Uniprocessor Scheduling

Chapter 9

Background

- **Why we need multiprogramming OS?**
- **How does multiprogramming OS work?**
  - **The concepts of process switching and CPU scheduling DO appear!**

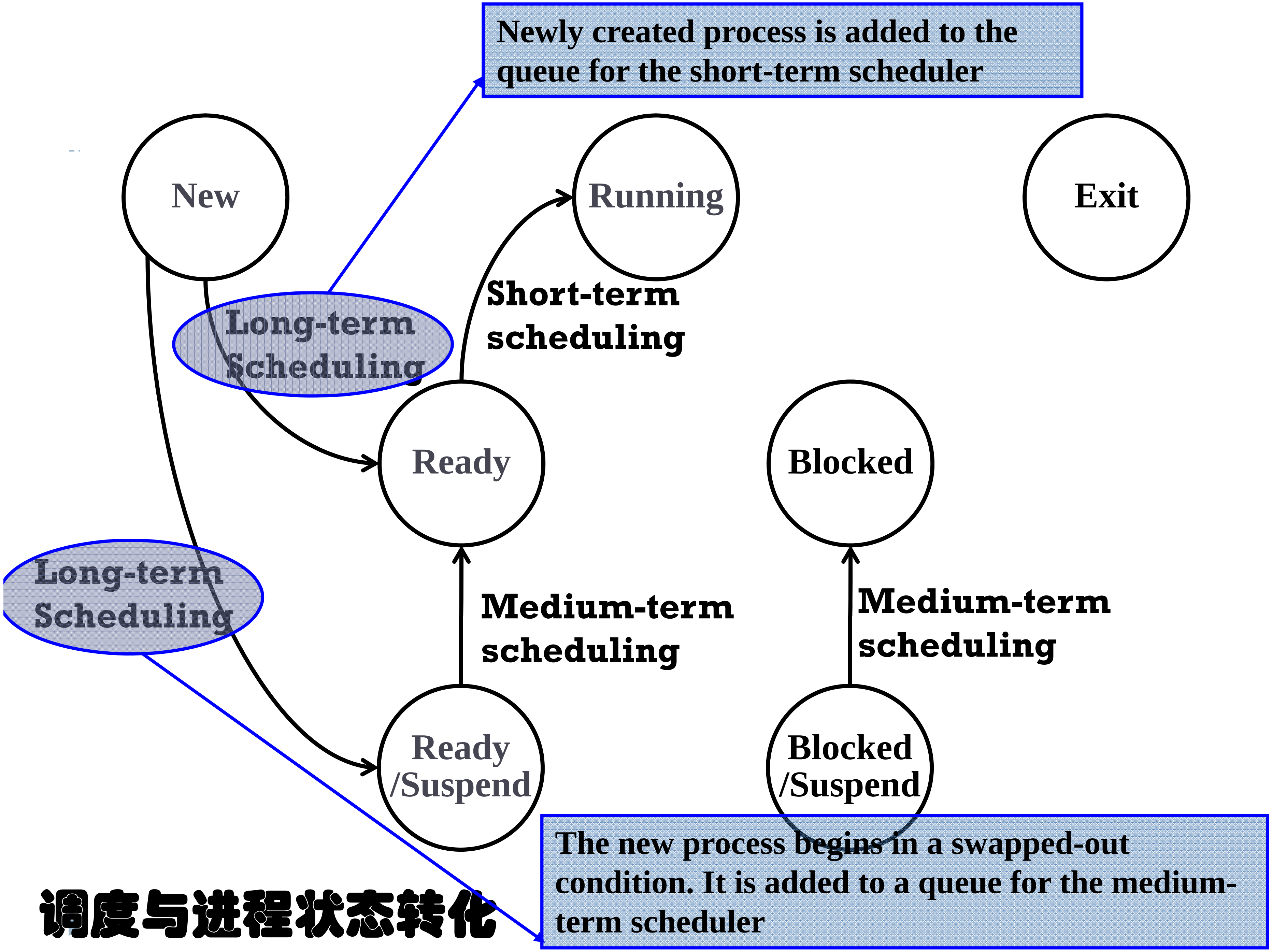
Organizations

- **Three-Level Scheduling**
  - Long-term (长程), medium-term (中程), short term (短程)
- **Short-Term Scheduling**
  - Short-Term Scheduling Criteria
  - The Use of Priorities
- **Short-Term Scheduling Policies**
  - First come first served, Round Robin, Shortest process next, Shortest remaining time, Highest response ratio next, and Feedback
  - Fair-share scheduling

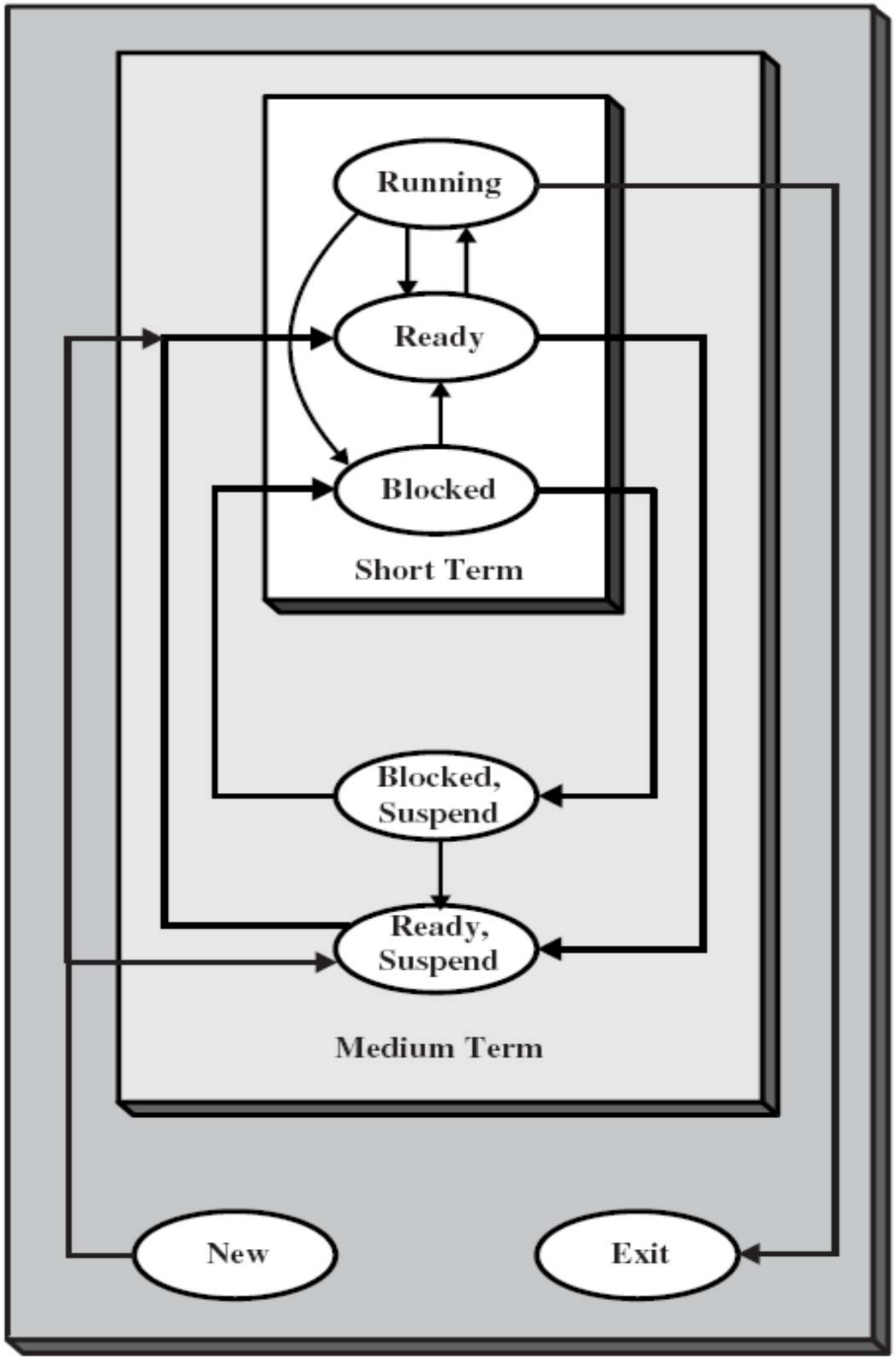
Types of Processor Scheduling

The Decisions to Make

- **Long-term scheduling (长程调度) is performed when a new process is created**
  - This is a decision to add a new process to the set of processes that are currently active
- **Medium-term scheduling (中程调度) is a part of the swapping function**
  - This is a decision to add a process to those that are at least partially in main memory and therefore available for execution
- **Short-term scheduling (短程调度) is that actual decision of which ready process to execute next**


Newly created process is added to the queue for the short-term scheduler
New
Running
Exit
Short-term scheduling
Long-term Scheduling
Ready
Blocked
Long-term Scheduling
Medium-term scheduling
Medium-term scheduling
Ready /Suspend
Blocked /Suspend
The new process begins in a swapped-out condition. It is added to a queue for the medium-term scheduler
调度与进程状态转化

