Uniprocessor Scheduling

Chapter 9

Contents

- Types of scheduling
- Scheduling algorithms
- Traditional Unix scheduling

Types of Scheduling

- Long-term
  - performed when new process is created
  - the decision to add to the pool of processes to be executed
- Medium-term
  - the decision to add to the number of processes that are partially or fully in main memory

Types of Scheduling

- Short-term
  - the decision as to which ready process will be executed by the processor
- I/O
  - the decision as to which process's pending I/O request shall be handled by available I/O device

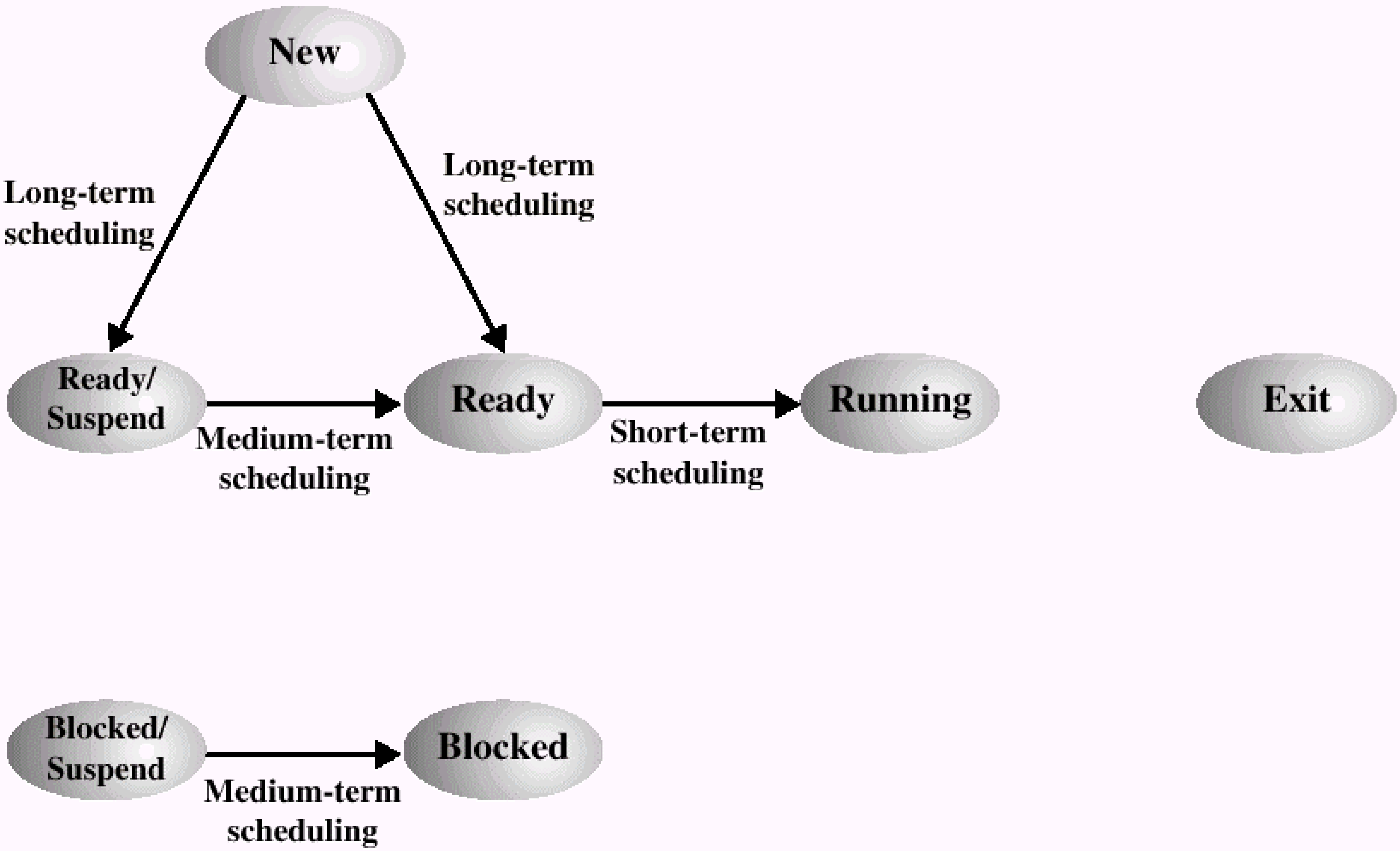


Figure 9.1 Scheduling and Process State Transitions

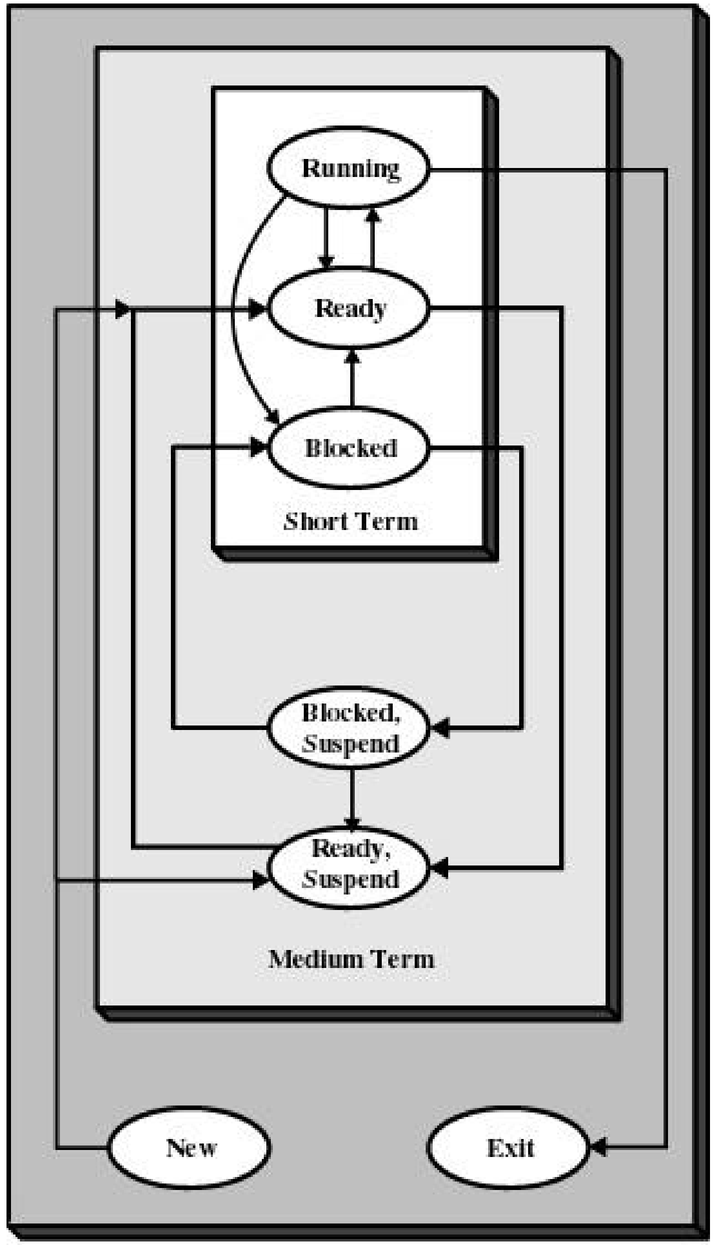


Figure 9.2 Levels of Scheduling

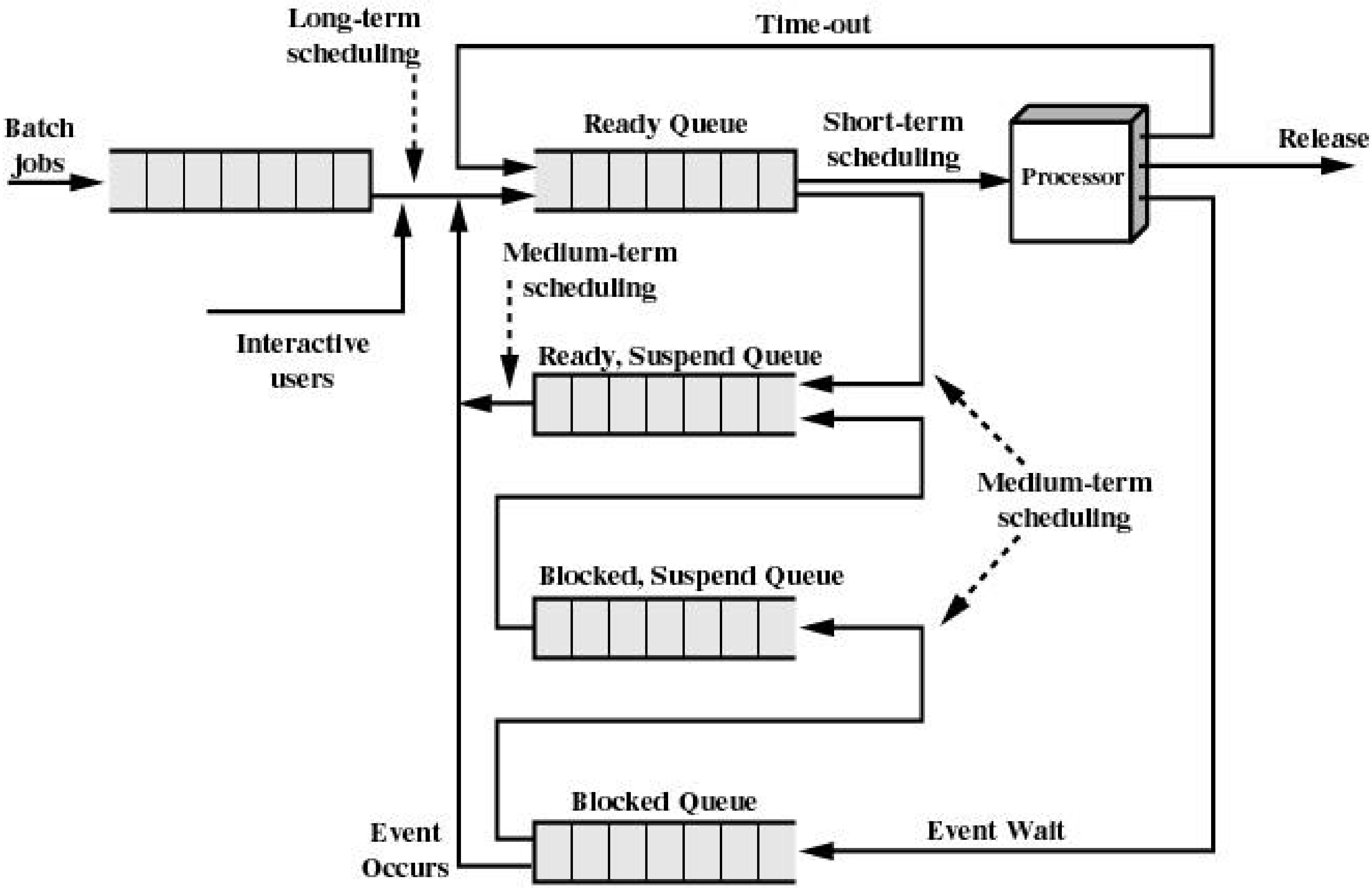


