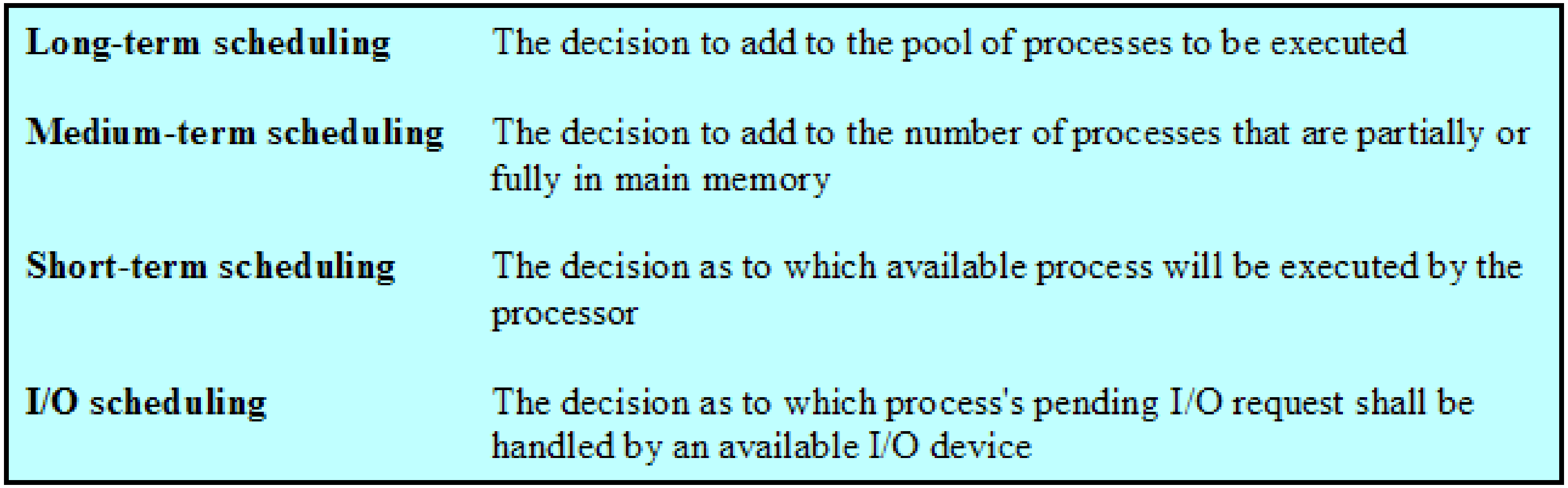
Uniprocessor Scheduling

Chapter 9

Aim of Scheduling

- Assign processes to be executed by the processor(s)
- Response time
- Throughput
- Processor efficiency

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

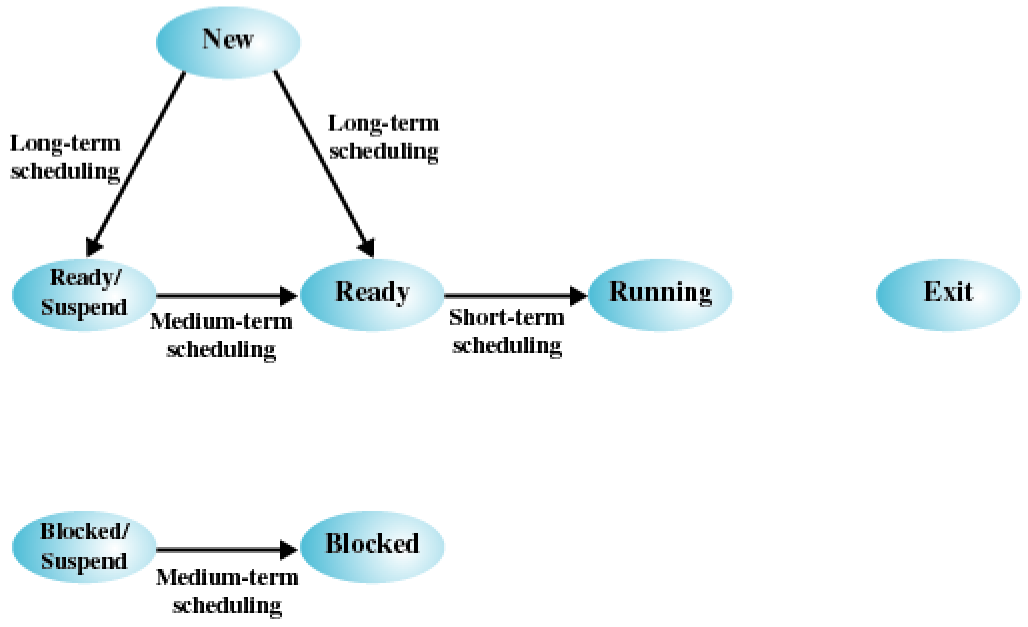


Figure 9.1 Scheduling and Process State Transitions

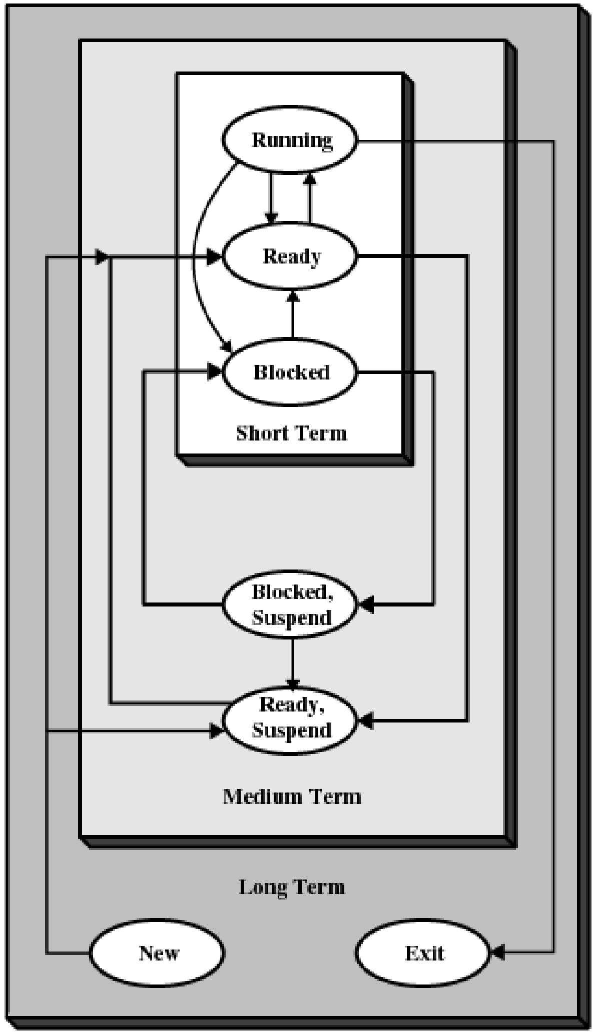


Figure 9.2 Levels of Scheduling

Long-Term Scheduling

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

Medium-Term Scheduling

- Part of the swapping function
- 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

Short-Tem Scheduling 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-related
  - Quantitative
  - Measurable such as response time and throughput