Levels of Scheduling



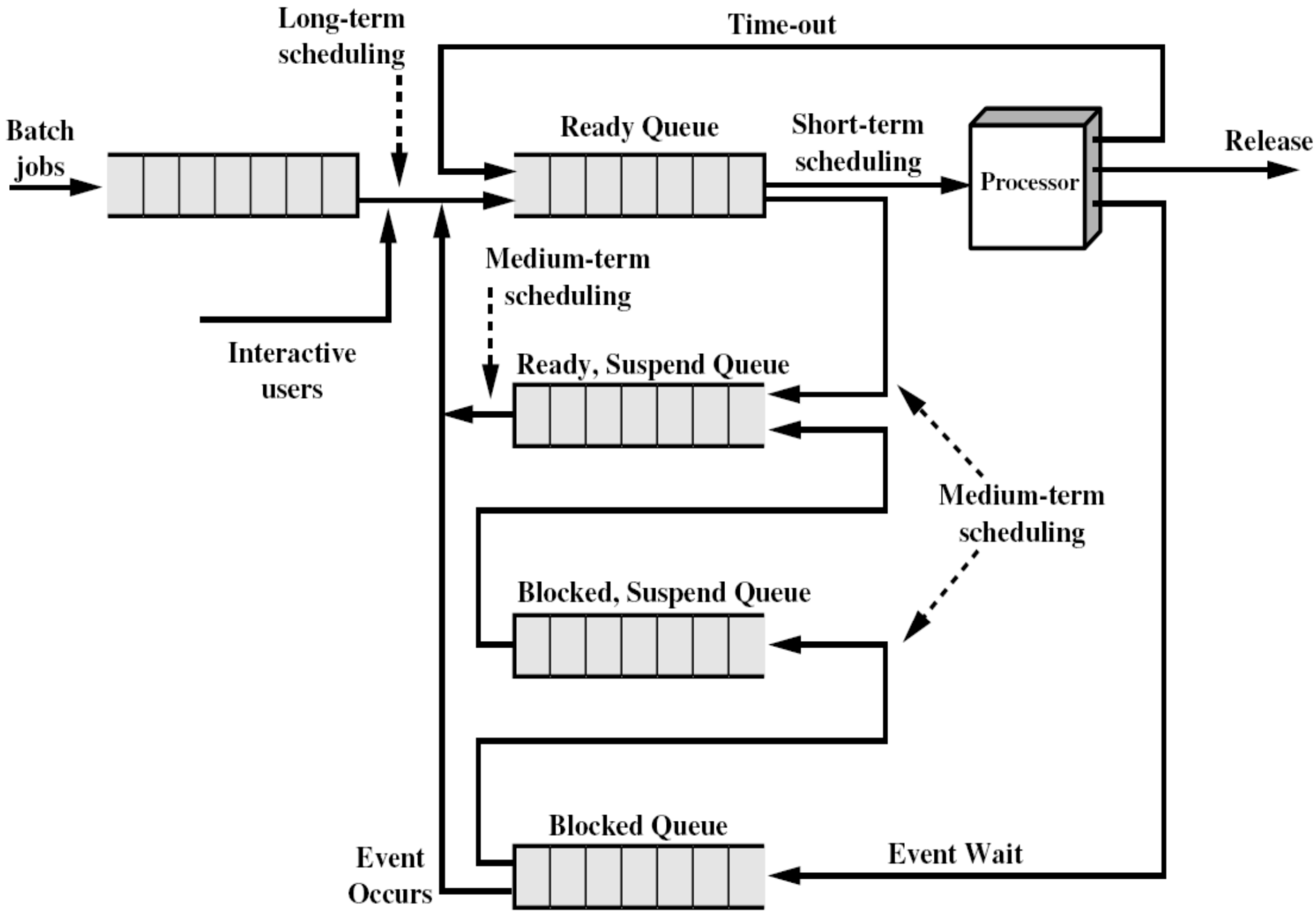


Figure 9.3 Queuing Diagram for Scheduling

Long-Term Scheduling

- **Determines which programs are admitted to the system for processing**
  - It controls the degree of multiprogramming
- **More processes, smaller percentage of time each process is executed**

Long-Term Scheduling in Batch System

- **In batch system, newly submitted jobs are routed to disk and held in a batch queue**
- **Two decisions involved in the long-term scheduler**
  - **When to create a new process?**
    - **Each time a job terminates, or when the fraction of time that the processor is idle exceeds a certain threshold**
  - **Which job to admit next?**
    - **First-come-first-served, or according to priority, expected execution time, and I/O requirements**

Long-Term Scheduling in Batch System

- **Processor-bounded processes (受处理机限制的进程)**
  - Mainly perform computational work and occasionally uses I/O devices
- **I/O-bounded processes (受I/O限制的进程)**
  - Spend more time using I/O devices than using processor
- **The scheduler may attempt to keep a mix of processor-bounded and I/O-bounded processes**
  - If such information is available

Long-Term Scheduling in Interactive Systems

- ***For interactive programs in a time-sharing system, a process request can be generated by the act of a user attempting to connect to the system***
- ***The OS will accept all authorized comers until the system is saturated.***

Medium-Term Scheduling

- **Part of the swapping function (交換功能)**
- **The swapping-in decision is based on *the* *need to manage the degree of multiprogramming***
- ***Memory management* is also an issue, on a system without virtual memory management**

Short-Term Scheduling

- CPU scheduler
  - Also known as the dispatcher (分派程序/分派器)
- Executes most frequently

- Its task: the scheduler selects one process from among the ready ones in memory, and allocates the CPU to it.

Short-Term Scheduling

CPU-I/O Burst Cycle
处理器-I/O脉冲循环

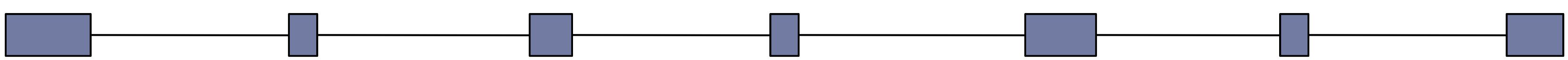
- **The success of CPU scheduling depends on the following observed property of processes:**
  - Process execution consists of a cycle of ***CPU execution*** and ***I/O wait***.
- **Process alternate between these two states**
  - Process execution begins with a CPU burst, followed by an I/O burst, then another CPU burst, then another I/O burst, and so on.
  - Eventually, the last CPU burst will end with a system call to terminate execution, rather than another I/O burst.

Process Behavior

- **Processes alternate bursting of computing with I/O requests.**

CPU-bounded process

I/O-bounded process



As CPUs get faster, processes tend to get more I/O-bound. This is because CPUs are improving much faster than disks

Nonpreemptive vs Preemptive Scheduling 非抢占式 VS 抢占式调度

- CPU scheduling decisions may take place under the following situations:
  1) When a currently running process switches to waiting state
  2) When the currently running process terminates
  3) When an interrupt occurs
  4) When a process switches from waiting state to ready state.

Nonpreemptive Scheduling:
Scheduling only occurs under these two situations

Nonpreemtive vs Preemptive Scheduling 非抢占式 VS 抢占式调度

- **Nonpreemptive (非抢占式)**
  - Once a process is in the running state, it will continue until it **voluntarily release the CPU**, or **terminates,** or **blocks itself for I/O**
- **Preemptive (抢占式)**
  - Currently running process may be interrupted and moved to the **Ready** state by the operating system
  - Allows for better service since any one process cannot monopolize the processor for very long

Nonpreemtive vs Preemptive Scheduling 非抢占式 VS 抢占式调度

- **Preemptive policies incur greater overhead than nonpreemptive ones, but may provide better service to the total population of processes**
  - Because they prevent one process from monopolizing the processor for very long.
- **Overhead can be reduced by using efficient process-switching mechanisms**
  - As much help from hardware as possible

Categories of Scheduling Algorithms

- **Different scheduling algorithms are needed in different environments.**
  - Different kinds of operating systems have different goals
- **Three Environments:**
  - Batch Systems
  - Interactive Systems
  - Real-time Systems.

Scheduling Algorithm Goals (1)

- ***All systems: Fairness, Policy enforcement, Balance.***
- ***Batch Systems: Throughput, Turnaround time, and Processor Utilization***
- ***Interactive Systems: Response time, Proportionality***
- ***Real-time systems: Meeting Deadlines***

Scheduling Algorithm Goals (2) All Systems

- ***Fairness***
  - Comparable processes should be treated the same, and *no process should suffer starvation*
- ***Enforcing priorities***
  - *favor higher-priority processes, when processes are assigned priorities*
- ***Balancing Resources*:**
  - *keep the resources of the system busy*

Scheduling Algorithm Goals (3) Batch Systems

- ***Throughput***
  - the number of processes completed per unit of time
- ***Turnaround Time***
  - The interval time between the submission of a process and its completion
- **Processor Utilization**
  - The percentage of time that the processor is busy.
  - Important for expensive shared system, while less important for single-user system or real-time system

Scheduling Algorithm Goals (4) Interactive Systems

- ***Response Time*** **(the most important for interactive systems)**
  - The time from the submission of a request until the response begins to be received
- ***Predictability/Proportionality*****: (also for batch systems)**
  - A given process should run in about the same amount 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

