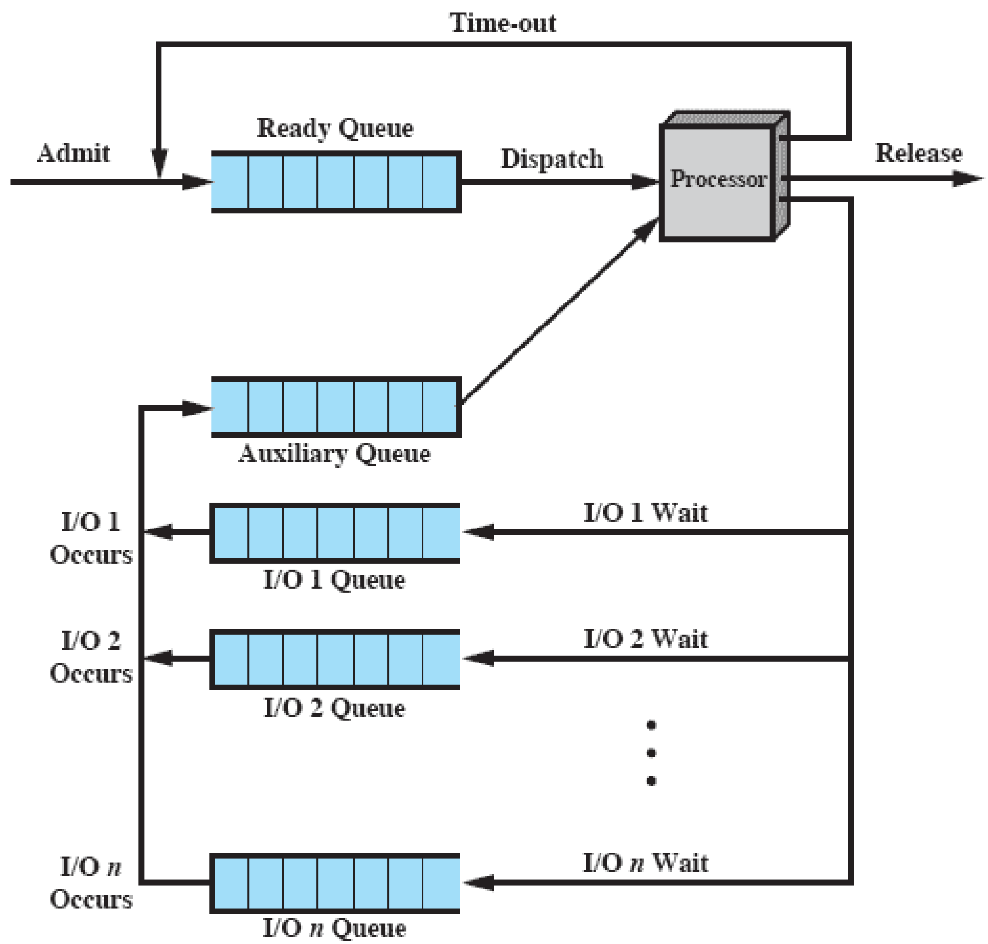


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

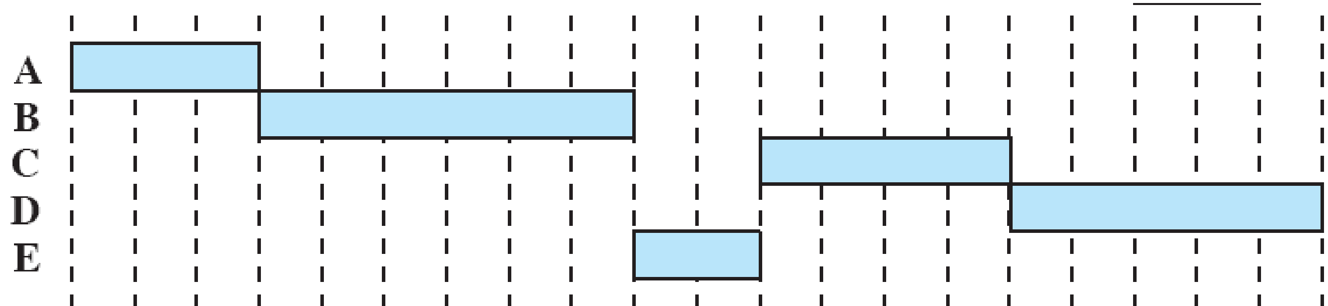
Shortest Job First (SJF)