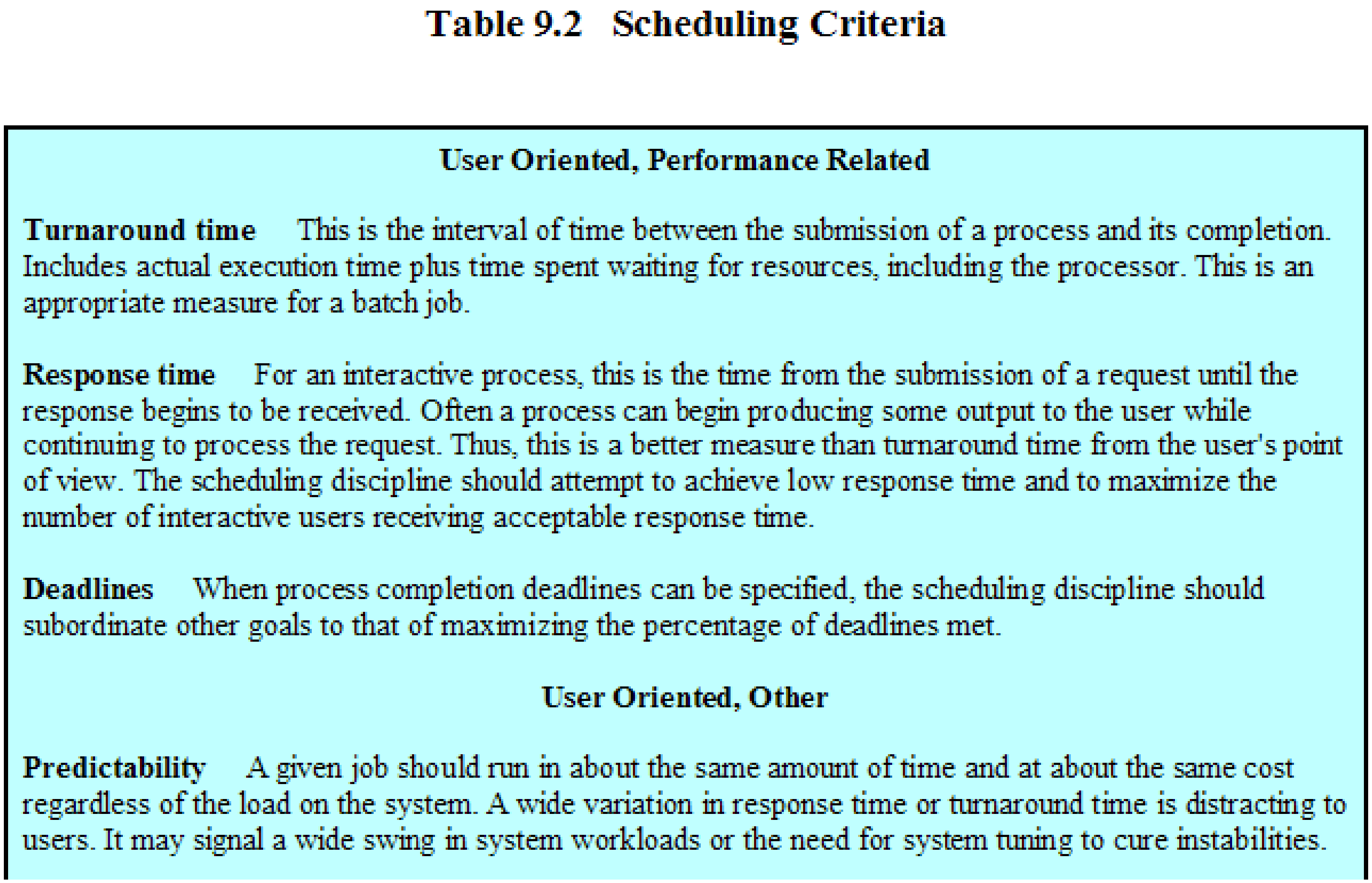


Table 9.2 Scheduling Criteria

User Oriented, Performance Related

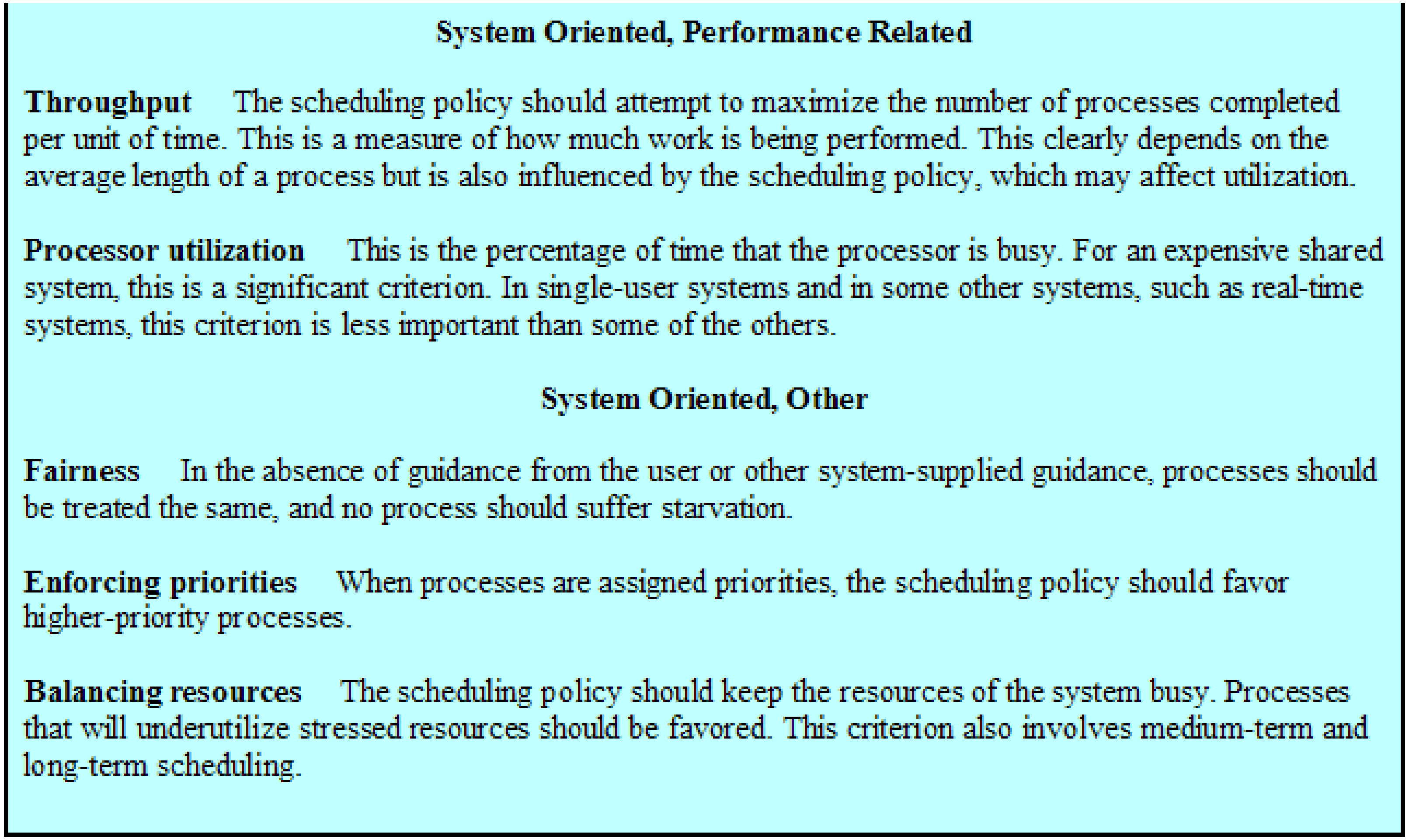
Turnaround time This is the interval of time between the submission of a process and its completion. Includes actual execution time plus time spent waiting for resources, including the processor. This is an appropriate measure for a batch job.

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.



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.

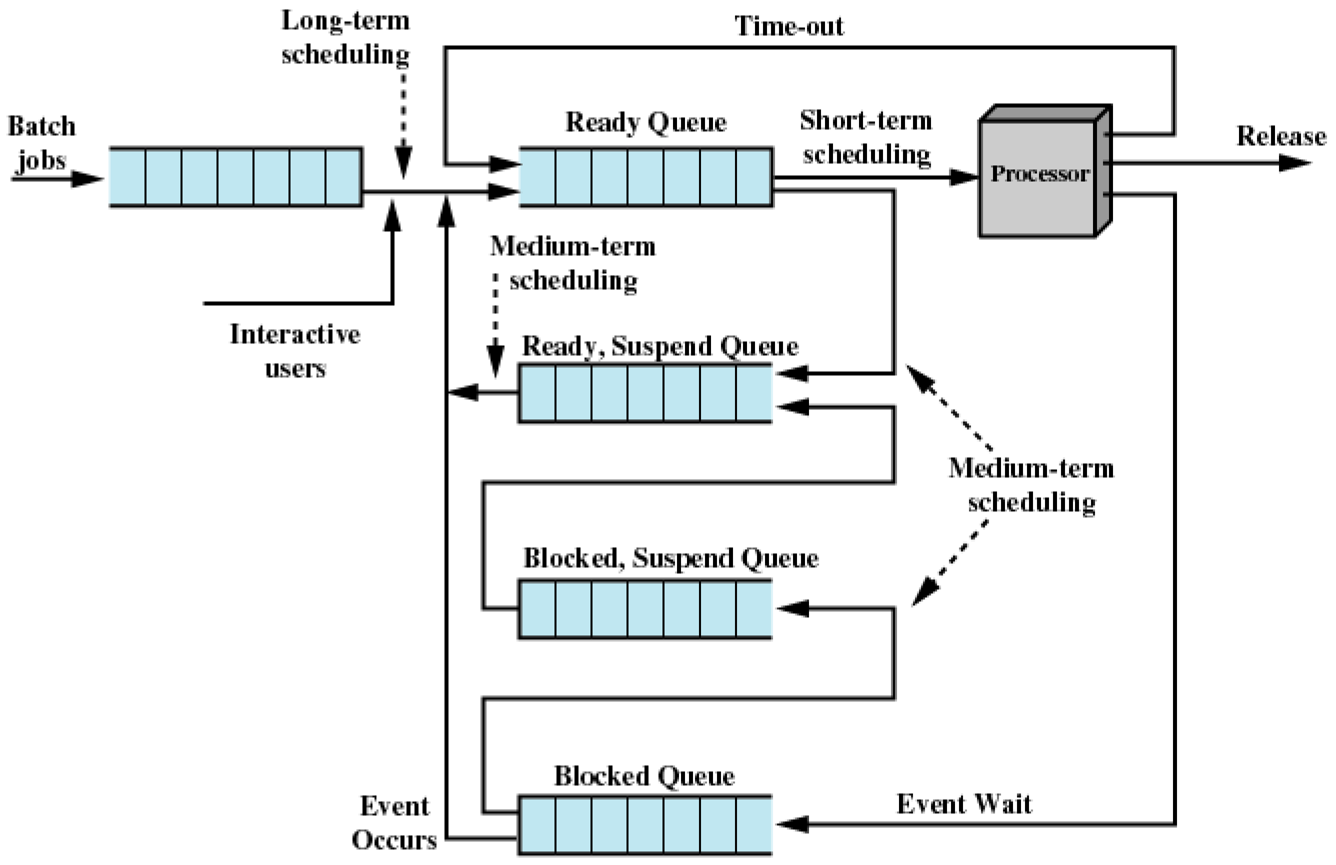


Figure 9.3 Queuing Diagram for Scheduling

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
- Lower-priority may suffer starvation
  - Allow a process to change its priority based on its age or execution history

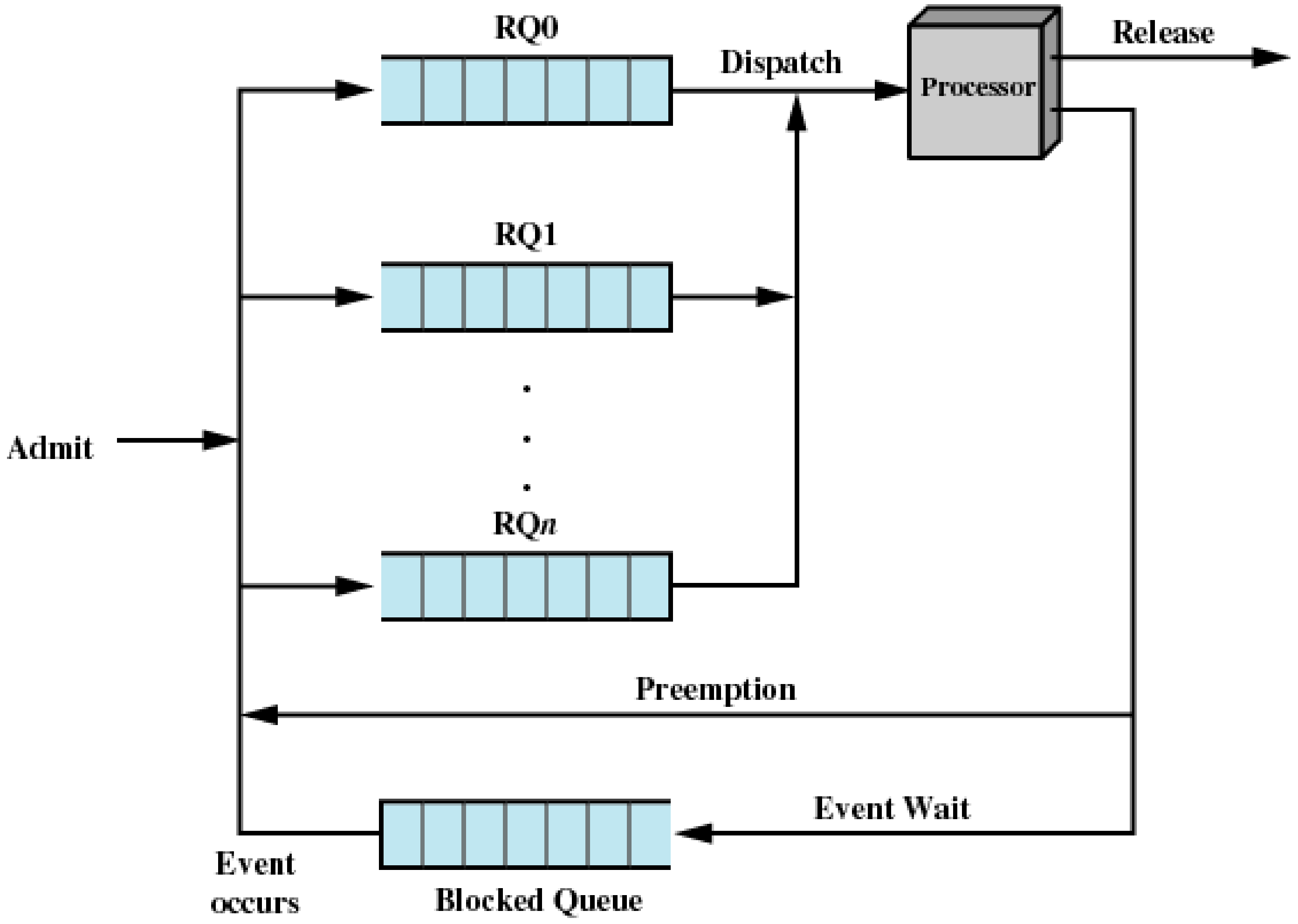


Figure 9.4 Priority Queuing

Alternate Scheduling Policies

- Selection Function determines which process among ready processes is selected next for execution.
- It can be based on priority, resource requirements or the execution characteristics of a process
- w=time spent in system so far, waiting and execution
- e = time spent in execution so far
- s = total service time required by the process including e. It should be supplied by or estimated by the user.