Scheduling Algorithm Goals (5) Real-time Systems

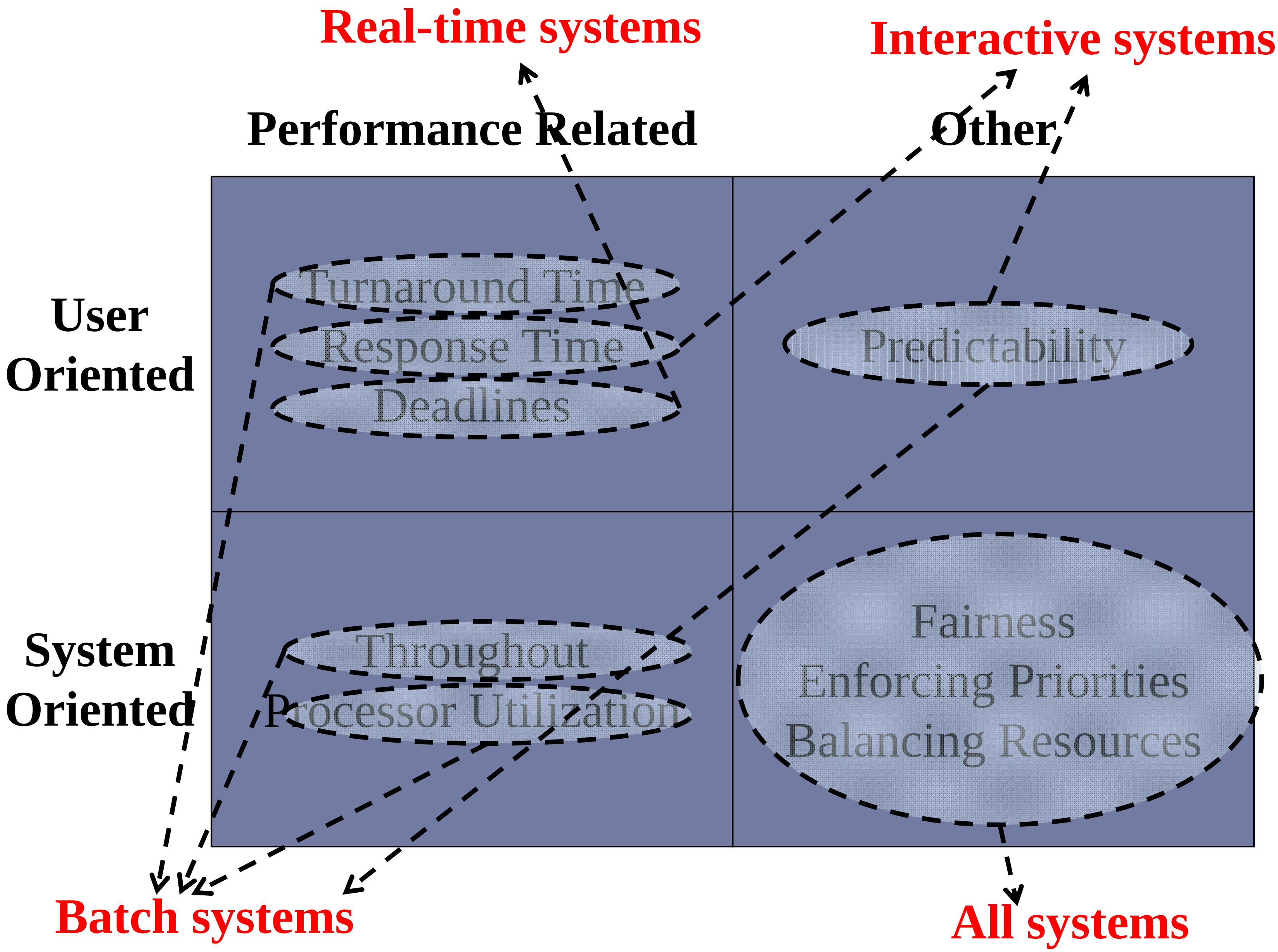
- ***Deadlines* (期限)**
  - **When process completion deadlines can be specified, the scheduling discipline should subordinate other goals to that of maximizing the percentage of deadlines met.**
  - **For example:**
    A device is producing data at a regular rate. Failure to run a data-collection process may result in data loss.

Scheduling Algorithm Goals (6)
User-Oriented v.s System-Oriented

- **User-oriented criteria relate to the behavior of the system as perceived by the individual user or process**
  - User-oriented criteria are important on virtually all systems
- **System-oriented criteria focus on effective and efficient utilization of the processor**
  - System-oriented criteria are generally of minor importance on single-user systems

Scheduling Algorithm Goals (7) Performance Related or Not

- **Performance-related criteria are quantitative and measurable. (such as response time and throughput)**
- **Criteria that are not performance related are either qualitative in nature or do not lend themselves to measurement and analysis**


Real-time systems
Interactive systems
Performance Related
Other
User Oriented
System Oriented
Turnaround Time
Response Time
Deadlines
Predictability
Throughout
Processor Utilization
Fairness
Enforcing Priorities
Balancing Resources
Batch systems
All systems

Interdependence in These Criteria

- **These criteria are interdependent and it is impossible to optimize all of them simultaneously**
  - **For example, providing good response time may require frequent switches between processes**
    - **Frequent switches increases the overhead of the system, and then reduces the throughout.**

The Use of 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
- Lower-priority may suffer starvation
  - Solution: allow a process to change its priority based on its age or execution history

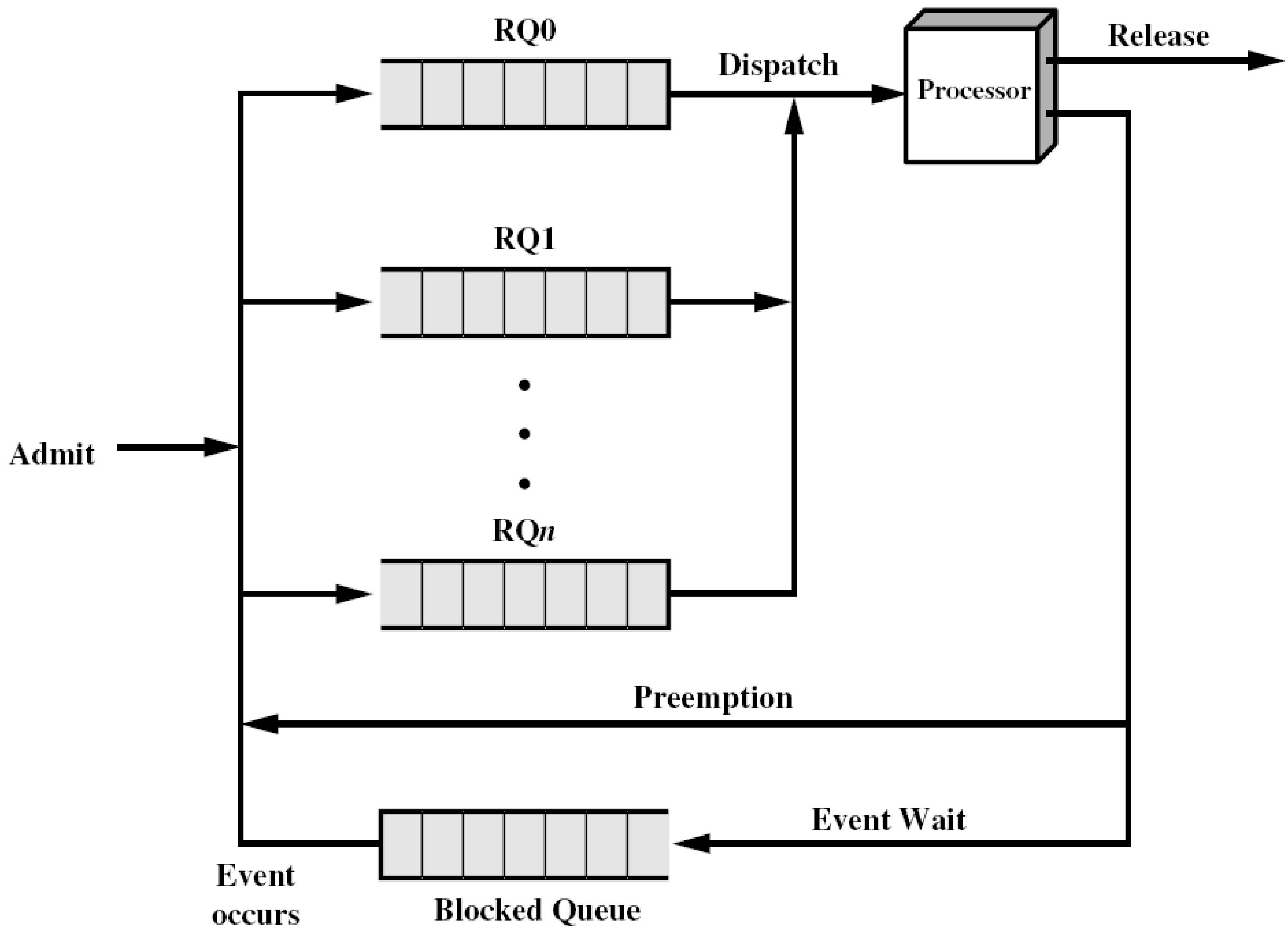


Figure 9.4 Priority Queuing

Selection Function

- The selection function determines which process, among ready processes, is selected next for execution
- This function may be based on
  - Priority,
  - Resource requirements, or
  - Execution characteristics
    - *w*=time spent in system so far, waiting and executing
    - *e*=time spent in execution so far
    - *s*=total service time required by the process, including e.

Short-Term Scheduling Policies

Process Scheduling Example

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

First-Come-First-Served (1) 先来先服务-What is it?