- or Shortest Process Next (SPN)
- Nonpreemptive policy
- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes

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

Shortest Job First (SJF)



Shortest Job First (SJF)

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

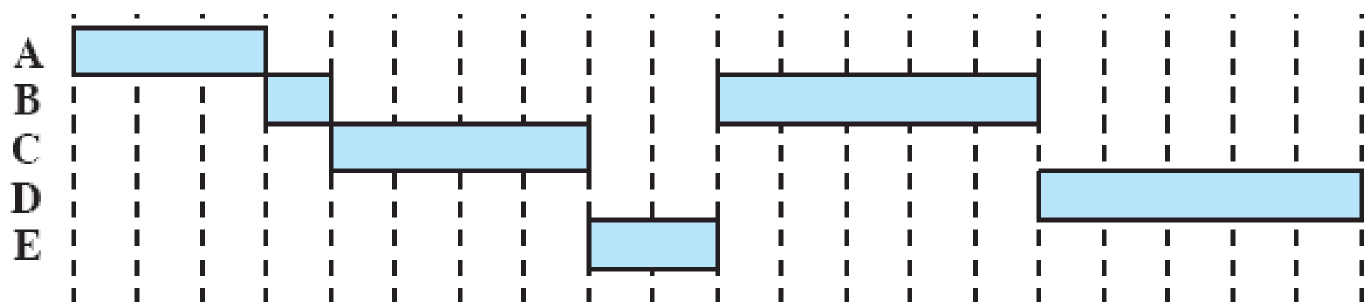
Shortest Remaining Time

- Preemptive version of shortest job first policy
- Scheduler always chooses the process that has the shortest expected remaining processing time

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

Shortest Remaining Time (SRT)



Shortest Remaining Time

- Must estimate processing time and choose the shortest
- Risk of starvation of longer processes
- Should give superior turnaround time performance to SPN because a short job is given immediate preference to a running longer job