Decision Mode

- Nonpreemptive
  - 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 the Ready state by the operating system
  - Allows for better service since any one process cannot monopolize the processor for very long

Process Scheduling Example

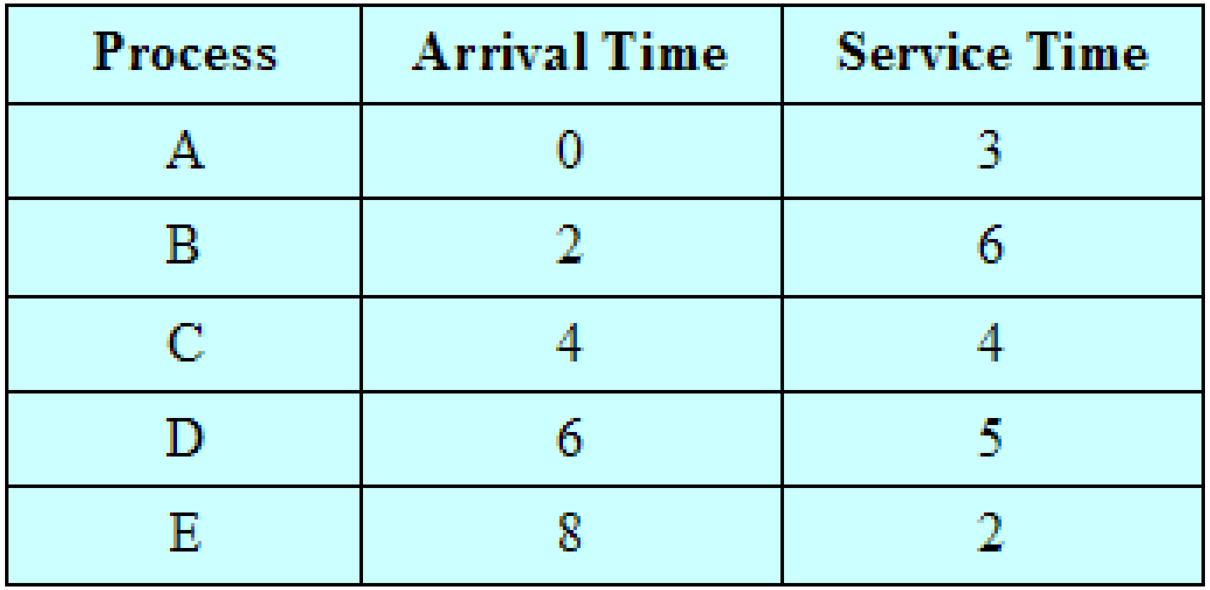
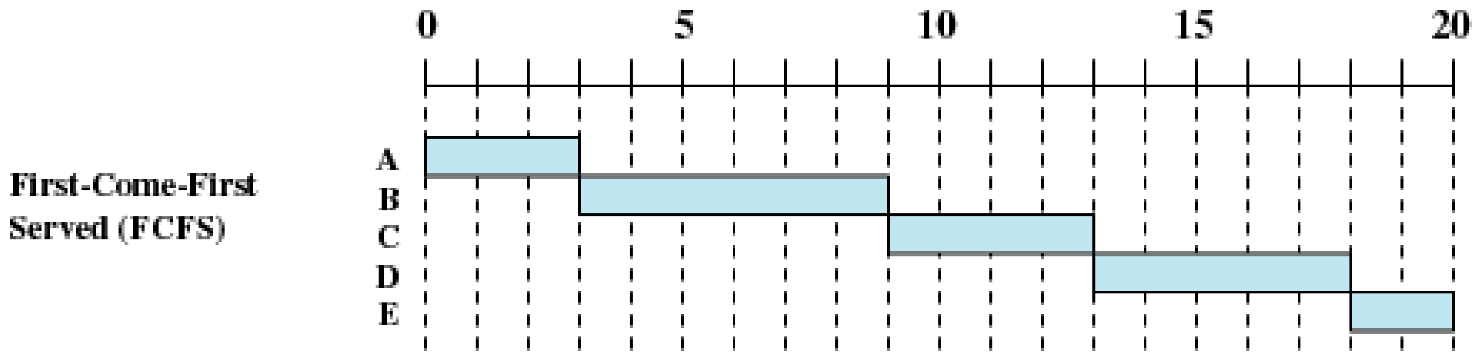


Table 9.4 Process Scheduling Example

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

First-Come-First-Served (FCFS)



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

First-Come-First-Served (FCFS)

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

First-Come-First-Served (FCFS)

- Favours processor-bound processes over I/O-bound processes.
  - When a processor-bound process is running, all of the I/O bound processes must wait, even if state is changed after some time
  - If the processor-bound process is also blocked, the processor becomes idle. Thus, FCFS may result in inefficient use of both the processor and the I/O devices.
  - often combined with a priority scheme to provide an effective scheduler. The scheduler may maintain a number of queues

Process	Arrival Time	Service Time (T_s)	Start Time	Finish Time	Turnaround Time (T_r)	T_r/T_s
W	0	1	0	1	1	1
X	1	100	1	101	100	1
Y	2	1	101	102	100	100
Z	3	100	102	202	199	1.99
Mean					100	26

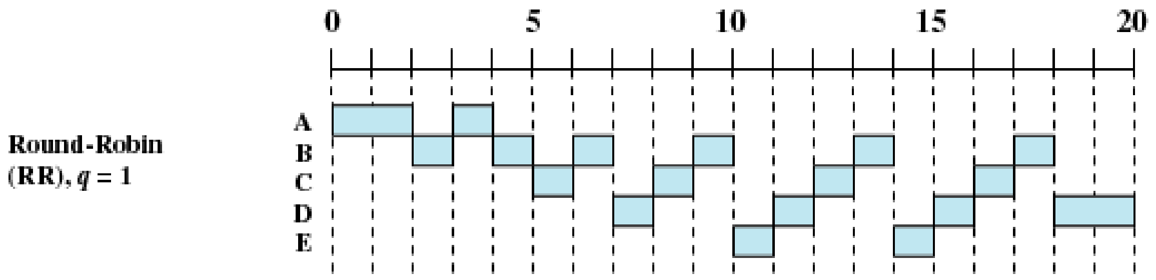
Table 9.5 A Comparison of Scheduling Policies

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

Table 9.5 A Comparison of Scheduling Policies

Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
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
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
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
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
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
FB q = 1						
Finish Time	4	20	16	19	11	
Turnaround Time (T_r)	4	18	12	13	3	10.00
T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB q = 2^i						
Finish Time	4	17	18	20	14	
Turnaround Time (T_r)	4	15	14	14	6	10.60
T_r/T_s	1.33	2.50	3.50	2.80	3.00	2.63

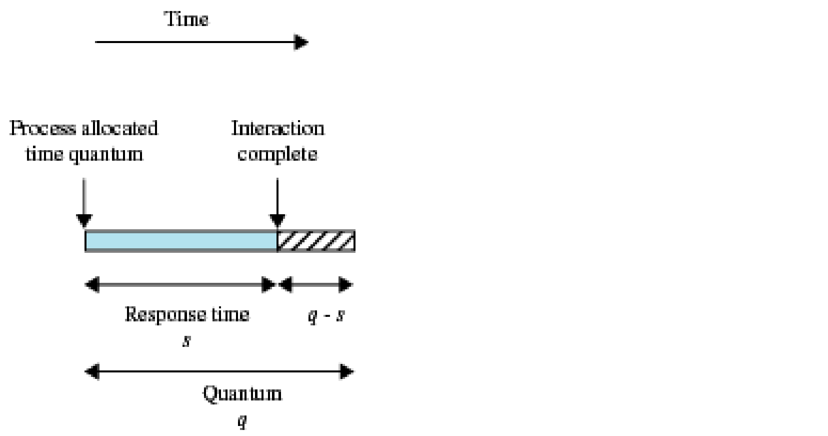
Round-Robin



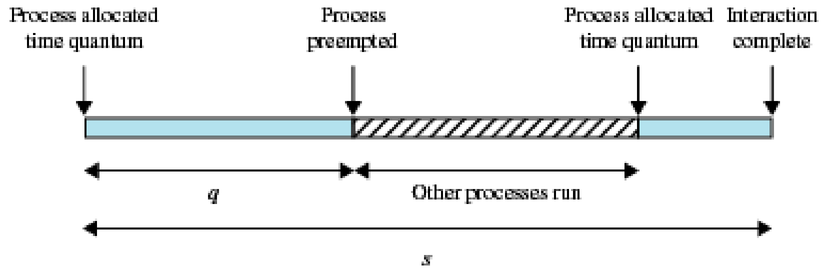
- A clock interrupt is generated at periodic intervals, i.e. time slicing
- Uses preemption based on a clock
- An amount of time is determined that allows each process to use the processor for that length of time

Round-Robin

- When an interrupt occurs, the currently running process is placed in the read queue
  - Next ready job is selected
- The principal design issue is the length of the time quantum
  - should be slightly greater than the time required for a typical interaction or process function
- One drawback to round robin is its relative treatment of processor-bound and I/O-bound processes.



(a) Time quantum greater than typical interaction


Process allocated time quantum
Process preempted
Process allocated time quantum
Interaction complete
q
Other processes run
s

(b) Time quantum less than typical interaction

Figure 9.6 Effect of Size of Preemption Time Quantum

Round-Robin

Auxiliary queue

- The new feature is an FCFS auxiliary queue to which processes are moved after being released from an I/O block.
- When a dispatching decision is to be made, processes in the auxiliary queue get preference over those in the main ready queue.
- It runs no longer than a time equal to the basic time quantum minus the total time spent running since it was last selected from the main ready queue..