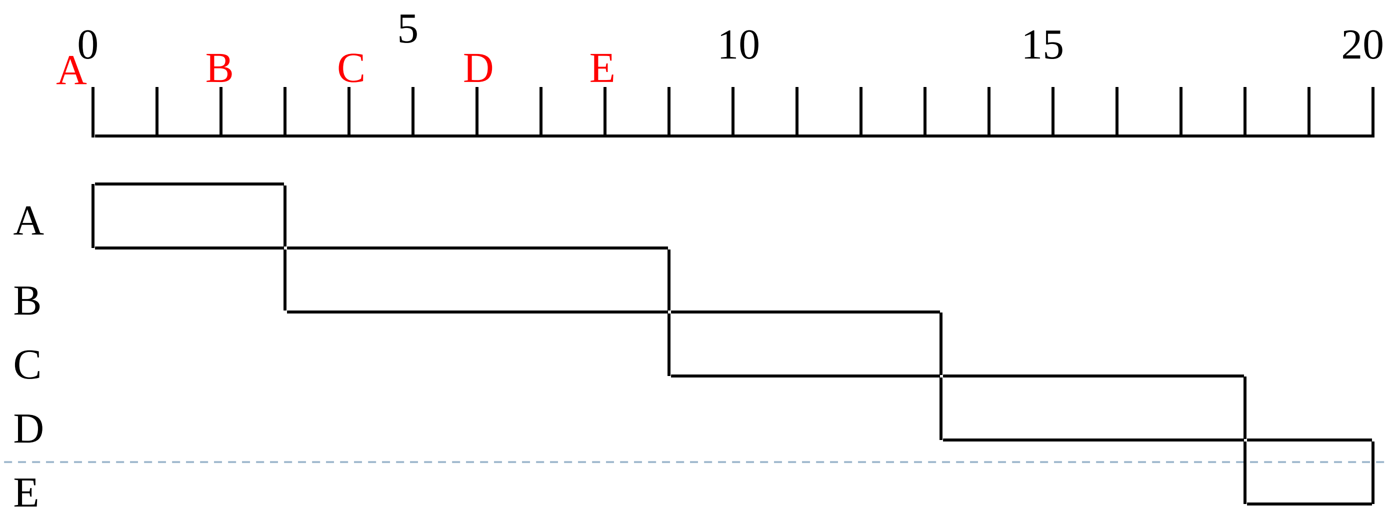
- First-Come-First-Served (FCFS) also known as First-In-First-Out (FIFO)
  - As each process becomes ready, it joins the Ready queue
  - When the current process ceases to execute, **the process that has been in the Ready queue the longest** is selected
- It is the simplest scheduling policy

First-Come-First-Served (2)
(Normalized) Turnaround Time

- ***Turnaround Time* (TAT) 周转时间**
  - **In queuing model, it is the residence time T_r, or *The* total time that the item spends in the system (*waiting time plus service time*)**
- ***Normalized Turnaround Time* 归一化周转时间**
  - **It is the *ratio of turnaround time to service time.***
    - **The minimum possible value is 1.0**
    - **Increasing values correspond to a decreasing level of service**
    - **The longer the process execution time, the greater the absolute amount of delay that can be tolerated**
  - **This figure is more useful**

First-Come-First-Served (3)
The Example

	Process	A	B	C	D	E	Mean
	Arrival Time	0	2	4	6	8	
	Service Time (T_s)	3	6	4	5	2	
FCFS	Finish Time	3	9	13	18	20	
	Turnaround Time (T_r)	3	7	9	12	12	8.60
	T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56



First-Come-First-Served (4) Problems

- ***FCFS performs much better for long processes than short ones.***

- ***FCFS tends to favor (偏爱) CPU-bound processes over I/O-bound processes***
  - **FCFS may result in inefficient use of both the processor and the I/O devices. Why?**

First-Come-First-Served (5) Conclusions

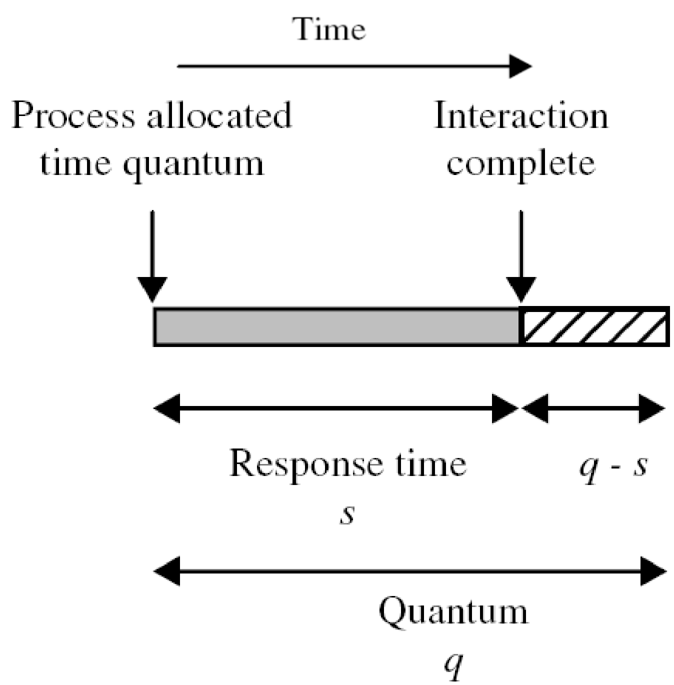
- **Advantage: It is the simplest, and thus is easiest for understanding and implementation**
- ***FCFS is not an attractive alternative on its own for a single-processor system***
    - **It is often combined with a priority scheme to provide an effective scheduler. (such as the feedback scheduler to be discussed later)**

Round-Robin 轮转(1)
What is it?

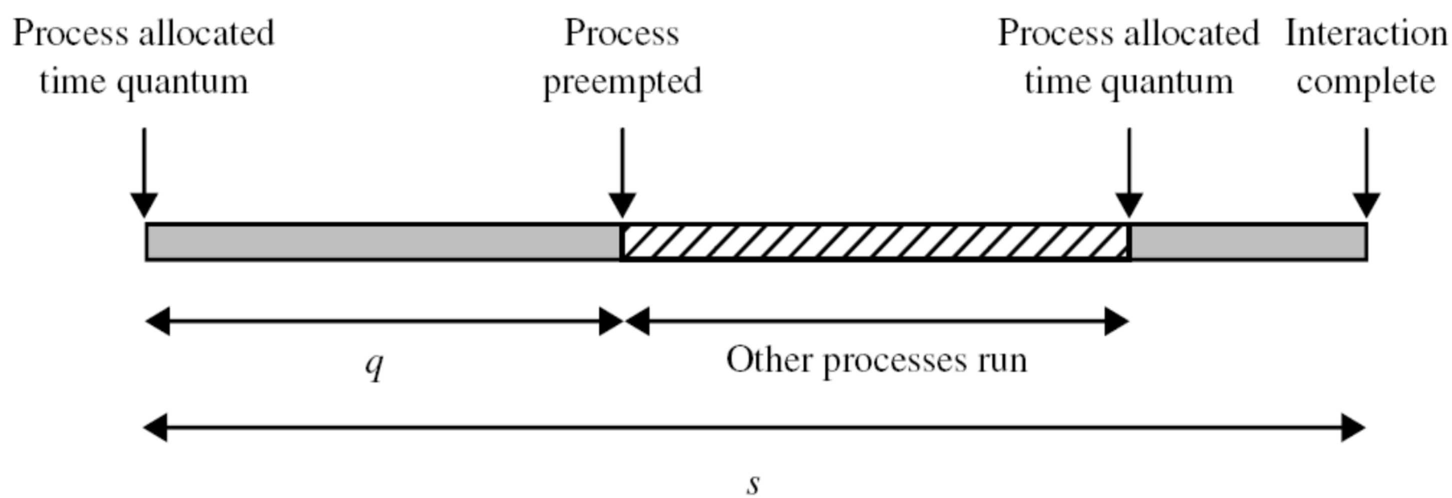
- **Round robin is a straightforward way to *reduce the penalty that short processes suffer with FCFS***
  - With round robin, Clock interrupt is generated at periodic intervals
  - When an clock interrupt occurs, the currently running process is placed in the Ready queue
    - Next ready job is selected
- **Known as time slicing (时间分片)**

Round-Robin (2)
Length of Time Quantum

- The ***length of time quantum* (时间片的长度)** is the key design issue with round robin
  - If the quantum is very short, then short processes will move through the system relatively quickly
    - **Short quantum should be advocated**
  - However, there is processing **overhead** involved in handling clock interrupts and performing the scheduling and dispatching functions
    - **Short quantum should be avoided (otherwise, too many process switches)**



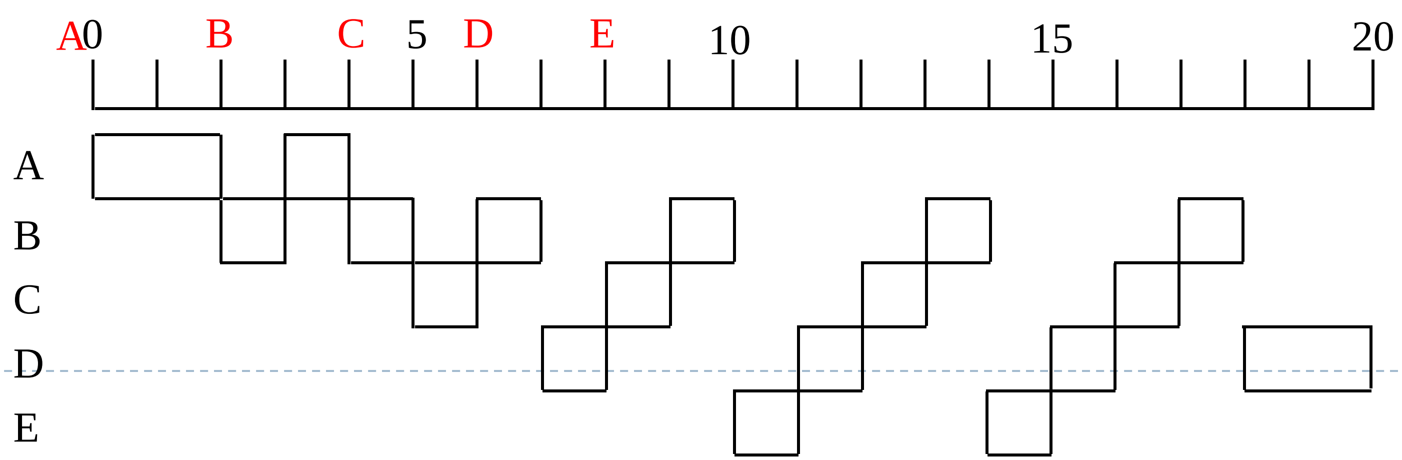
(a) Time quantum greater than typical interaction