Feedback Scheduling

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

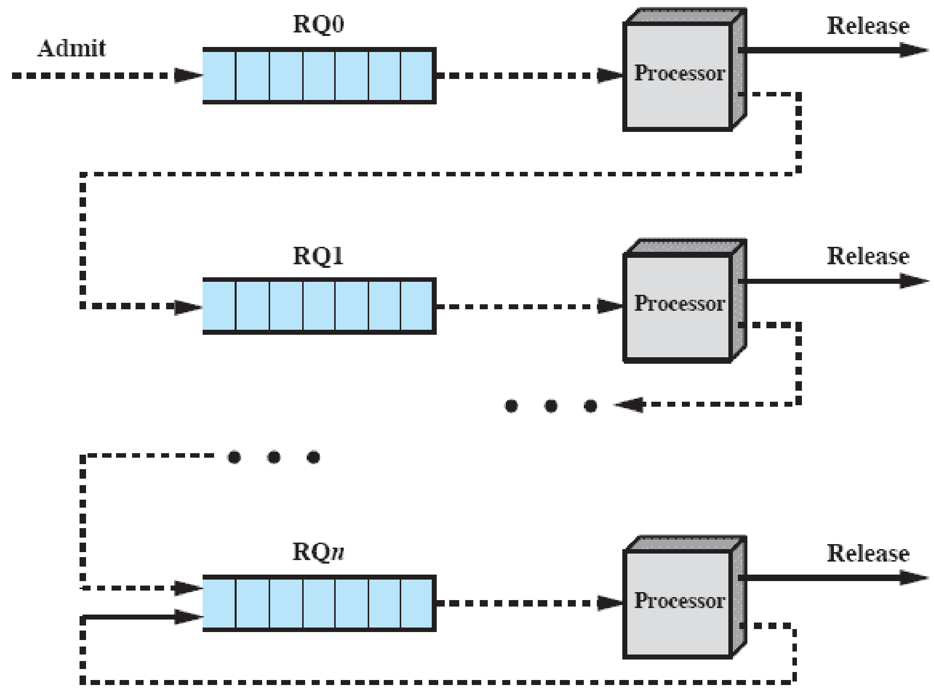


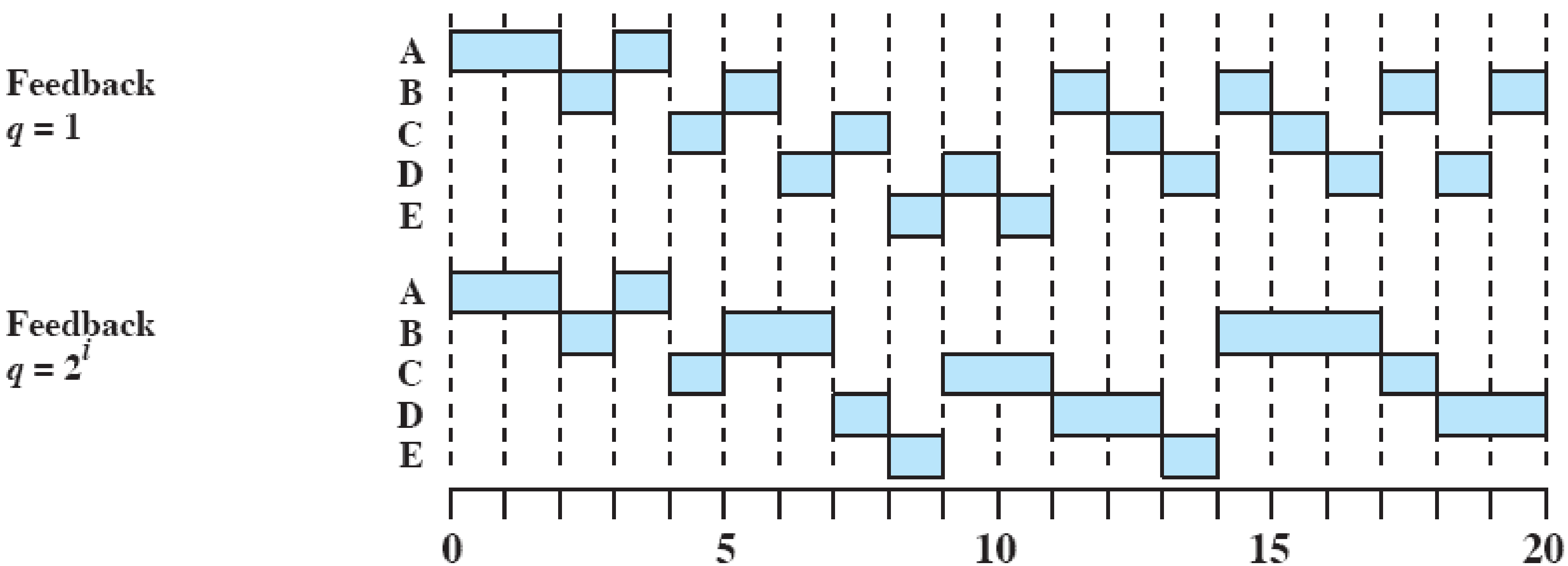
Figure 9.10 Feedback Scheduling

Feedback Performance

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

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



Priority Scheduling

- The CPU is allocated to the process with the highest priority
- Explicit priority
  - A priority number (integer) is associated with each process
- Implicit priority
  - SJF is a priority scheduling where priority is the predicted CPU time
- Problem: Starvation – low priority processes may never execute

Highest Response Ratio Next

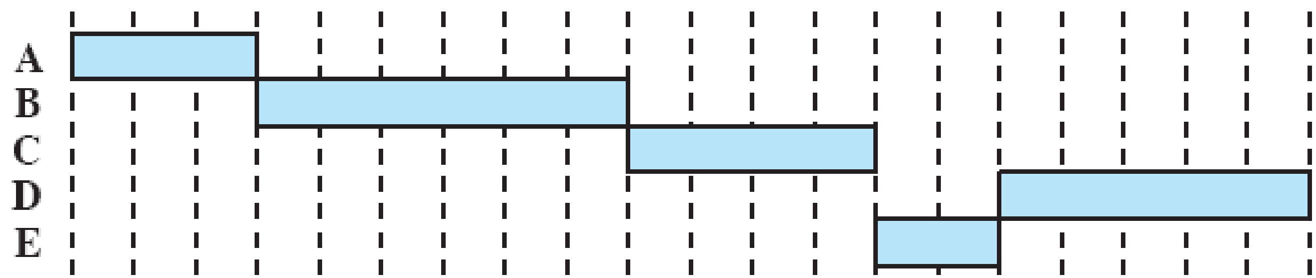
- Choose next process with the greatest ratio