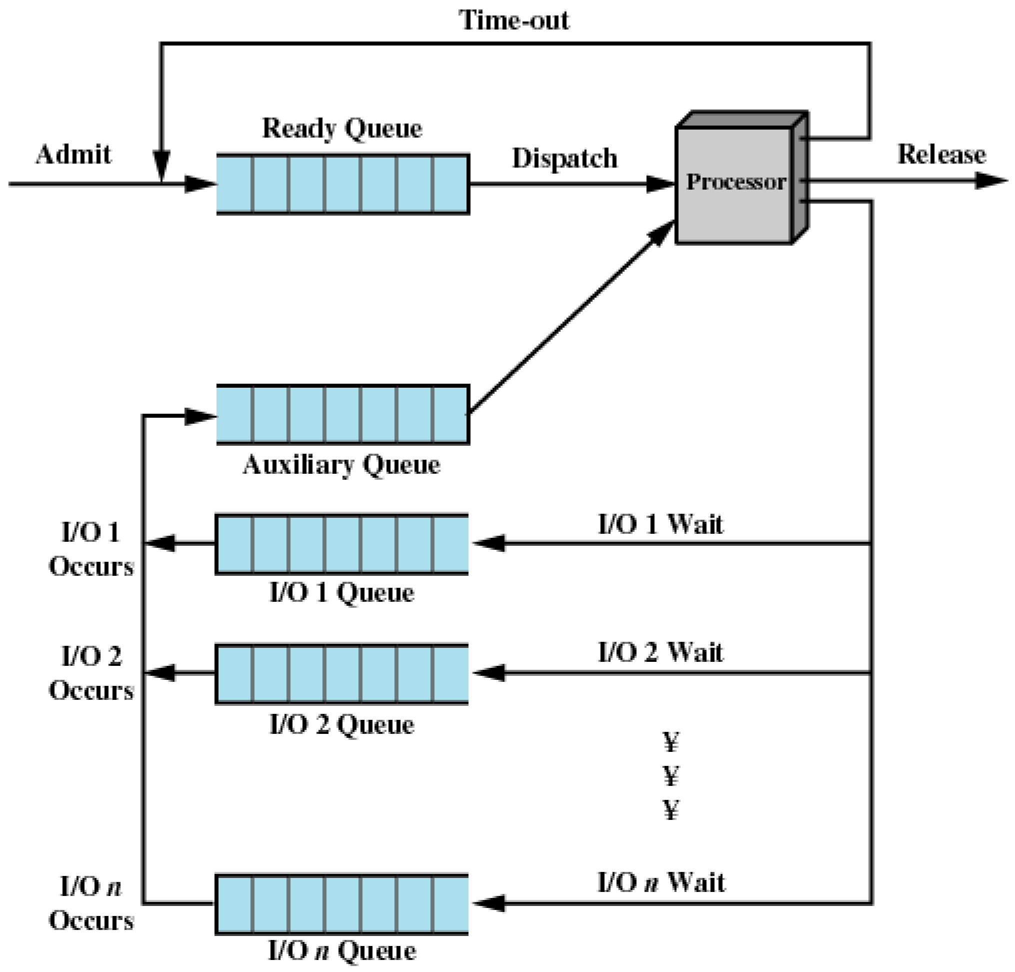
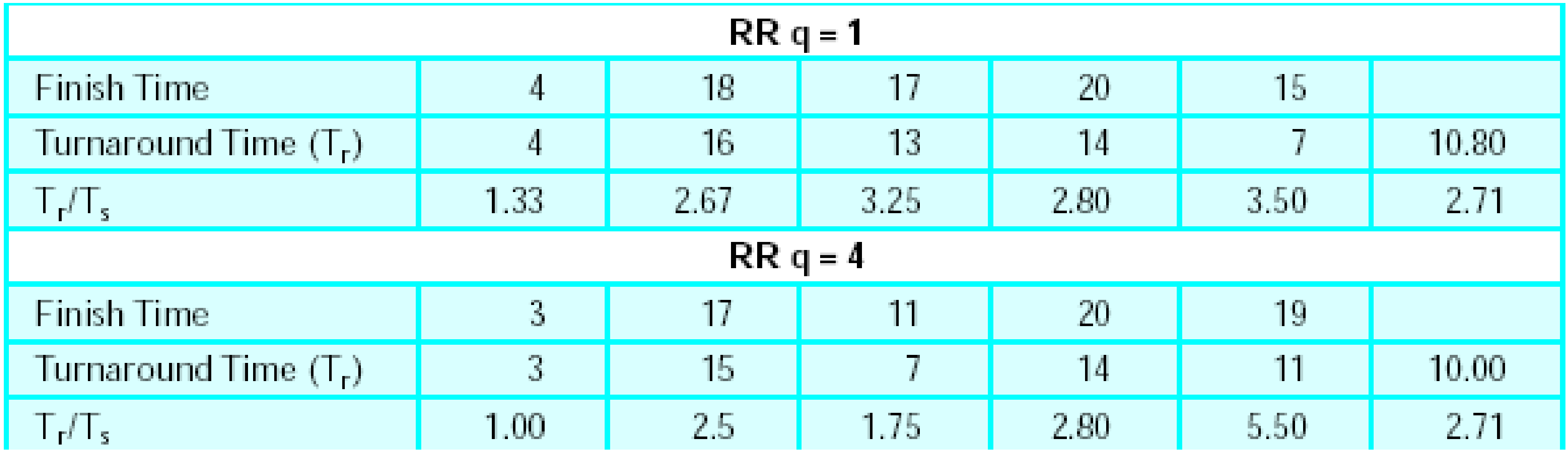
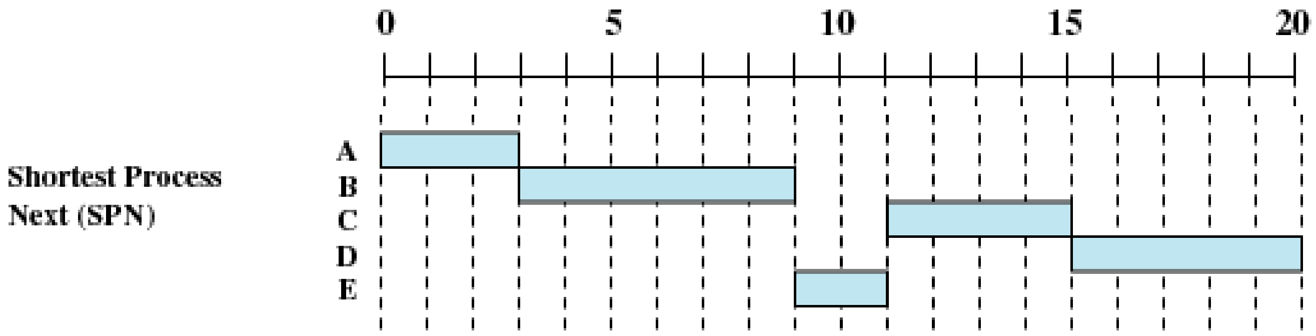


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

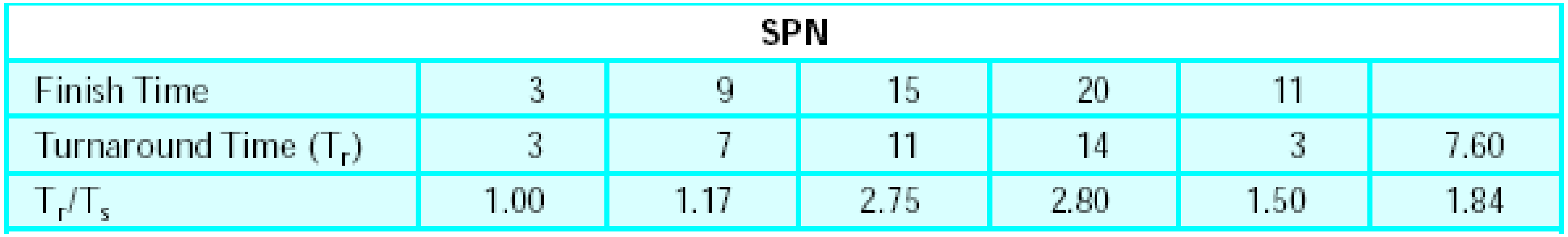


RR q = 1						
Finish Time	4	18	17	20	15	
Turnaround Time (T_r)	4	16	13	14	7	10.80
T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR q = 4						
Finish Time	3	17	11	20	19	
Turnaround Time (T_r)	3	15	7	14	11	10.00
T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71

Shortest Process Next



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



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

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, the need to know or at least estimate the required
- How to know processing time?
  - For batch jobs: the programmer to estimate the value and supply it
  - In a production environment: the same jobs run frequently, and statistics may be gathered.
  - For interactive processes: the operating system may keep a running average of each “burst” for each process.
- SPN reduces the bias in favor of longer jobs, it still is not desirable for a time-sharing or transaction processing environment because of the lack of preemption.

Shortest Process Next

- Note that this formulation gives equal weight to each instance.
- Typically, we would like to give greater weight to more recent instances, because these are more likely to reflect future behavior.
- A common technique for predicting a future value on the basis of a time series of past values is exponential averaging.