Figure 9.3 Queuing Diagram for Scheduling

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
- Swapping-in decision is based on the need to manage the degree of multiprogramming

Short-Term Scheduling

- Short-term scheduler is known as the dispatcher
- Invoked when following events occur
  - clock interrupts
  - I/O interrupts
  - operating system calls
  - signals

Short-Tem Scheduling Criteria

- User-oriented, Performance Related
- User-oriented, Other
- System-oriented, Performance Related
- System-oriented,Other

Short-Tem Scheduling Criteria

- User-oriented, Performance Related
  - Response Time
    - time from the submission of a request until the response
  - Turnaround Time
    - interval of time between the submission of a process and its completion
  - Deadline
    - meet the deadline

Short-Tem Scheduling Criteria

- 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

Short-Term Scheduling Criteria

- System-oriented, Performance Related
  - Throughput
    - number of processes completed per unit of time
    - a measure of how much work is being done
  - Processor Utilization
    - the percentage of time that the processor is busy

Short-Term Scheduling Criteria

- System-oriented,Other
  - Fairness
    - processes should be treated the same
    - no process should suffer starvation
  - Enforcing Priorities
    - when priorities are assigned, higher priority process should be favored
  - Balancing Resources
    - keep the resources of the system busy

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
  - allow a process to change its priority based on its age or execution history