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

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

Highest Response Ratio Next (HRRN)



Performance Comparison

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

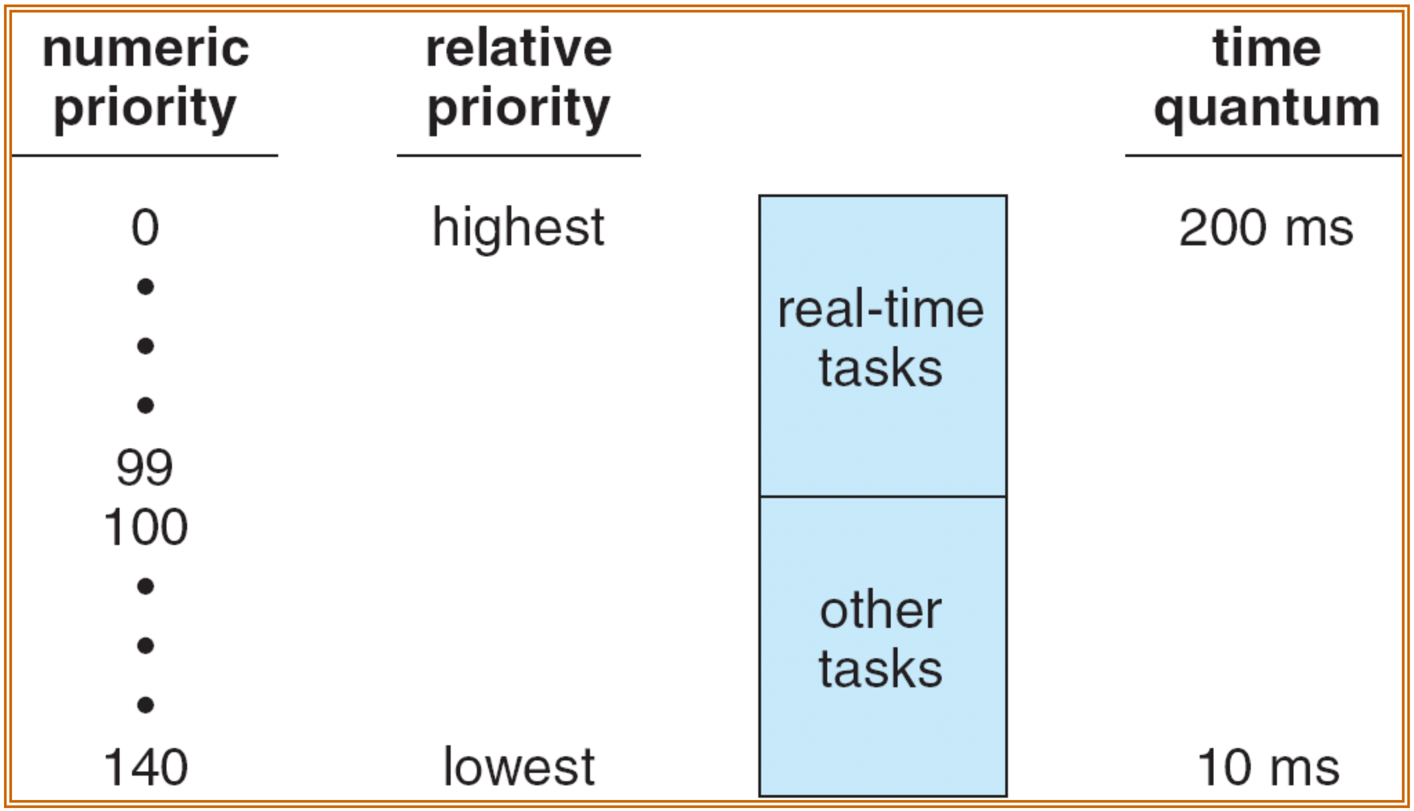
Contemporary Scheduling

- CPU sharing -- timer interrupts
  - Time quantum (or time slice) determined by interval timer
- With preemption
- Priority-based process (job) selection
  - Select the highest priority process
  - Priority reflects policy
- Usually a variant of Multilevel Feedback Queues