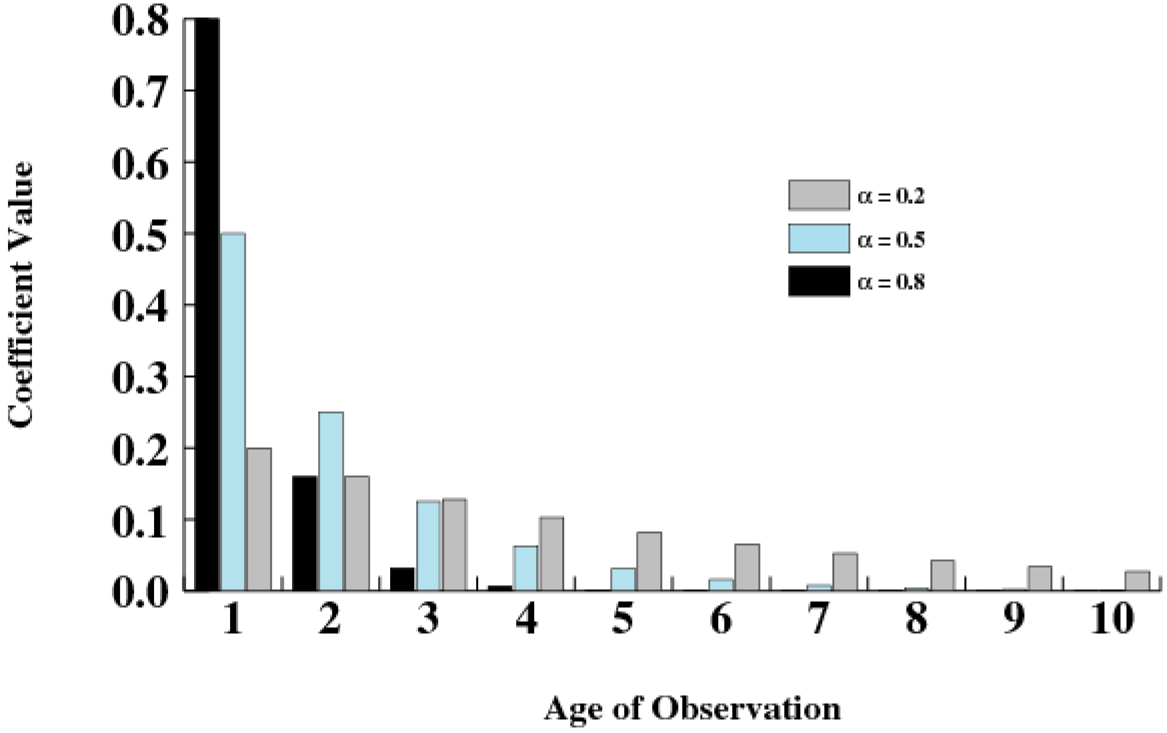
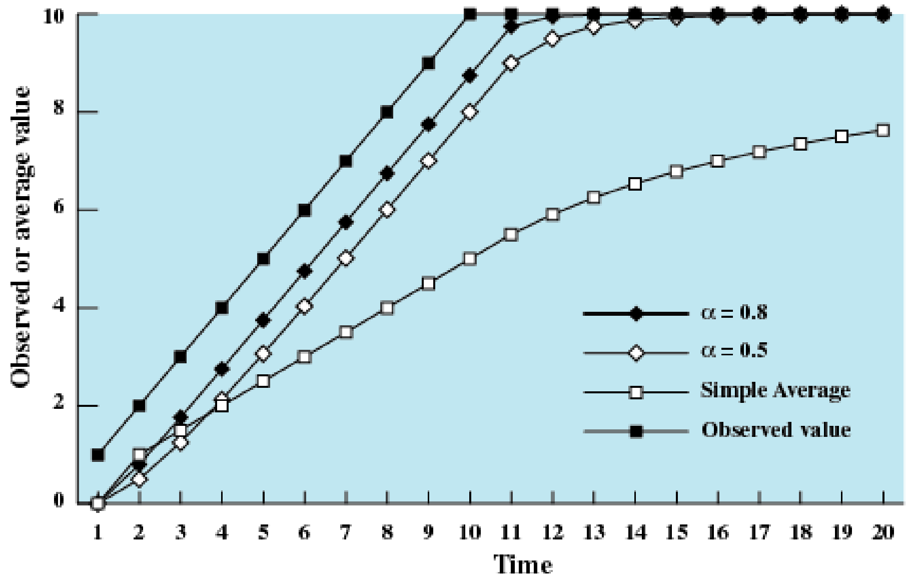
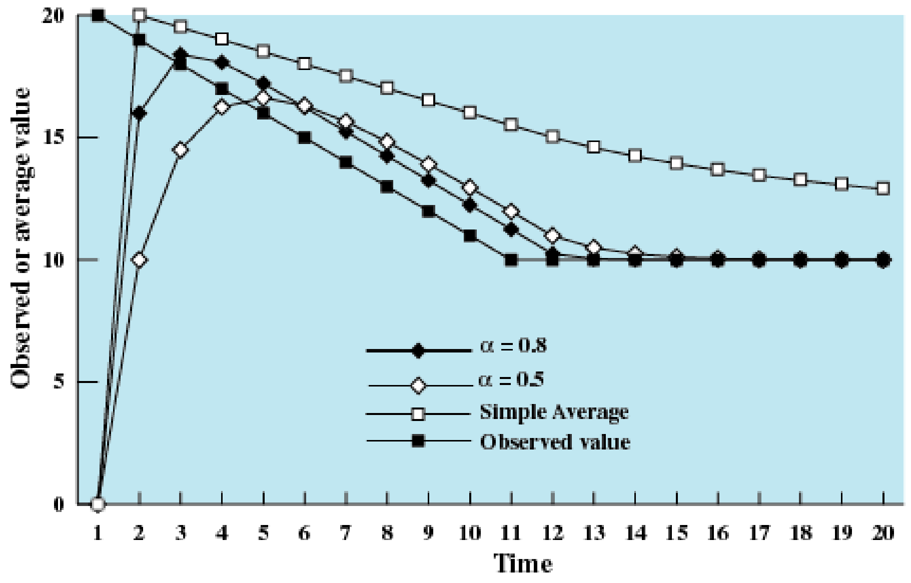


Figure 9.8 Exponential Smoothing Coefficients



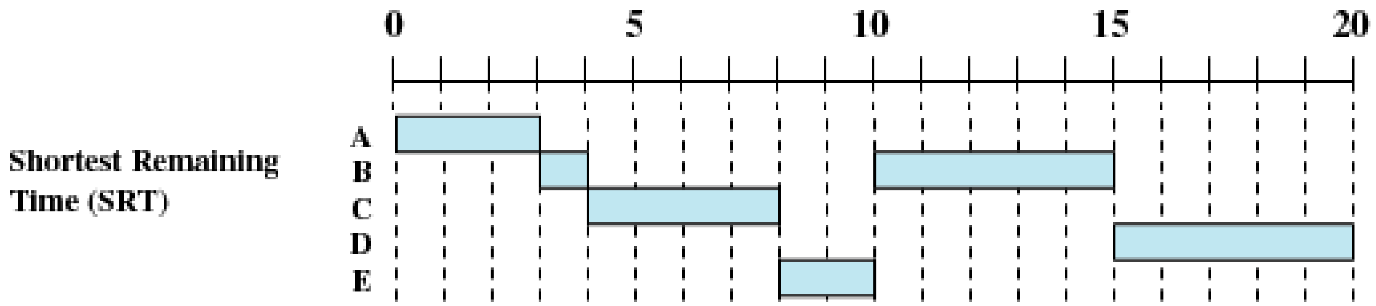
(a) Increasing function



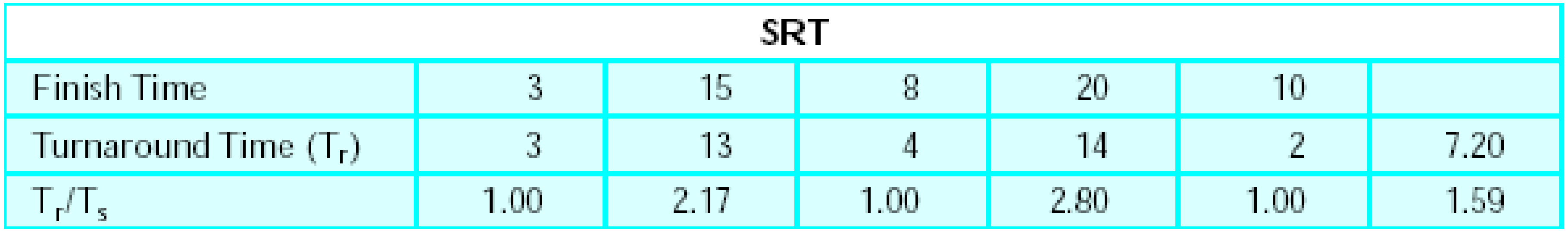
(b) Decreasing function

Figure 9.9 Use of Exponential Averaging

Shortest Remaining Time



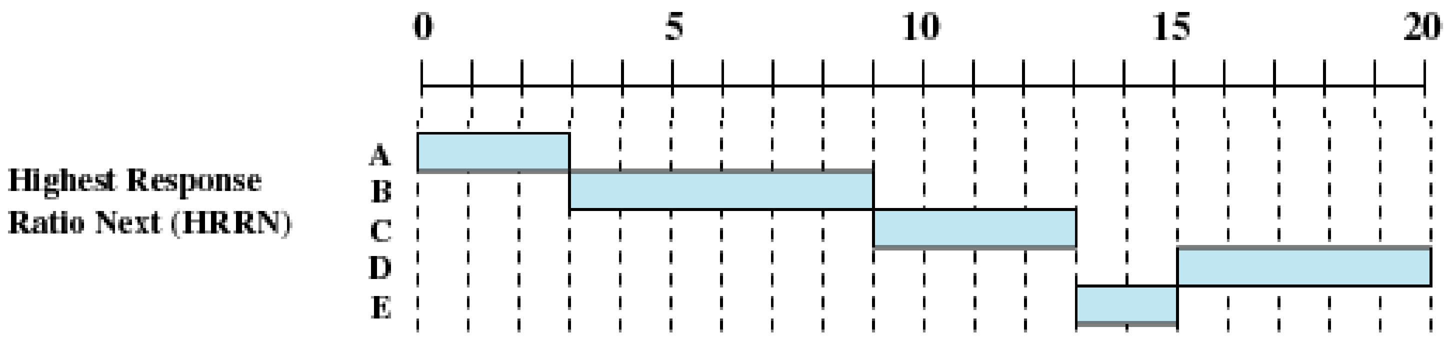
- Preemptive version of shortest process next policy
- When a new process joins the ready queue, it may in fact have a shorter remaining time than the currently running process.
- Accordingly, the scheduler may preempt the current process when a new process becomes ready.
- Must estimate processing time



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

- SRT does not have the bias in favor of long processes found in FCFS. Unlike round robin, no additional interrupts are generated, reducing overhead.
- On the other hand, elapsed service times must be recorded, contributing to overhead.
- SRT should also give superior turnaround time performance to SPN, because a short job is given immediate preference to a running longer job.

Highest Response Ratio Next (HRRN)



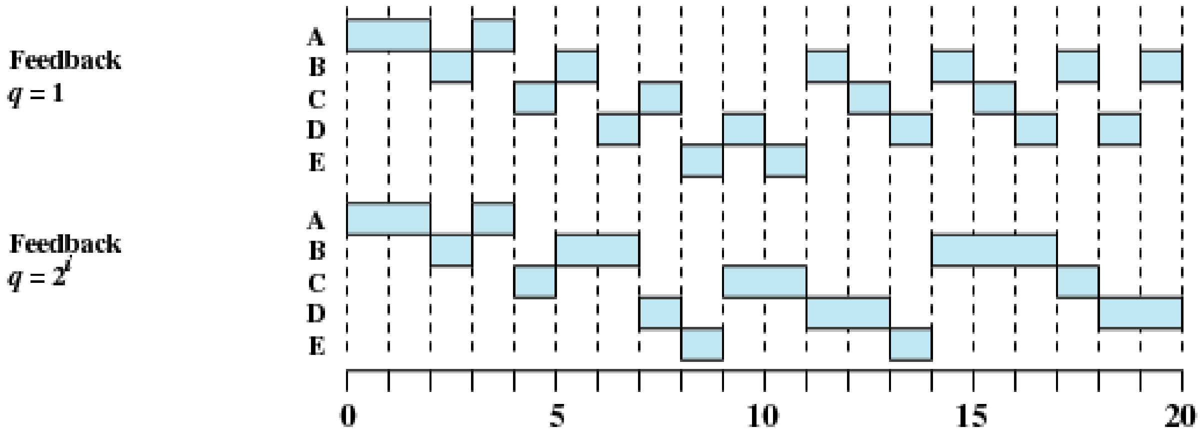
- R = (w + s) / s
- R = response time
- w= time spent waiting for the processor
- s = executed service time.
- For each process, minimize the ratio and the average value over all processes.

Highest Response Ratio Next (HRRN)

- Choose next process with the greatest ratio
- When the current process completes or is blocked, choose the ready process with the greatest value of R.
- The expected service time must be estimated to use highest response ratio next (HRRN).