(b) Time quantum less than typical interaction

Round-Robin (3)
The Example

	Process	A	B	C	D	E	Mean
	Arrival Time	0	2	4	6	8	
	Service Time (T_s)	3	6	4	5	2	
RR $q = 1$	Finish Time	4	18	17	20	15	
	Turnaround Time (T_r)	4	16	13	14	7	10.80
	T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$	Finish Time	3	17	11	20	19	
	Turnaround Time (T_r)	3	15	7	14	11	10.00
	T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71



Round-Robin (4)

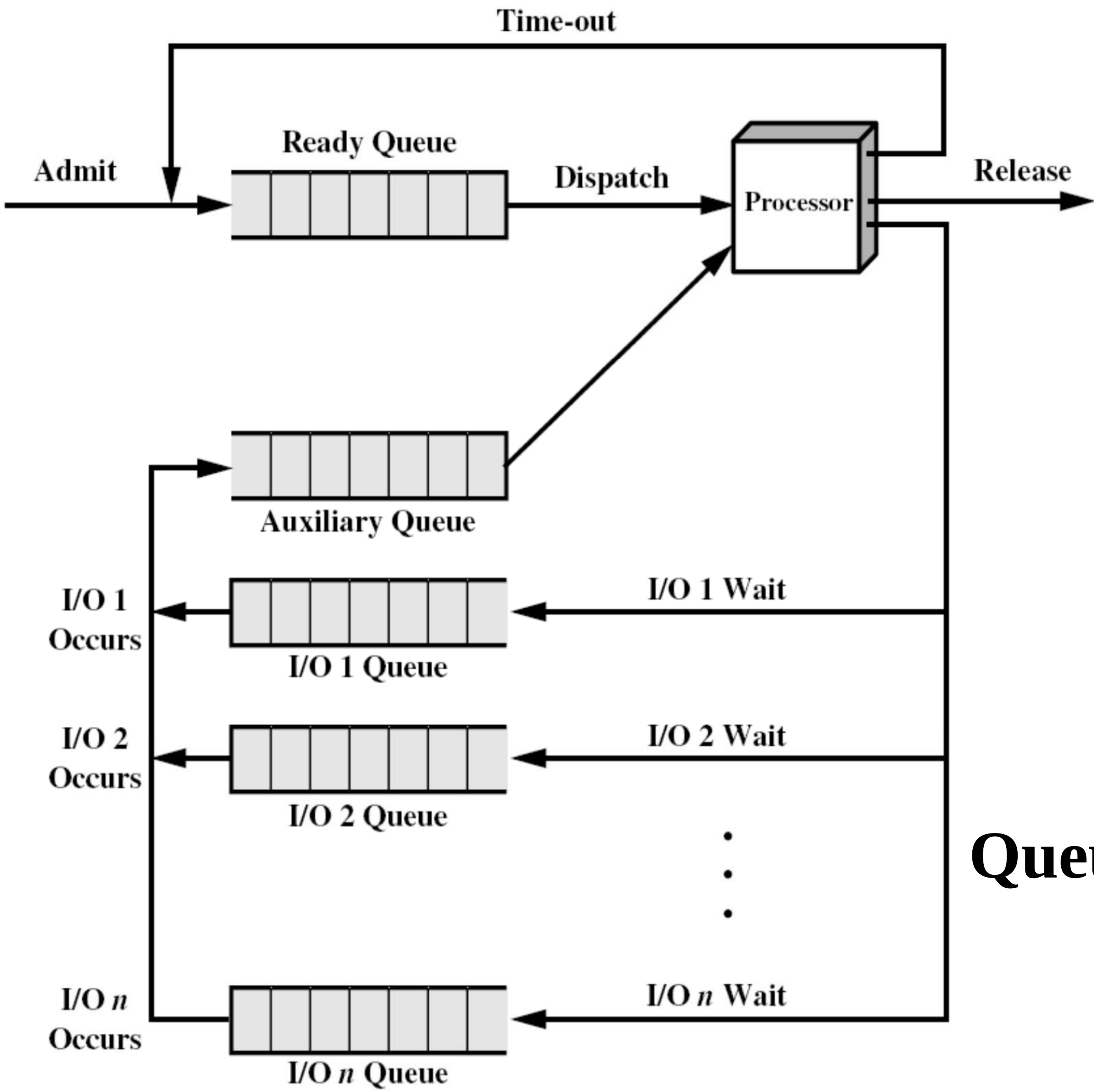
- **Round-robin is particularly effective in a general-purpose *time-sharing system* or *transaction processing system*.**

Round Robin (4)
Unfairness

- **Generally, An I/O-bound process has a shorter processor burst than a processor-bound process**
  - An I/O-bound process uses a processor for a short period and then is blocked for I/O
  - it waits for the I/O operation to complete and then joins the ready queue
- **Results**
  - Poor performance for I/O-bound processes
  - Inefficient use of I/O devices
  - An increase in the variance of response time

Round Robin (5)
A Refinement: VRR

- **Virtual round robin (VRR) is a refinement**
- **An FCFS auxiliary queue is introduced**
  - Processes are moved to this queue after being released from an I/O block.
  - When a dispatching decision is to be made, processes in auxiliary queue get preference over those in the main Ready queue.



Queuing Diagram for VRR

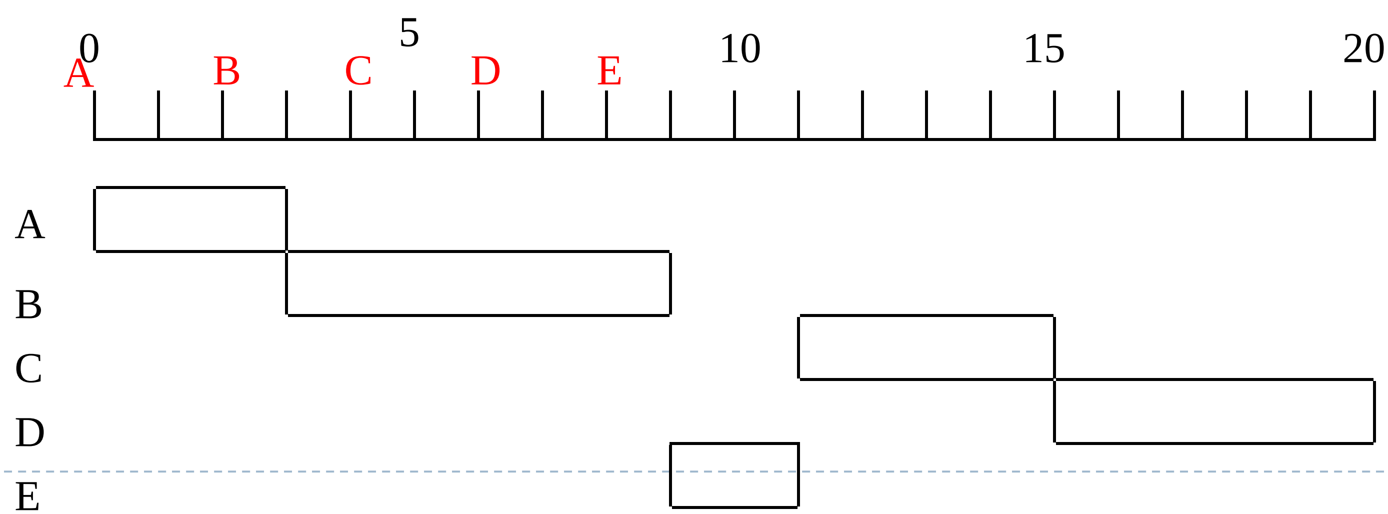
(Nonpreemptive) Shortest Process Next (1)
What is it?

Also called "Shortest Job First"

- **Shortest Process Next (SPN) policy is another approach to reducing the bias in favor of long processes inherent in FCFS.**
- **SPN is a *nonpreemptive* policy**
  - **The process with shortest expected processing time is selected next**
  - **That is to say, a short process can jump to the head of the queue past longer jobs**

(Nonpreemptive) Shortest Process Next(2) The Example

							Mean
	Process	A	B	C	D	E	
	Arrival Time	0	2	4	6	8	
	Service Time (T_s)	3	6	4	5	2	
SPN	Finish Time	3	9	15	20	11	
	Turnaround Time (T_r)	3	7	11	14	3	7.60
	T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84



(Nonpreemptive) Shortest Process Next(3) Benefits

- **Compared with FCFS, overall performance is significantly improved.**
  - It is optimal among all the nonpreemptive scheduling strategies when all the jobs are available simultaneously!!
  - How to prove it?
- **It is not necessarily optimal when all the jobs do not arrive simultaneously.**
  - Example: five jobs A through E with service times of 2, 4, 1, 1, and 1, respectively. Their arrival times are 0, 0, 3, 3, and, 3.
  - What is the order of running jobs with SPN?
  - How about the order of BCDEA?