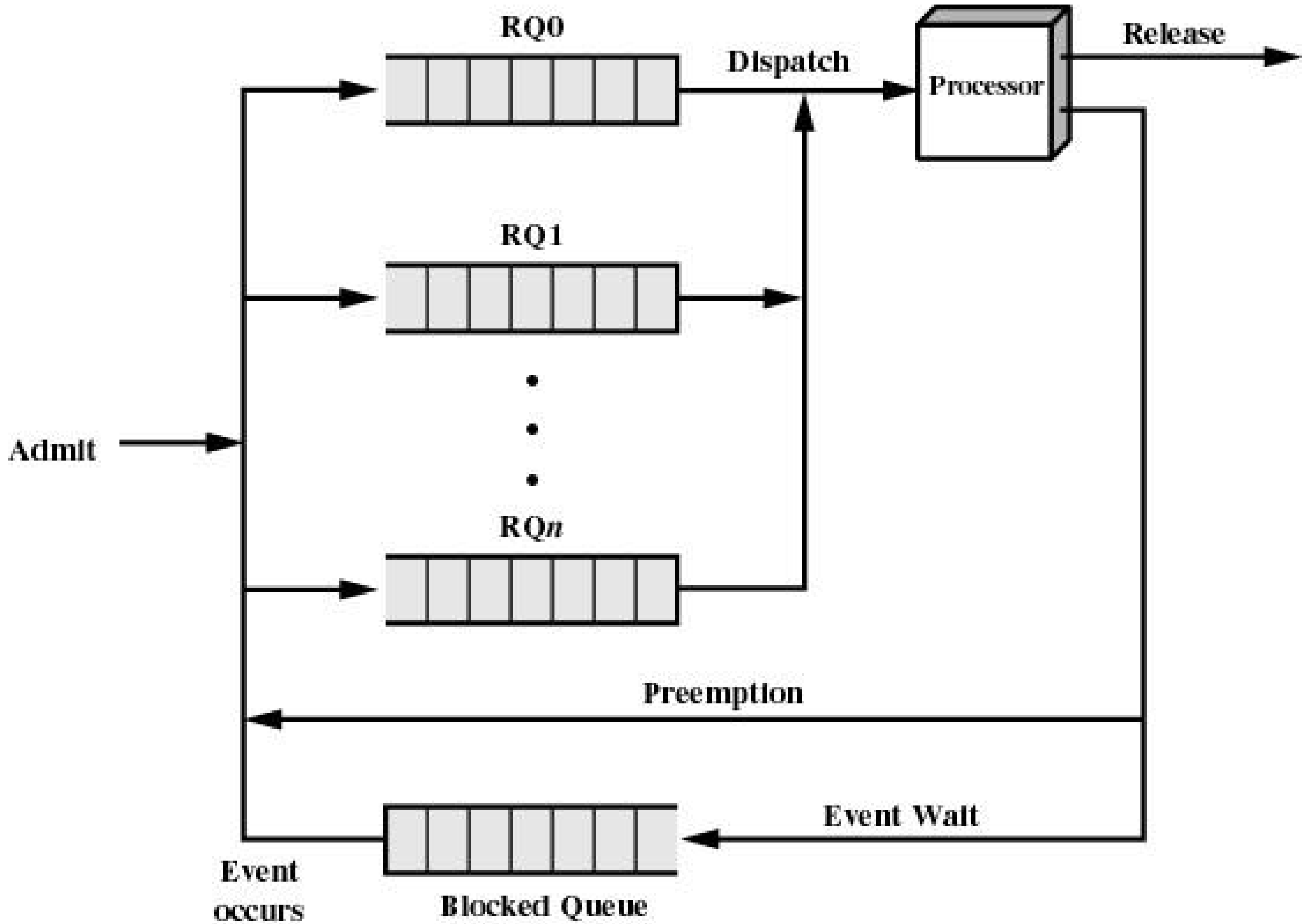


Figure 9.4 Priority Queuing

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 OS
  - Allows for better service since any one process cannot monopolize the processor for very long

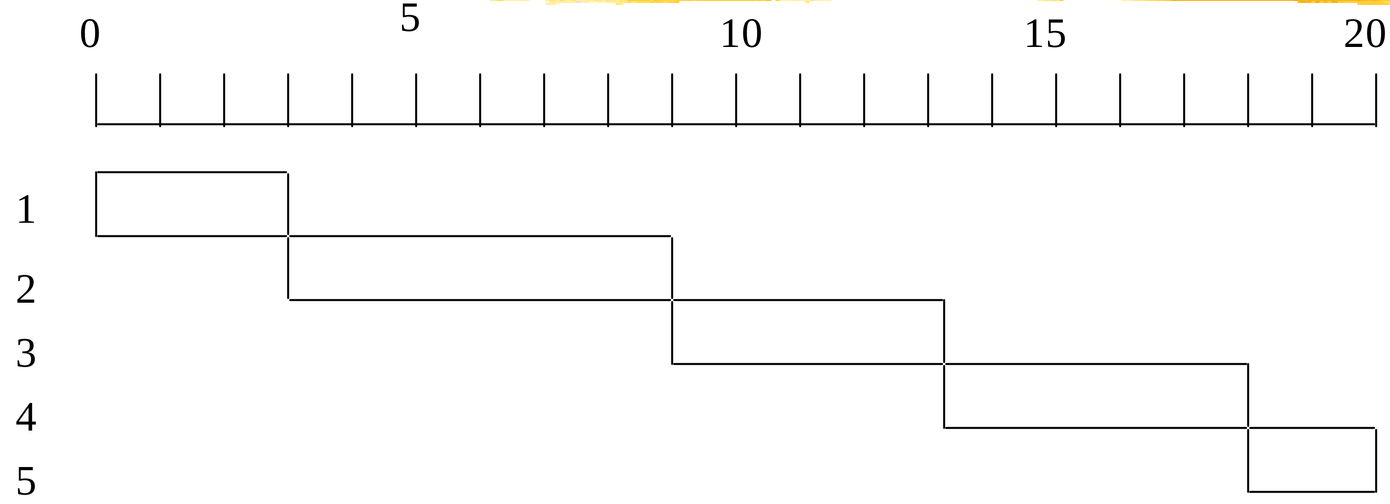
Scheduling Algorithms

- First-Come-First-Served
- Round-Robin
- Shortest Process Next
- Shortest Remaining Time
- Highest Response Ratio Next
- Feedback