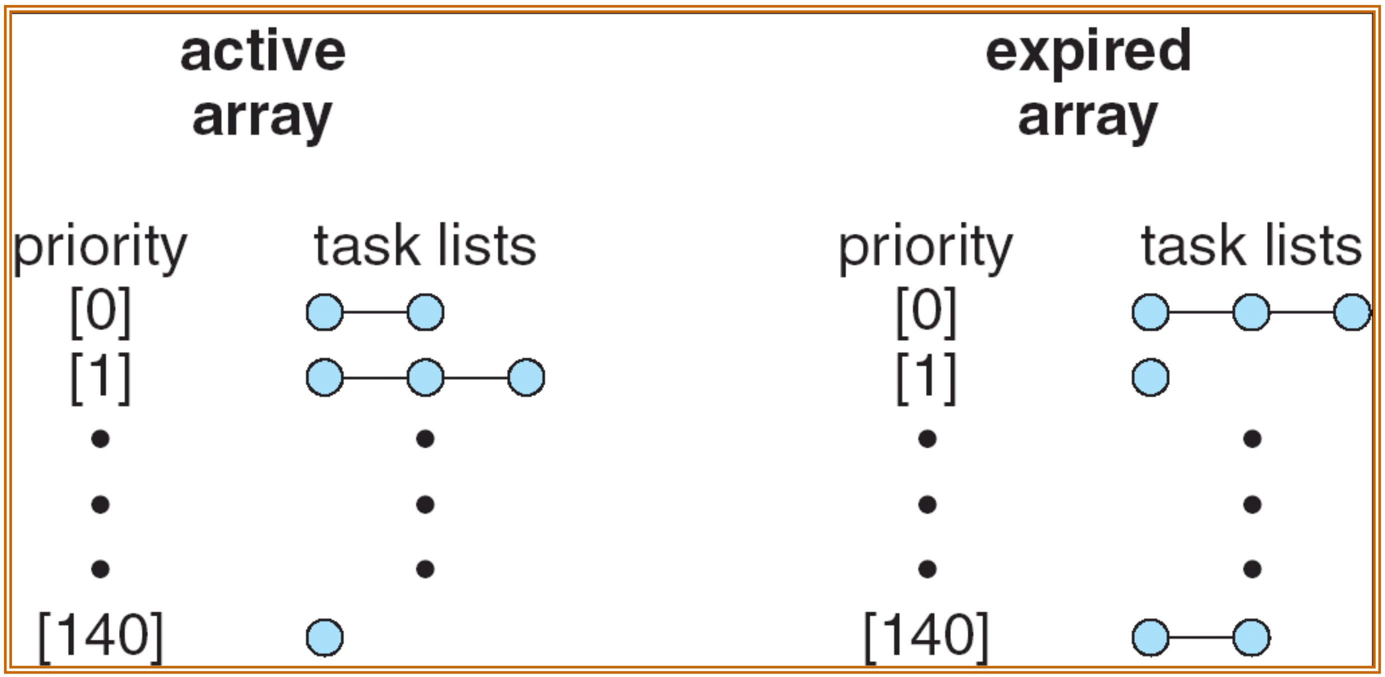
Linux Scheduling

- Linux O(1) scheduler
  - (before 2.6.23)
  - Two classes: real-time and others
  - Tasks have priorities ranging from 0 to 140
    - Real-time [0, 99]
    - Nice value (Others) [100, 140]
  - Each process is assigned a certain number of credits
  - Maintain two arrays for each CPU
    - Active array – processes with credits
    - Expired array – processes with no credits