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

Feedback



- If we have no indication of the relative length of various processes, then none of SPN, SRT, and HRRN can be used.
- Penalize jobs that have been running longer
- Don’t know remaining time process needs to execute

Feedback

- if we cannot focus on the time remaining to execute, let us focus on the time spent in execution so far.
- Scheduling is done on a preemptive (at time quantum) basis, and a dynamic priority mechanism is used.

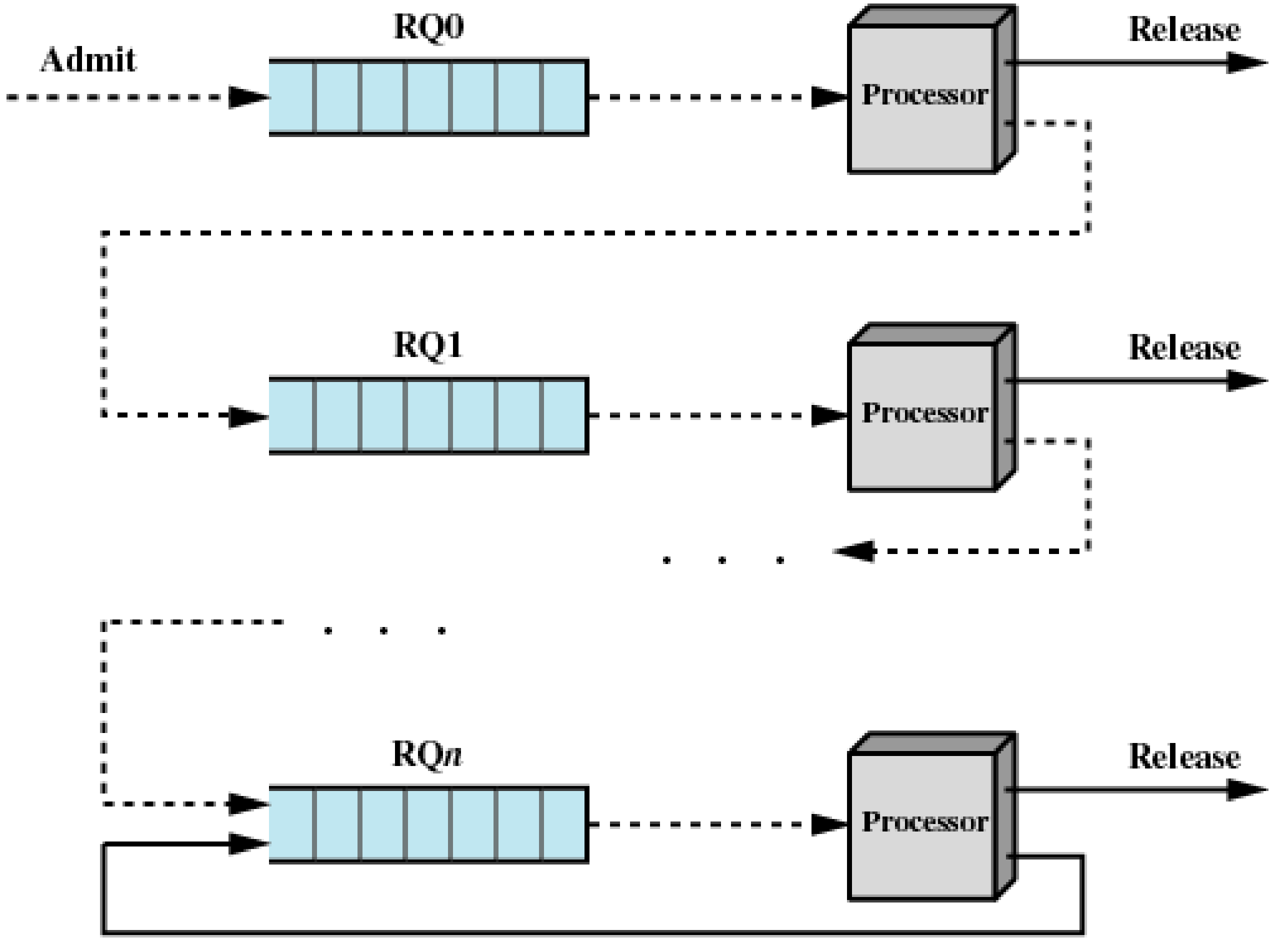
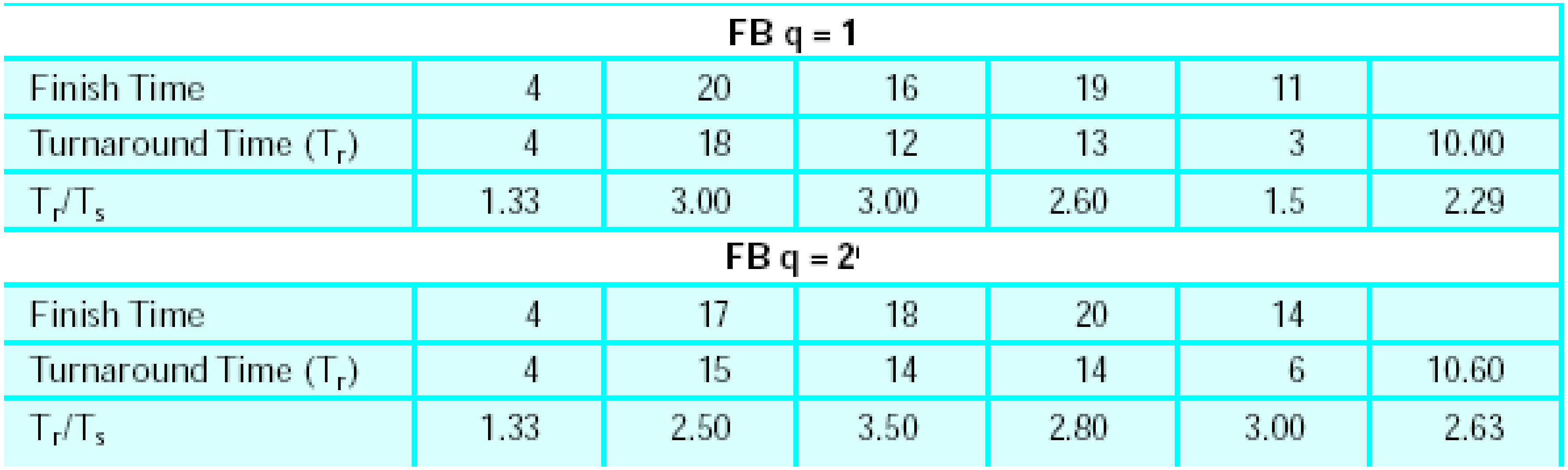


Figure 9.10 Feedback Scheduling

Feedback

- **Multilevel feedback:** operating system allocates the processor to a process and, when the process blocks or is preempted, feeds it back into one of several priority queues.
- Turnaround time of longer processes be alarmingly.
- Possibility of starvation, if new jobs are entering the system frequently.
- To compensate for this, we can vary the preemption times according to the queue, i.e. RQ0 can be given 1 unit of time, RQ2 can be given 2 unit of time and so on, before it is preempted.



FB q = 1						
Finish Time	4	20	16	19	11	
Turnaround Time (T_r)	4	18	12	13	3	10.00
T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2^i$						
Finish Time	4	17	18	20	14	
Turnaround Time (T_r)	4	15	14	14	6	10.60
T_r/T_s	1.33	2.50	3.50	2.80	3.00	2.63

- A longer process may still suffer starvation.
  - A possible remedy is to promote a process to a higher-priority queue after it spends a certain amount of time waiting for service in its current queue.

Table 9.3 Characteristics of Various Scheduling Policies

	Selection Function	Decision Mode	Throughput	Response Time	Overhead	Effect on Processes	Starvation
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 waiting
e	=	time spent in execution so far
s	=	total service time required by the process, including e

Performance Comparison

- Performance of various scheduling policies is a critical factor in the choice of a scheduling policy.
- Relative performance will depend on a variety of factors
  - The probability distribution of service times of the various processes,
  - the efficiency of the scheduling and context switching mechanisms, and
  - the nature of the I/O demand and the performance of the I/O subsystem.

Table 9.5 A Comparison of Scheduling Policies

	Process	A	B	C	D	E	
	Arrival Time	0	2	4	6	8	
	Service Time (T_s)	3	6	4	5	2	Mean
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
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
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
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
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
FB $q = 1$	Finish Time	4	20	16	19	11	
	Turnaround Time (T_r)	4	18	12	13	3	10.00
	T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2^i$	Finish Time	4	17	18	20	14	
	Turnaround Time (T_r)	4	15	14	14	6	10.60
	T_r/T_s	1.33	2.50	3.50	2.80	3.00	2.63

Table 9.6 Formulas for Single-Server Queues with Two Priority Categories

Assumptions:
1. Poisson arrival rate.
2. Priority 1 items are serviced before priority 2 items.
3. First-in-first-out dispatching for items of equal priority.
4. No item is interrupted while being served.
5. No items leave the queue (lost calls delayed).

(a) General Formulas

$$\lambda = \lambda_1 + \lambda_2$$
$$\rho_1 = \lambda_1 T_{s1};\ \ \rho_2 = \lambda_2 T_{s2}$$
$$\rho = \rho_1 + \rho_2$$
$$T_s = \frac{\lambda_1}{\lambda} T_{s1} + \frac{\lambda_2}{\lambda} T_{s2}$$
$$T_r = \frac{\lambda_1}{\lambda} T_{r1} + \frac{\lambda_2}{\lambda} T_{r2}$$

b) No interrupts; exponential service times

$$T_{r1} = T_{s1} + \frac{\rho_1 T_{s1} + \rho_2 T_{s2}}{1 - \rho_1}$$
$$T_{r2} = T_{s2} + \frac{T_{r1} - T_{s1}}{1 - \rho}$$

(c) Preemptive-resume queuing discipline; exponential service times

$$T_{r1} = T_{s1} + \frac{\rho_1 T_{s1}}{1 - \rho_1}$$
$$T_{r2} = T_{s2} + \frac{1}{1 - \rho_1}\left(\rho_1 T_{s2} + \frac{\rho T_s}{1 - \rho}\right)$$