(Nonpreemptive) Shortest Process Next (4)
The required processing time

- **For batch jobs, the programmer is required to estimate the value and supply it to OS.**
  - If this value is substantially under the actual running time, the system may abort the job
- **In a production system, the OS may keep a running average (运行平均值) of each "burst" for each process**

(Nonpreemptive) Shortest Process Next (5) Disadvantages

- **Possibility of *starvation for longer processes* exists with SPN**
  - *Predictability of longer processes is reduced*
- **The reduction of the bias in favor of longer jobs is not enough**
  - Because of the *lack of preemption*
  - Refer to the table on top of the page 404.
- **With SPN policy, we need to know or at least estimate the required processing time of each process**

The simplest calculation : $S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i$

where T_i = processor execution time for the ith instance of this process

S_i = predicted value for the ith instance

S_1 = predicted value for the first instance : not calculated

To avoid recalculating the entire summation each time, it can be rewritten as :

$$S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n$$

Each instance is given equal weight!!
How about giving greater weight to more recent instances, because they are more likely to reflect future behavior.

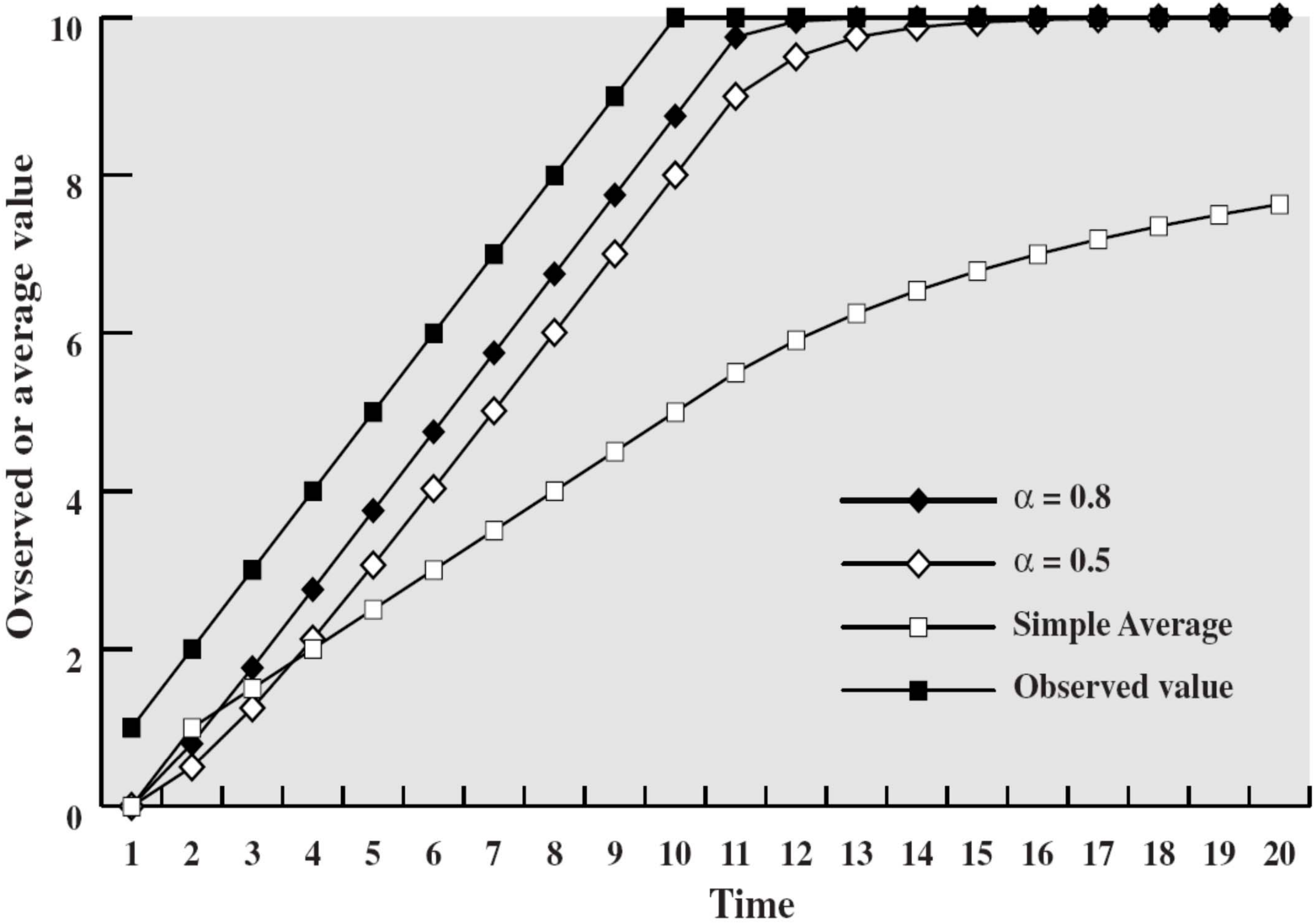
Exponential averaging (指数平均法): a common technique for predicting a future value on the basis of a time series of past value

$$S_{n+1} = \alpha T_n + (1-\alpha) S_n$$

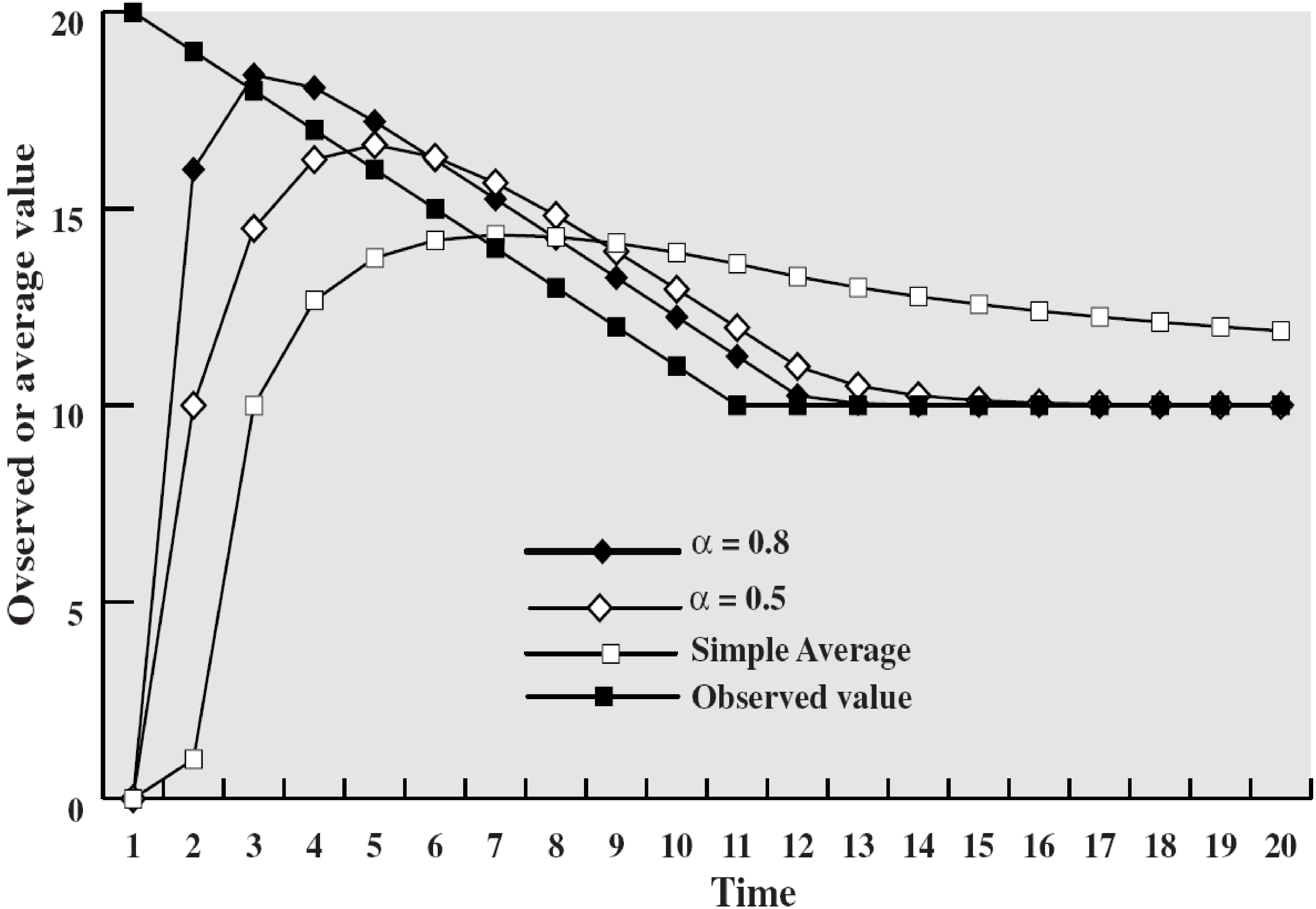
α: constant weighting factor (0< α <1) 常数加权因子

To see it more clearly, above formula can be rewritten as:

$$S_{n+1} = \alpha T_n + \alpha(1-\alpha) T_{n-1} + \cdots + (1-\alpha)^i \alpha T_{n-i} + \cdots + (1-\alpha)^n S_1$$