The Relationship Between Priorities and Time-slice length



List of Tasks Indexed According to Priorities

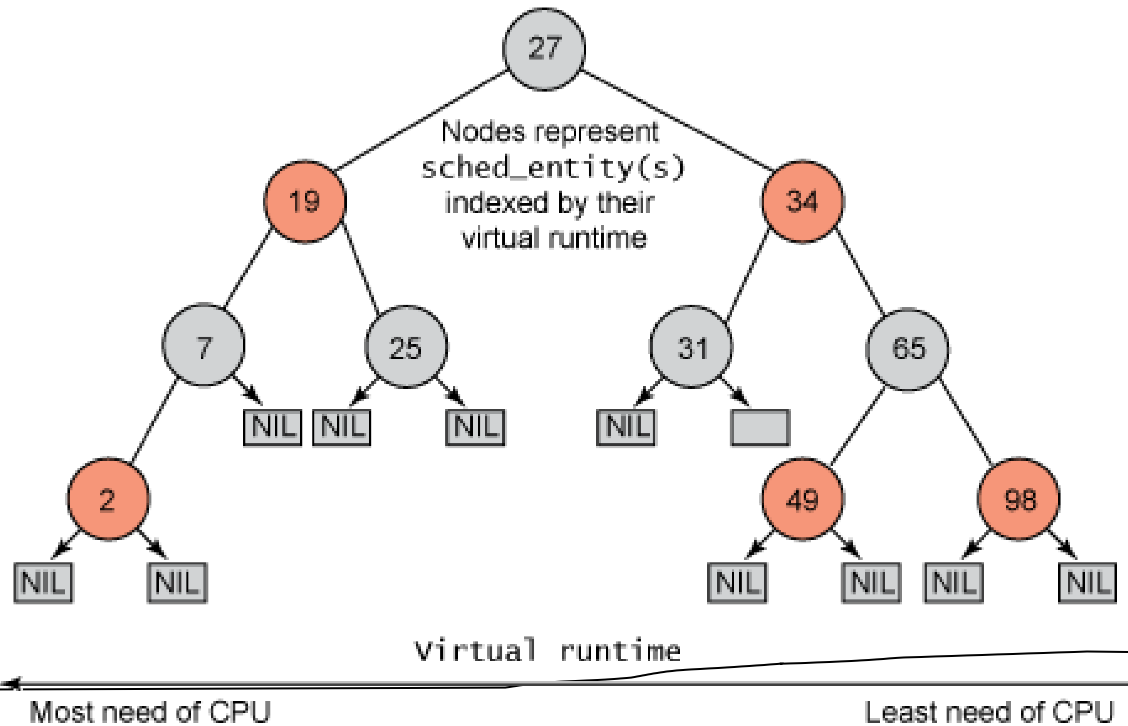


Linux Scheduling

- Completely Fair Scheduler (CFS)
  - More recent scheduler (2.6.23)
  - Try to improve the interactive performance for desktops
    - Select the process that has run least
    - Maintain a virtual runtime, used to account for how long a task has run, the amount of time actually spent weighted by its niceness
    - Each task will run for a "*timeslice*" proportional to its weight divided by total weight of all runnable tasks
    - A high priority tasks' virtual runtime grows slower than the virtual runtime of a low priority one

Linux Scheduling

– Stores processes in a red-black tree for selecting the process with the highest need for CPU
  - O(1) selection
  - O(log N) insertion and deletion