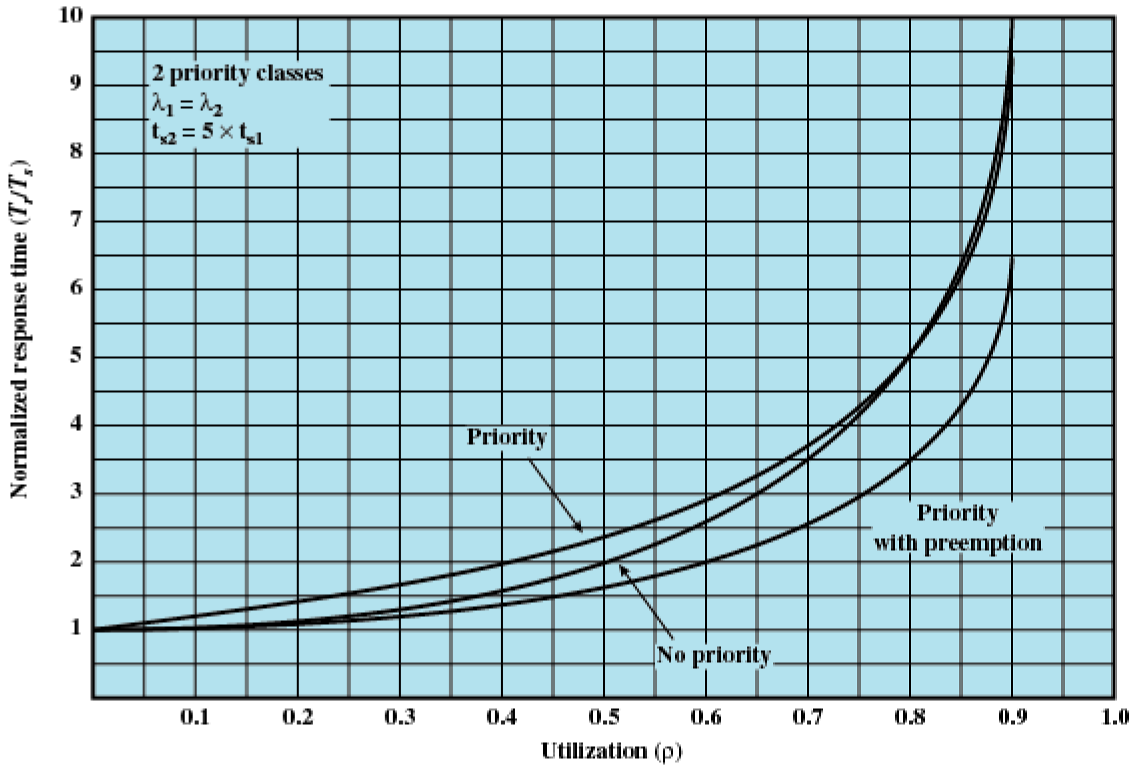


Figure 9.11 Overall Normalized Response Time

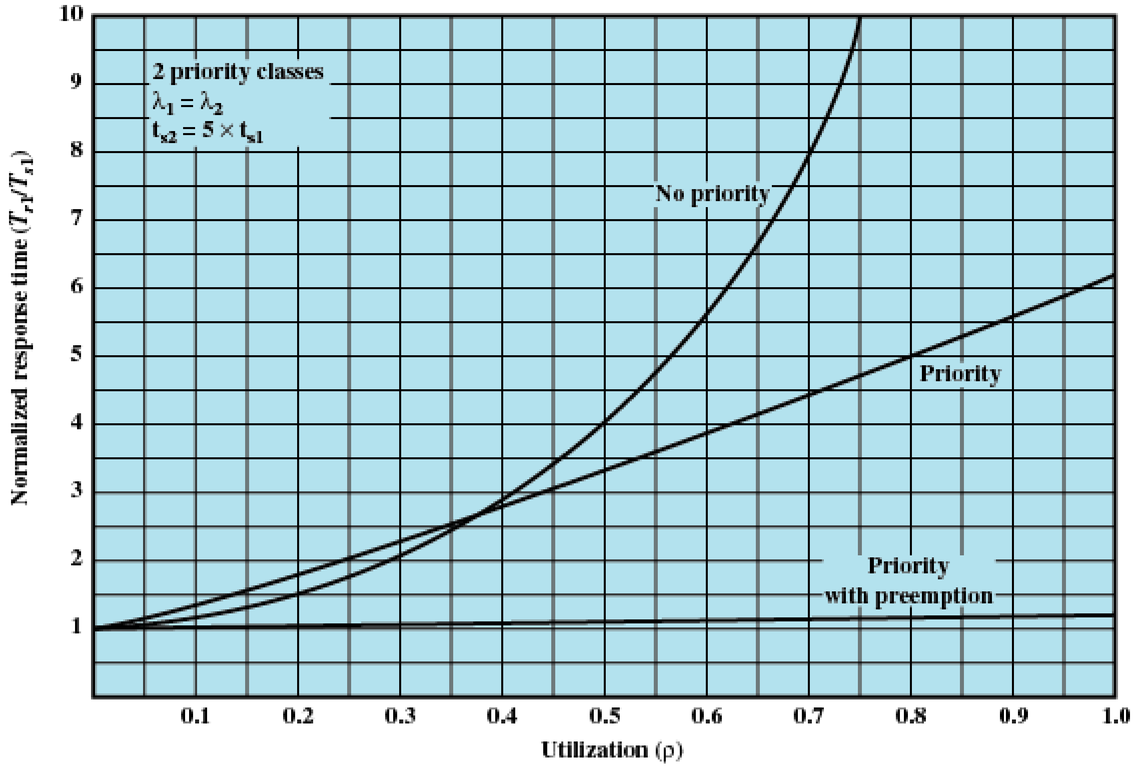


Figure 9.12 Normalized Response Time for Shorter Processes

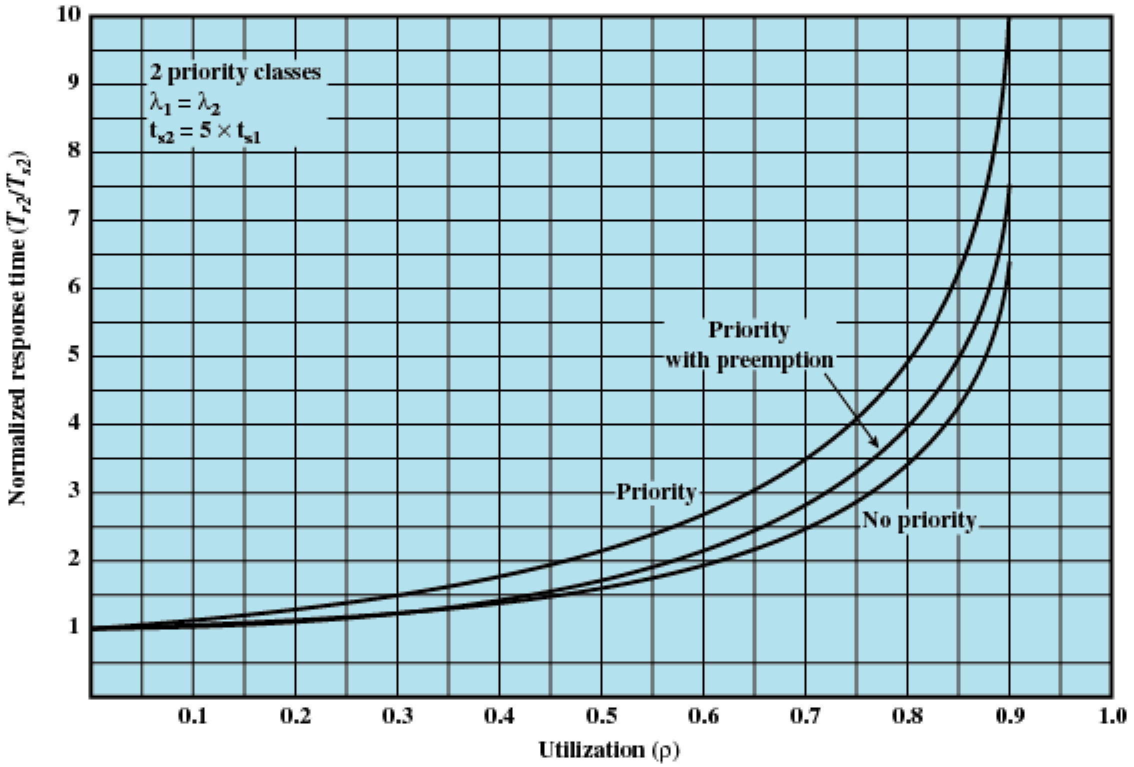


Figure 9.13 Normalized Response Time for Longer Processes

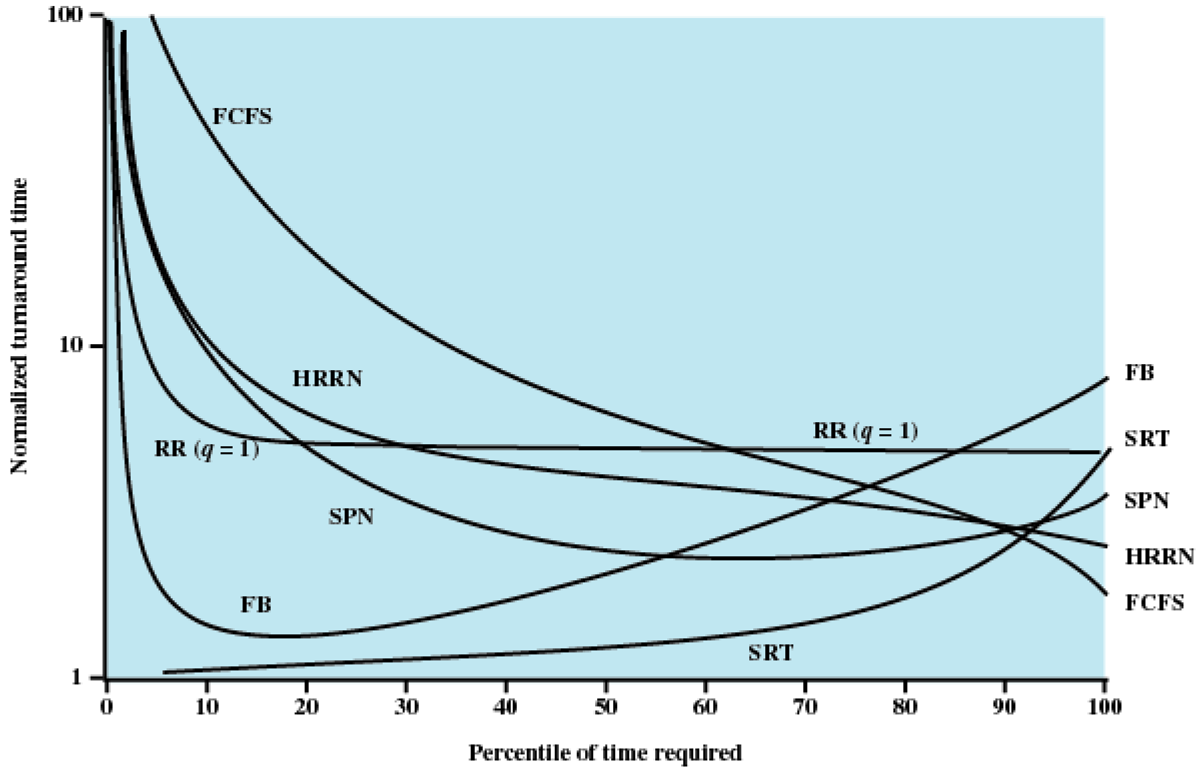


Figure 9.14 Simulation Results for Normalized Turnaround Time

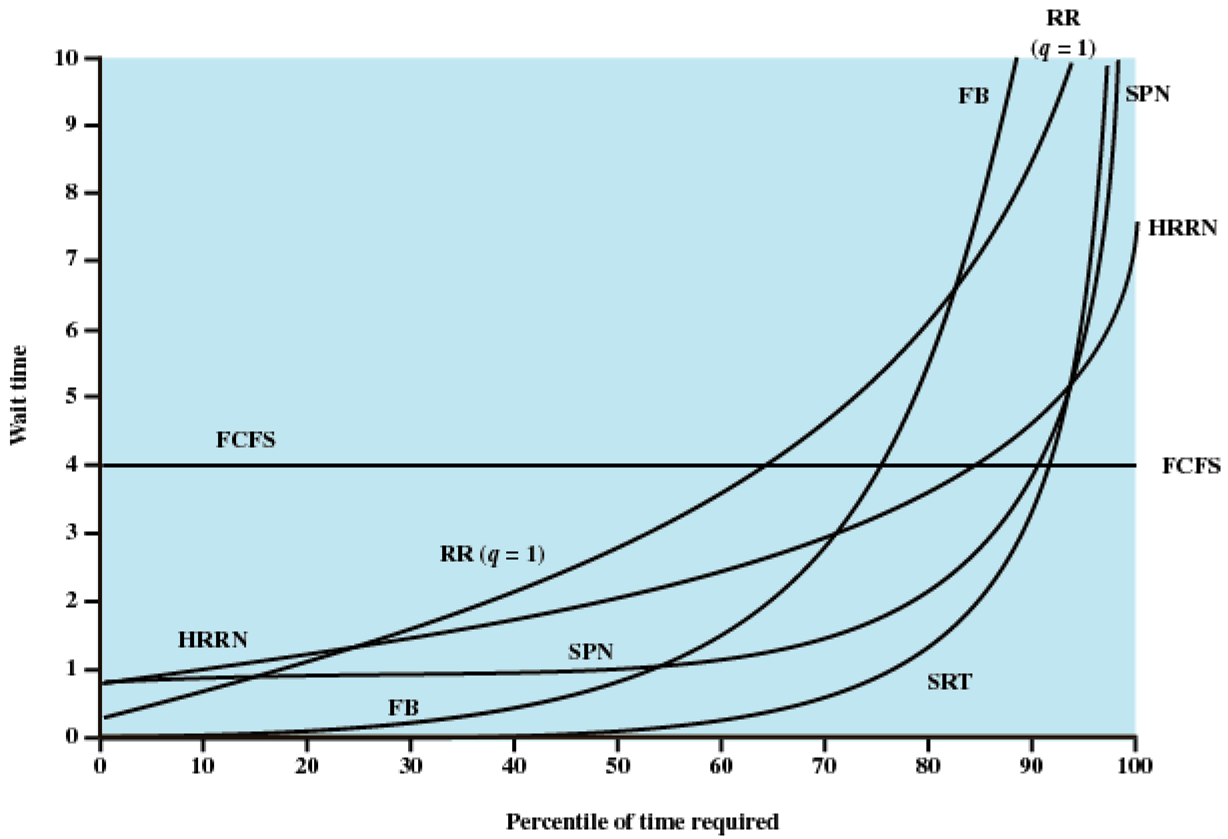


Figure 9.15 Simulation Results for Waiting Time

Fair-Share Scheduling

- User’s application runs as a collection of processes (threads)
- User is concerned about the performance of the application and not with individual processes or threads
- Need to make scheduling decisions based on process sets
- The concept can be extended to groups of users, provide similar services to each group.

Fair-Share Scheduling

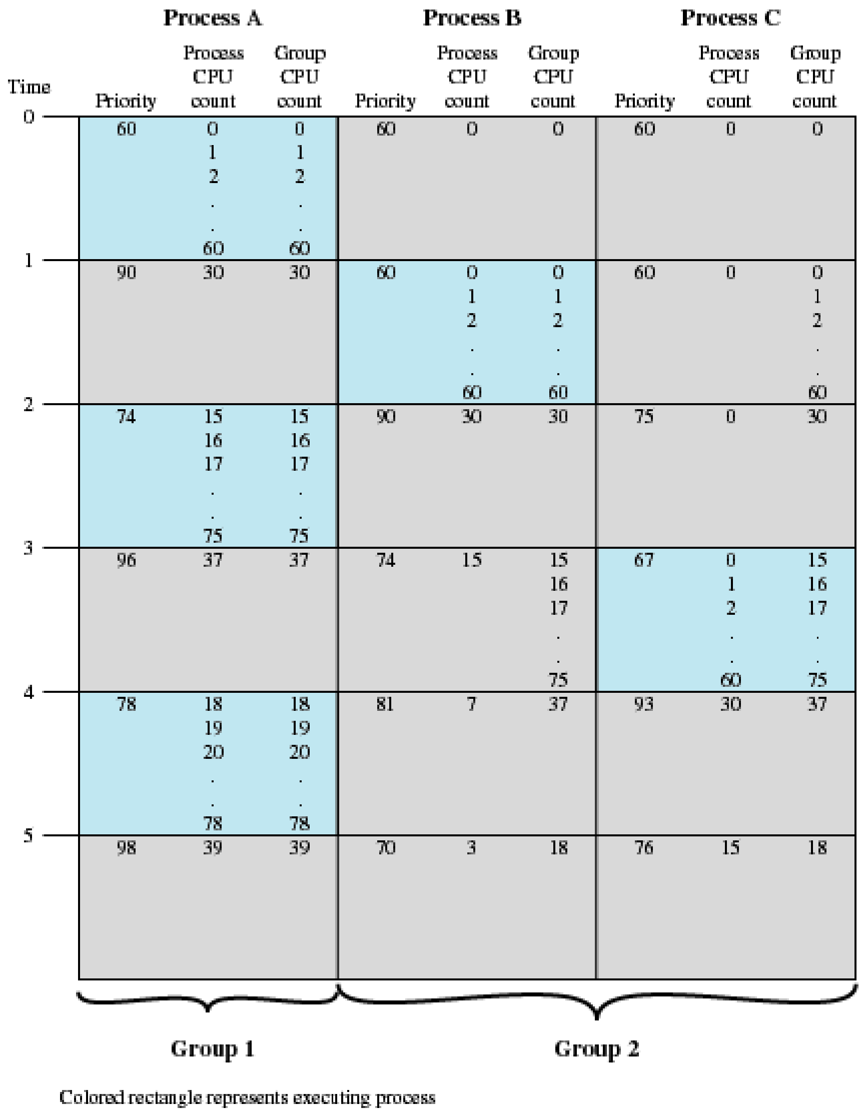
- Each user is assigned a weighting of some sort that defines that user’s share of system resources as a fraction of the total usage of those resources.
- If user A has twice the weighting of user B, then in the long run, user A should be able to do twice as much work as user B.
- The objective of a fair-share scheduler is to monitor usage to give fewer resources to users who have had more than their fair share and more to those who have had less than their fair share.
- Fair-share schedulers (FSS) on Unix

Fair-share schedulers (FSS)

- FSS considers the execution history of a related group of processes, along with the individual execution history of each process in making scheduling decisions.
- The system divides the user community into a set of fair-share groups and allocates a fraction of the processor resource to each group, e.g. four groups, each with 25% of the processor usage.
- Each fair-share group is provided with a virtual system that runs proportionally slower than a full system.