(a) Increasing function



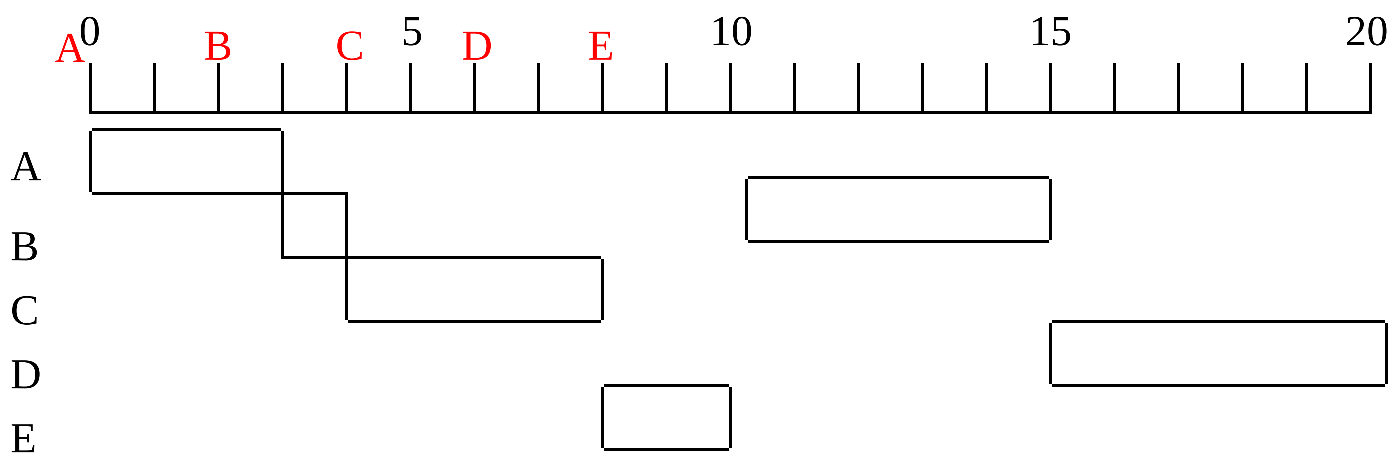
(b) Decreasing function

Shortest Remaining Time (1)

- A preemptive version of SPN
  - *When a new process joins the ready queue*, it may in fact have a shorter remaining time than the currently running process
  - SRT does not have the bias in favor of long processes found in FCFS
  - Compared with round robin, SRT does not generate additional interrupts, reducing overhead.
  - SRT should give superior turnaround time performance to SPN, because a short job is given immediate preference to a running longer job.
- Elapsed time must be recorded

Shortest Remaining Time (2)
The Example

	Process	A	B	C	D	E	Mean
	Arrival Time	0	2	4	6	8	
	Service Time (T_s)	3	6	4	5	2	
SRT	Finish Time	3	15	8	20	10	
	Turnaround Time (T_r)	3	13	4	14	2	7.20
	T_r/T_s	1.00	2.17	1.00	2.80	1.00	1.59



- ***Three shortest processes (A, C, E) all receive immediate service, yielding a normalized turnaround time for each of 1.0***

Priority Scheduling I
优先级调度

- **Priority scheduling algorithm**
  - A priority is associated with each process
  - The CPU is allocated to the process with the highest priority
  - Equal-priority processes are scheduled in FCFS order
- **Preemptive and non-preemptive SPF algorithms are special cases of priority scheduling**
  - The priority is the inverse of the (predicted) next CPU burst

Priority Scheduling II

- **Priorities can be assigned to processes statically or dynamically.**
  - Some processes are more important than others
  - To prevent high-priority processes from running indefinitely, the scheduler may decrease the priority of the currently running process at each clock interrupt
  - Priorities can be assigned dynamically by the system to achieve certain system goals
    - Highly I/O bound processes should be given higher priorities

Priority Scheduling III

- **Priority Scheduling can be either preemptive or nonpreemptive.**
  - **When a process arrives at the ready queue:**
    - **A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process**
    - **A nonpreemptive priority scheduling algorithm will simply put the new process into the ready queue**

Highest Response Ratio Next (1) What is it?

- **We would like to minimize the normalized turnaround time for each process, and the averaged value over all processes**
- **When the current process completes or is blocked, choose next process with the greatest value of R**

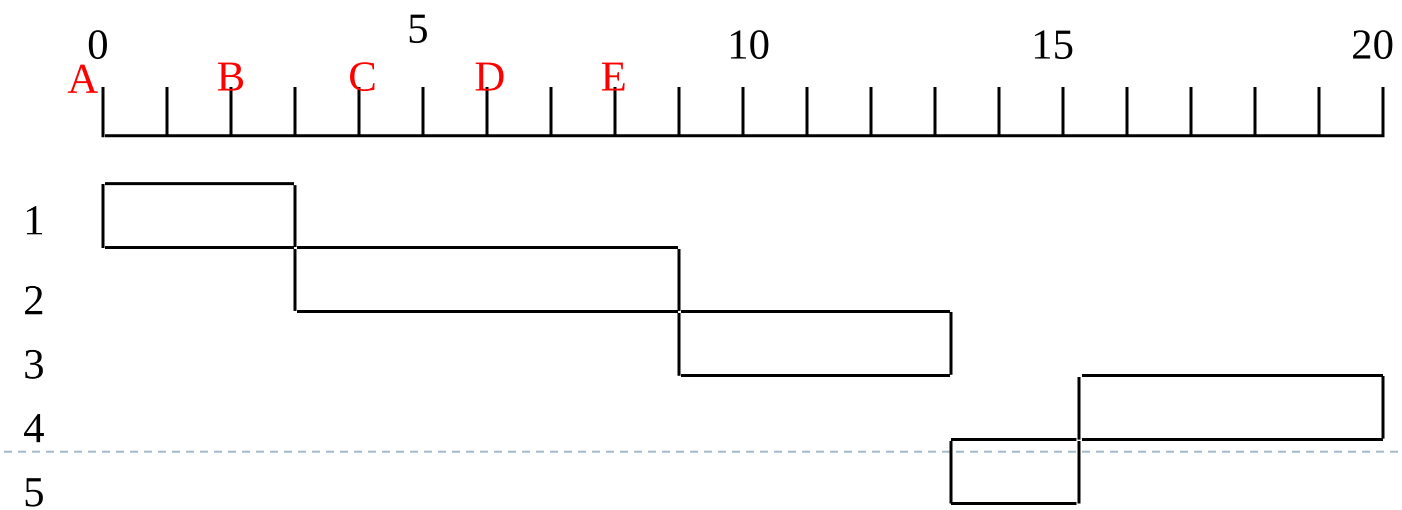
nonpreemptive

$$R = \frac{w+s}{s}, \; where \begin{cases} R = \text{response ratio} \\ w = \text{time spent waiting for the processor} \\ s = \text{expected service time} \end{cases}$$

If a process with this value is dispatched immediately, R is equal to the normalized turnaround time

Highest Response Ratio Next (2) The Example

	Process	A	B	C	D	E	Mean
	Arrival Time	0	2	4	6	8	
	Service Time (T_s)	3	6	4	5	2	
HRRN	Finish Time	3	9	13	20	15	
	Turnaround Time (T_r)	3	7	9	14	7	8.00
	T_r/T_s	1.00	1.17	2.25	2.80	3.5	2.14



Highest Response Ratio Next (3) Analysis

- **This approach is attractive because it accounts for the age of the process**
  - While shorter jobs are favored (a smaller denominator yields a larger ratio), aging without service increases the ratio so that a longer process will eventually get past competing shorter jobs
  - No starvation!!! (compared with SPN and SRT)
- **Disadvantage: the expected service time must be estimated to use this strategy (as with SRT and SPN)**

Multilevel Queue Scheduling
多层队列调度

- **Motivation: Processes are easily classified into different groups**
  - For example: foreground (interactive) processes vs. background (batch) processes
    - **They have different response-time requirements and may have different scheduling needs.**
    - **Foreground processes may have priority over background processes**

Multilevel Queue Scheduling
多层队列调度

- **Multilevel queue scheduling:**
  - The ready queue is partitioned into several separate queues
  - The processes are permanently assigned to one queue
  - Each queue has its own scheduling algorithm (for example:foreground queue with RR; background queue with FCFS)
  - There must be scheduling among the queues (commonly fixed-priority preemptive scheduling)