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)

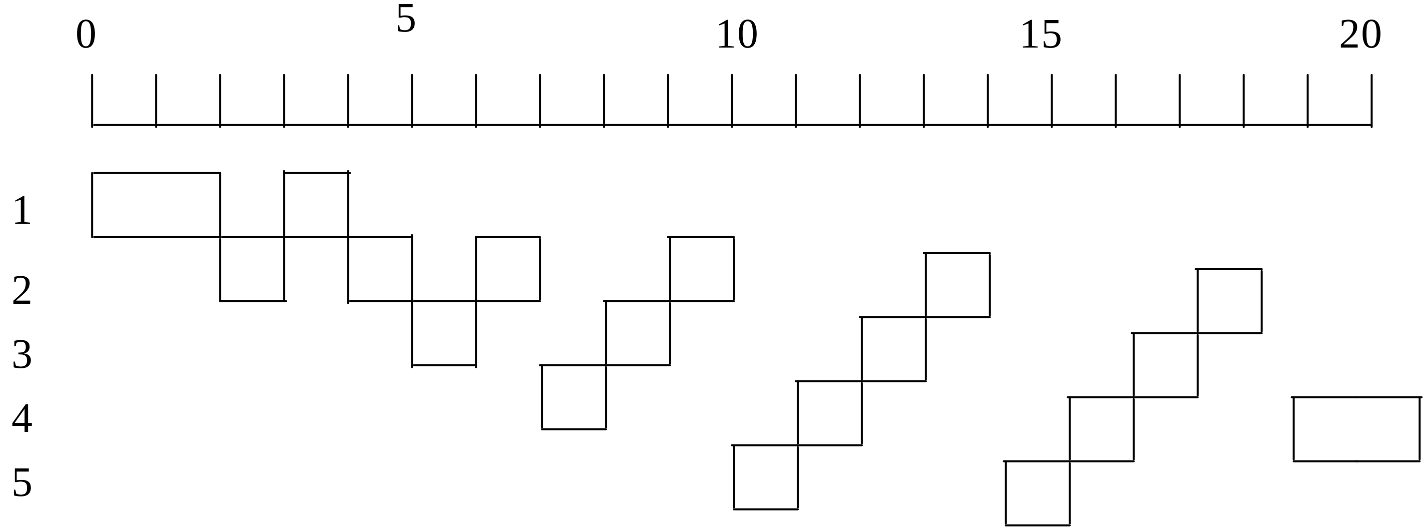


- As each process becomes ready, it joins the Ready queue
- When the current process ceases to execute, the oldest process in the Ready queue is selected

First-Come-First-Served (FCFS)

- Perform much better for long processes
  - 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
- Not an attractive method