- Scheduling is done on the basis of priority, which takes into account the underlying priority of the process, its recent processor usage, and the recent processor usage of the group to which the process belongs. The higher the numerical value of the priority, the lower the priority.

Fair-share schedulers (FSS)

- Each process is assigned a base priority.
- The priority of a process drops as the process uses the processor and as the group to which the process belongs uses the processor.
- In the case of the group utilization, the average is normalized by dividing by the weight of that group. The greater the weight assigned to the group, the less its utilization will affect its priority.



Time	Process A Priority	Process A Process CPU count	Process A Group CPU count	Process B Priority	Process B Process CPU count	Process B Group CPU count	Process C Priority	Process C Process CPU count	Process C Group CPU count
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			Group 2					

Colored rectangle represents executing process

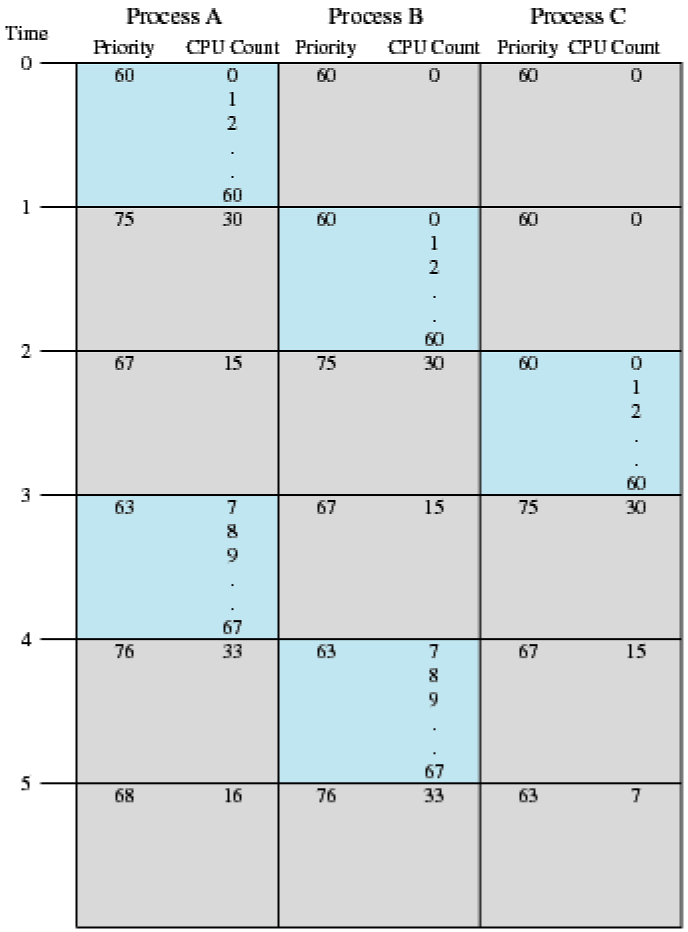
Figure 9.16 Example of Fair Share SchedulerÑThree Processes, Two Groups

Traditional UNIX Scheduling

- Multilevel feedback using round robin within each of the priority queues
- If a running process does not block or complete within 1 second, it is preempted
- Priorities are recomputed once per second
- Base priority divides all processes into fixed bands of priority levels

Bands

- These bands are used to optimize access to block devices (e.g., disk) and to allow the operating system to respond quickly to system calls.
- Decreasing order of priority
  - Swapper
  - Block I/O device control
  - File manipulation
  - Character I/O device control
  - User processes



Time	Process A		Process B		Process C	
	Priority	CPU Count	Priority	CPU Count	Priority	CPU Count
0	60	0 1 2 . . 60	60	0	60	0
1	75	30	60	0 1 2 . . 60	60	0
2	67	15	75	30	60	0 1 2 . . 60
3	63	7 8 9 . . 67	67	15	75	30
4	76	33	63	7 8 9 . . 67	67	15
5	68	16	76	33	63	7

Colored rectangle represents executing process

Figure 9.17 Example of Traditional UNIX Process Scheduling