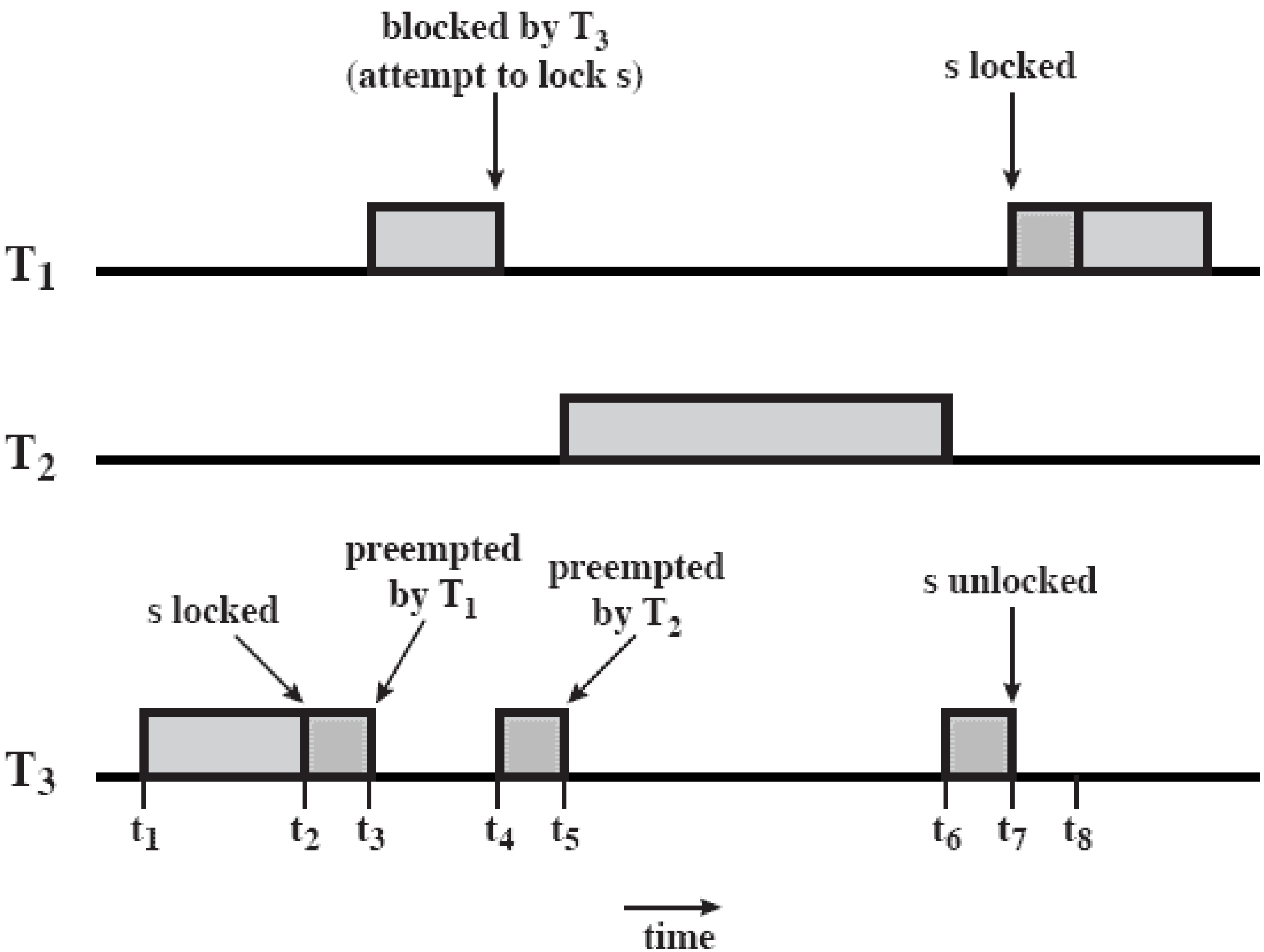


Priority Inversion

- Occurs when circumstances within the system forces a higher priority task to wait for a lower priority task
- Can occur in any priority-based preemptive scheduling scheme
- Particularly relevant in the context of real-time scheduling
  - Best-known instance involved the Mars Pathfinder mission