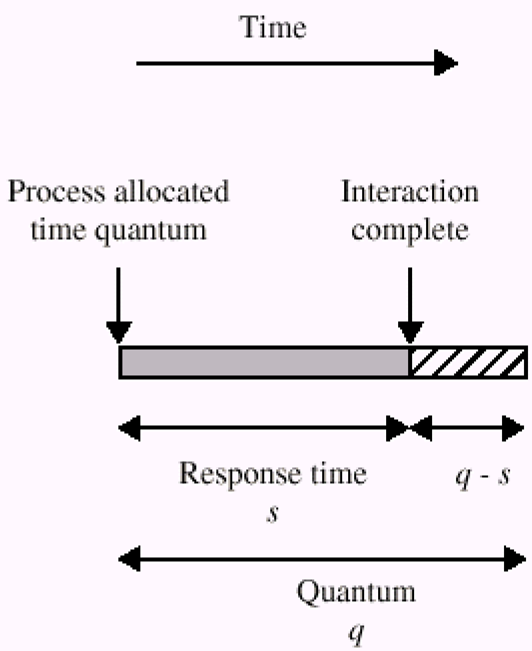
Round-Robin



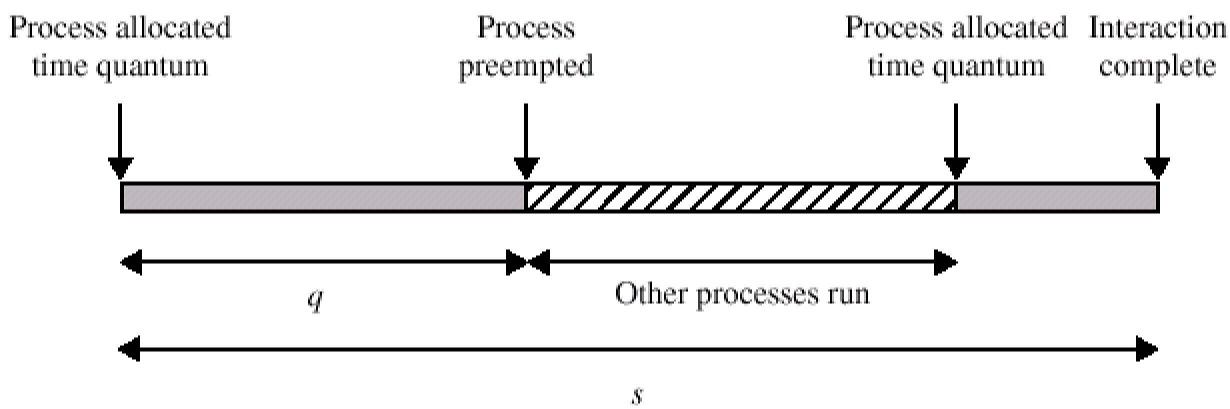
- Uses preemption based on a clock
- An amount of time is determined that allows each process to use the processor for that length of time

Round-Robin

- Clock interrupt is generated at periodic intervals
  - when an interrupt occurs, the currently running process is placed in the read queue
  - next ready job is selected
  - known as time slicing
- Principal design issue is the length of time quantum
  - should be slightly greater than the time required for a typical interaction



(a) Time quantum greater than typical interaction



(b) Time quantum less than typical interaction

Figure 9.6 Effect of Size of Preemption Time Quantum

Round-Robin

- Relatively favors the processor-bound job
  - I/O-bound process uses a processor for a short period and then is blocked for I/O
  - after waking up, it joins the ready queue
- Poor performance for I/O-bound processes
  - inefficient use of I/O devices
  - increase in the variance of response time
- Virtual Round-Robin Scheduler