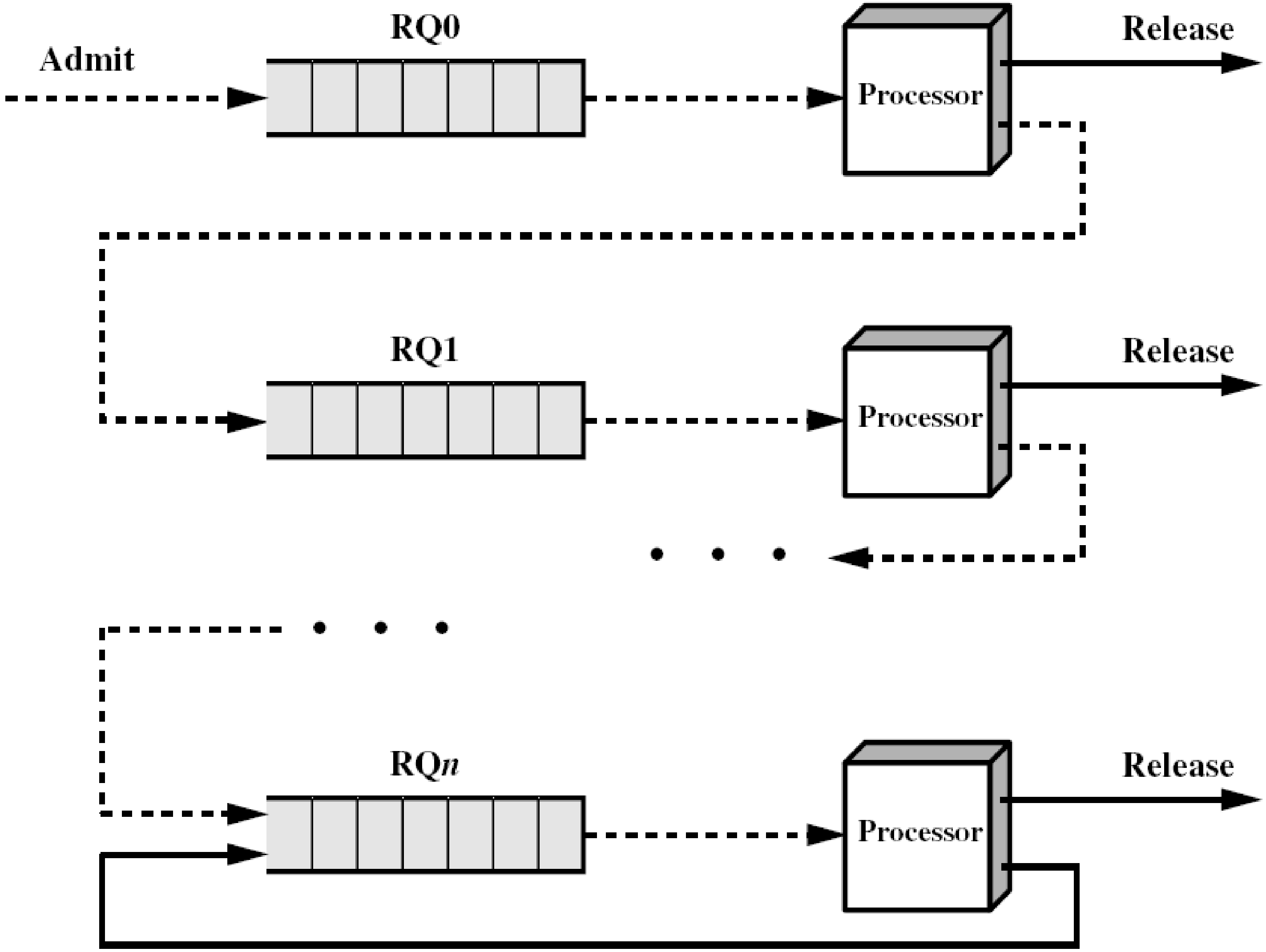
Feedback 反馈 (1)
The Underlying Idea

- **What if there is no indication of the relative length of various processes?**
  - None of SPN, SRT, and HRRN can be used!
- **Another way of establishing a preference for shorter job: to penalize jobs that have been running longer**
  - We focus on the time spent in execution so far, instead of the time remaining to execute

Feedback (2)
How to do?

- **Scheduling is done on a preemptive (at time quantum) basis, and a dynamic priority mechanism is used**
  - A process is placed in RQ0 when first enters system
  - After its first execution, when it returns to the Ready state, it is placed in RQ1
  - Each subsequent time that it is preempted, it is demoted to the next lower-priority queue
- **Within each queue (except the lowest-priority queue), FCFS is used**
  - The lowest-priority queue is treated in round-robin fashion

Figure. Feedback Scheduling



*This approach is known as **multilevel feedback***

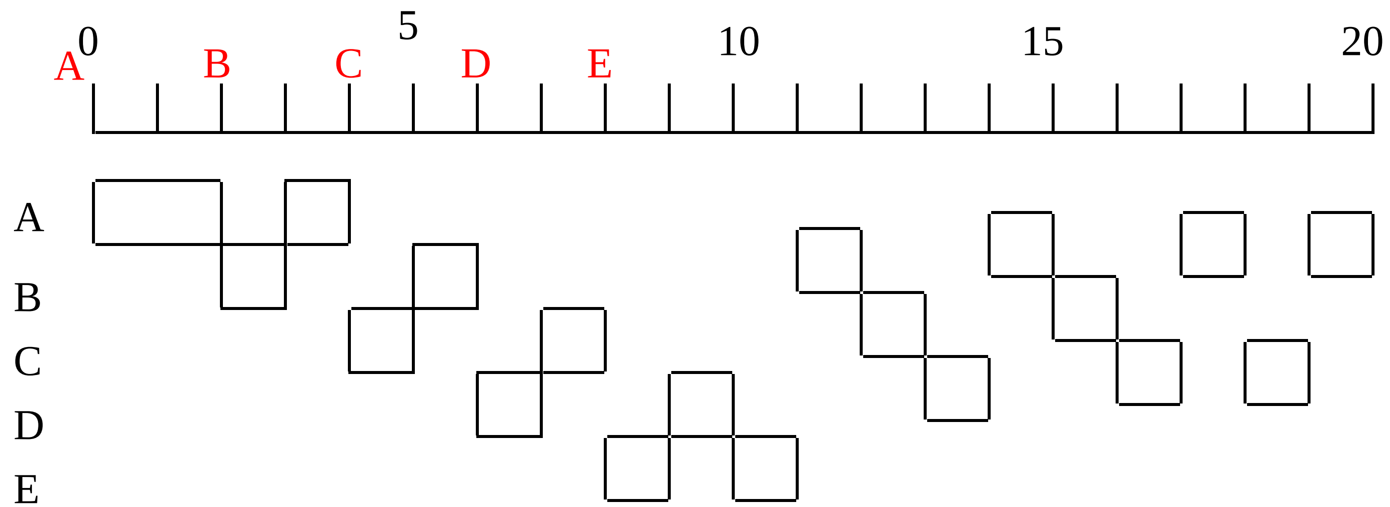
Feedback (3)
What are the effects?

- **A shorter process will complete quickly, without migrating very far down the hierarchy of ready queues**
- **A longer process will gradually drift downward.**
- **Thus, *Newer, shorter processes are favored over older, longer processes***

Feedback (4)
Variations

- **A simple version: perform preemption at periodic intervals (same as in round robin)**
  - Problem: the turnaround time of longer processes can stretch out alarmingly.
- **One variation: vary the preemption times according to the queue**
  - A process from RQi is to execute for 2^i time units and then is preempted
- **In both cases, longer process may suffer starvation**
  - A possible remedy: move a process to a higher-priority queue after it waits a long time for service in the current queue

Feedback (5)
The Example



Problem: When there is only one process in the system, will the process be demoted to lower priority queue?

Feedback (6)
General Framework

- **In general, a multilevel feedback-queue scheduler is defined by**
  - The number of queues
  - The scheduling algorithm for each queue
  - The method used to determine when to upgrade a process to a higher-priority queue
  - The method used to determine when to demote a process to a lower-priority queue
  - The method used to determine which queue a process will enter when that process need service

Table 9.3 Characteristics of Various Scheduling Policies

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

w = time spent in system so far, waiting and executing
e = time spent in execution so far
s = total service time required by the process, including e

Fair-Share Scheduling (1) 公平共享调度-Motivation

- In a *multiuser* system, if individual user applications or jobs may be organized as ***multiple processes (or threads)***
  - User is concerned about the performance of his/her set of processes
    - instead of a particular process
- This can be extended to ***groups of users***, even if each user is represented by a single process.
  - Scheduler could attempt to given each group similar service

- **A better name should be “公平份额调度”！！**

Fair-Share Scheduling (2)
One Scheme

- **This scheme is referred to as the fair-share scheduler (FSS)**
- **The system divides the user community into a set of fair-share groups**
  - Each group is allocated a fraction of the processor resource
- **Scheduling is done on the basis of priority**
  - Priority takes into account the ***underlying priority of the process*** (进程的基础优先级), ***its recent processor usage***, and ***the recent processor usage of the group***

Fair-Share Scheduling (3)
Priority

- **The higher the numerical value of the priority, the lower the priority**

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$GCPU_k(i) = \frac{GCPU_k(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4 \times W_k}$$

where

$CPU_j(i) =$ Measure of processor utilization by process j through interval i

$GCPU_k(i) =$ Measure of processor utilization by group k through interval i

$P_j(i) =$ Priority of process j at beginning of interval i

$Base_j =$ Base priority of process j

$W_k =$ Weighting assigned to group k, with the constraints: $0 < W_k \leq 1$ *and* $\sum_k W_k = 1$

Fair-Share Scheduling (4)
Processor Utilization Measuring

- **Processor utilization is measured as follows:**
  - The processor is interrupted 60 times per second
  - During each interrupt, the processor usage field of the currently running process is incremented, as is the corresponding group processor field
  - Once per second, priorities are recalculated for all processes.

Time	Process A Priority	Process A Process	Process A Group	Process B Priority	Process B Process	Process B Group	Process C Priority	Process C Process	Process C Group
0	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0	60	0	0
1	90	30	30	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0 1 2 • • 60
2	74	15 16 17 • • 75	15 16 17 • • 75	90	30	30	75	0	30
3	96	37	37	74	15	15 16 17 • • 75	67	0 1 2 • • 60	15 16 17 • • 75
4	78	18 19 20 • • 78	18 19 20 • • 78	81	7	37	93	30	37
5	98	39	39	70	3	18	76	15	18

Group 1 (Process A) **Group 2** (Process B, Process C)

Shaded rectangle represents executing process

Three processes
A: in the first group
B, C: in the 2nd group

Each group has a weighting of 0.5

Assume all processes are processor bound

All processes have a base priority of 60

Processor is interrupted 60 times per second

Priorities are recalculated once per second