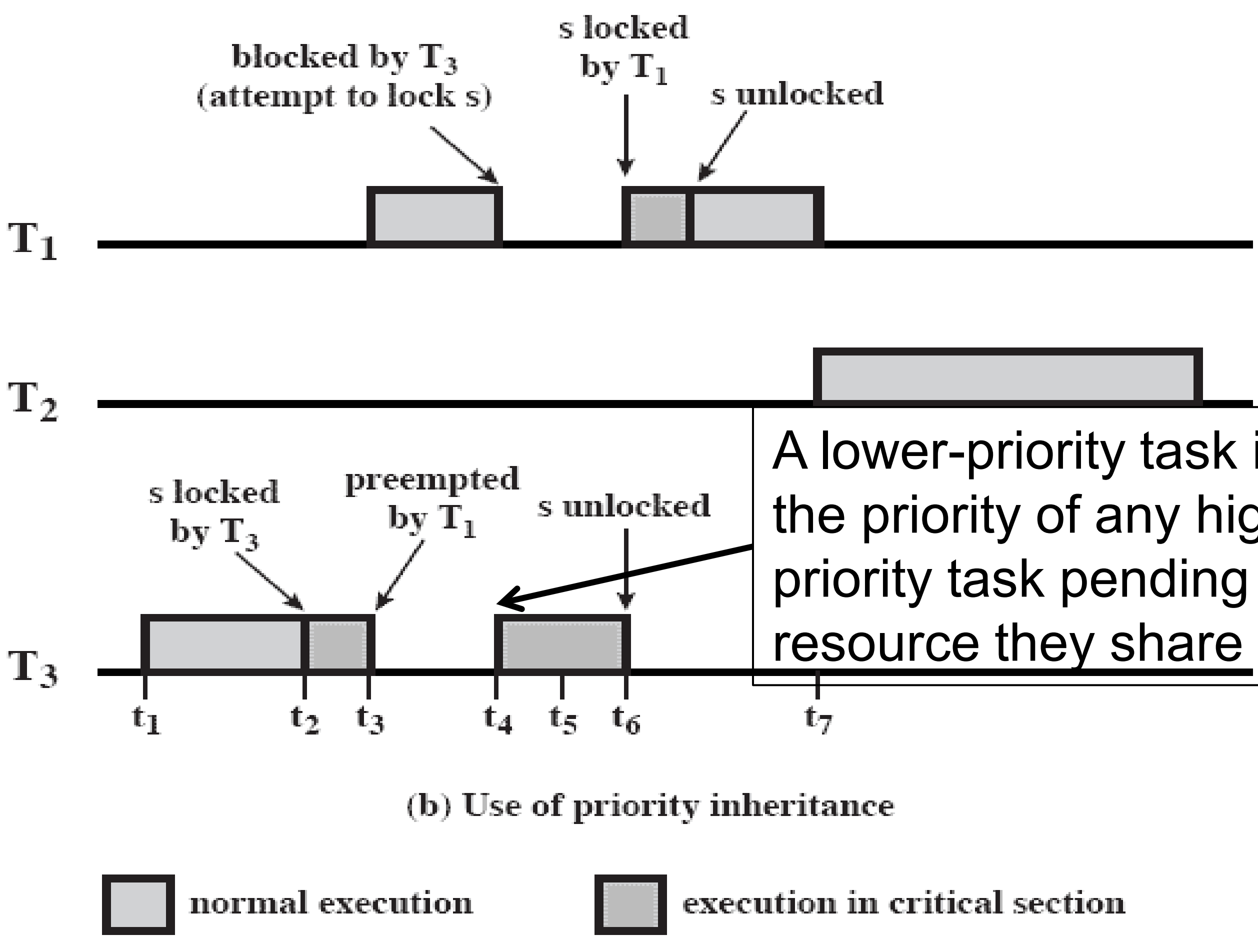
Unbounded Priority Inversion
• the duration of a priority inversion depends not only on the time required to handle a shared resource, but also on the unpredictable actions of other unrelated tasks

Unbounded Priority Inversion



(a) Unbounded priority inversion

Priority Inheritance



(b) Use of priority inheritance

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

Multicore CPU Scheduling

- CPU scheduling is more complex when multiple CPUs are available
- Symmetric multiprocessing (SMP) – each processor is self-scheduling, all processes/threads in common ready queue, or each has its own private queue of ready processes
  - Currently, most common
- SMP systems allow several threads to run concurrently multiple processors and thus complicate scheduling issues

Multicore Processor Scheduling – Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- Load balancing attempts to keep workload evenly distributed
- Push migration – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- Pull migration – idle processors pulls waiting task from busy processor

Algorithm Evaluation

- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- Deterministic modelling
  - Type of analytic evaluation
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
  - For each algorithm, calculate minimum average waiting time
  - Simple and fast, but applies only to those inputs

Algorithm Evaluation

- Queueing Models
  - Describes the arrival of processes, and CPU and I/O bursts probabilistically
    - Commonly exponential, and described by mean
    - Computes average throughput, utilization, waiting time, etc
  - Computer system described as network of servers, each with queue of waiting processes
  - Knowing arrival rates and service rates
  - Computes utilization, average queue length, average wait time, etc
  - often only approximations of real systems
  - the accuracy of the computed results may be questionable

Algorithm Evaluation

- Little's Formula
  - n = average queue length
  - W = average waiting time in queue
  - λ = average arrival rate into queue
  - Little's law – in steady state, processes leaving queue must equal processes arriving, thus:

    $n = \lambda \times W$

    - Valid for any scheduling algorithm and arrival distribution
  - For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds

Algorithm Evaluation

- Simulations
  - more accurate
  - Programmed model of computer system
  - A variable representing a clock
  - Gather statistics indicating algorithm performance
  - Data to drive simulation generated via
    - Random number generator according to probabilities
    - Distributions defined mathematically or empirically
    - Trace tapes record sequences of real events in real systems

Summary

- The operating system may make three types of scheduling decisions with respect to the execution of processes: long-term, medium-term and short-term
- From a user's point of view, response time is generally the most important characteristic of a system
- From a system point of view, throughput or processor utilization is important
- Algorithms:
  - FCFS, Round Robin, SPN, SRT, HRRN, Feedback