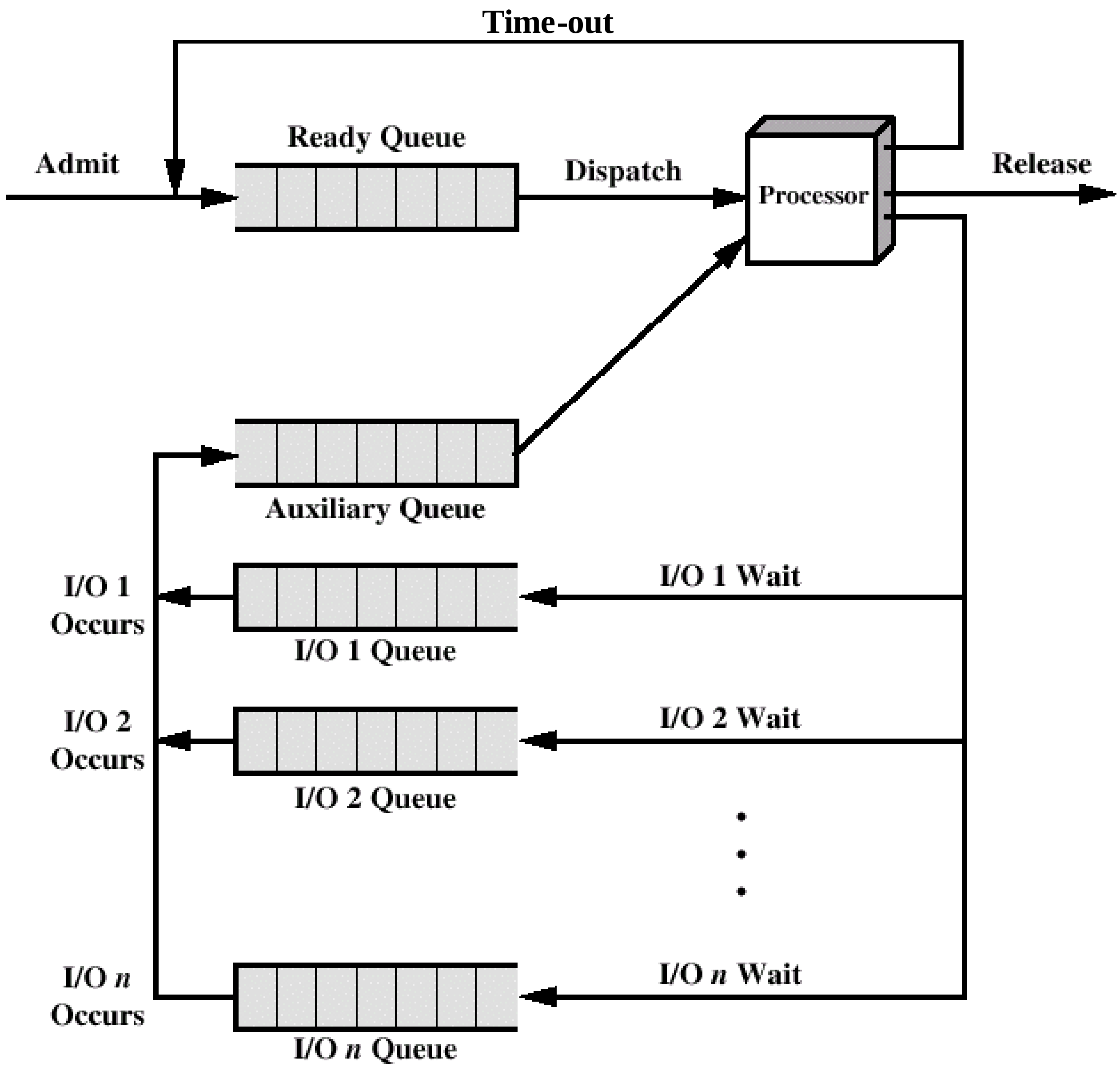
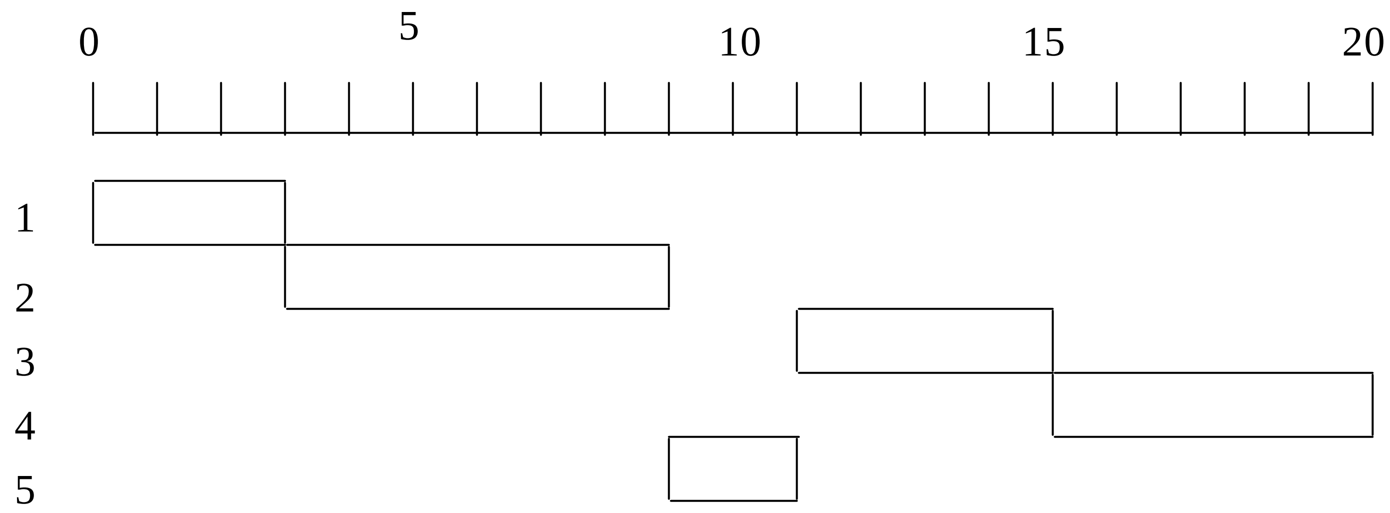


Figure 9.7 Queuing diagram for virtual round-robin scheduler

Shortest Process Next

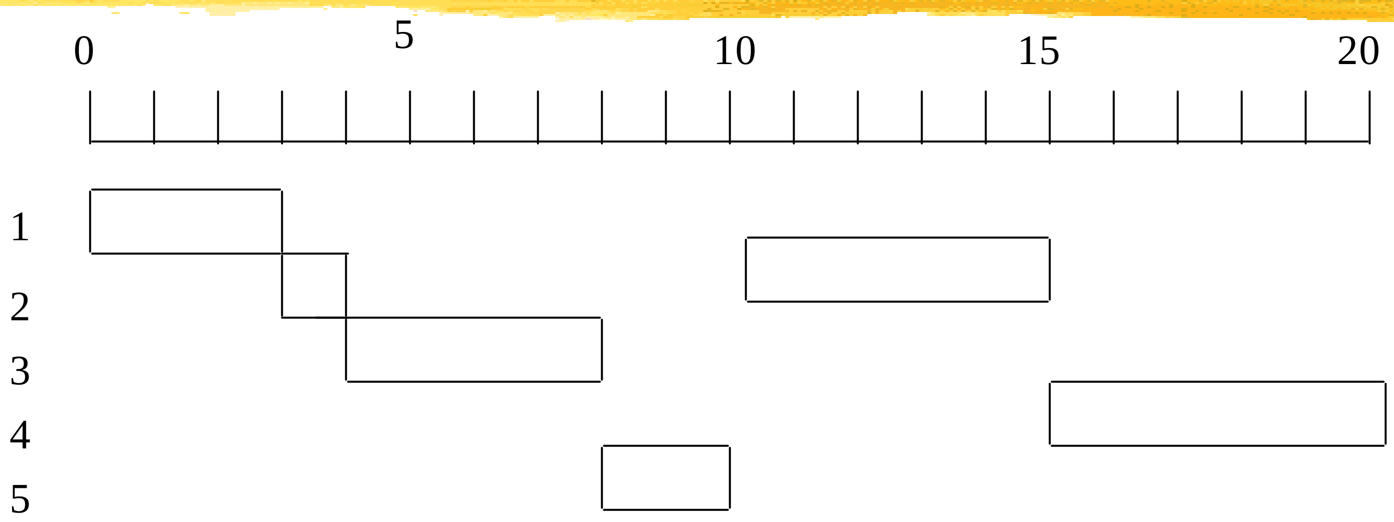


- Nonpreemptive policy
- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes

Shortest Process Next

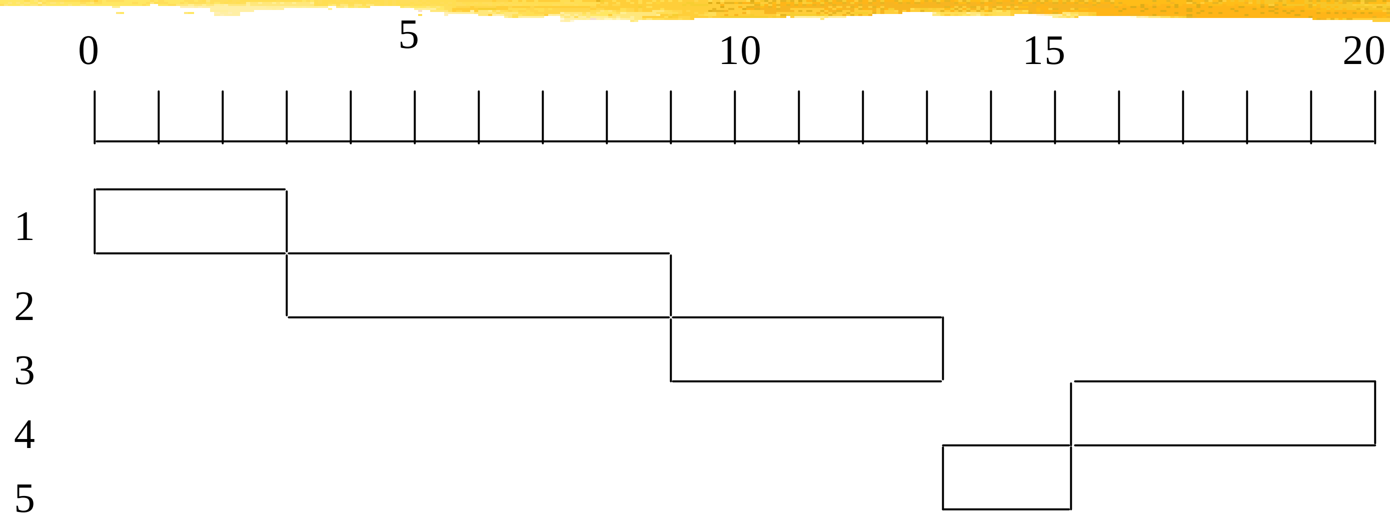
- Need to estimate the required processing time
  - in a production environment, same jobs run frequently and statistics may be gathered
  - if estimated time for process not correct, the operating system may abort it
- Predictability of longer processes is reduced
- Possibility of starvation for longer processes

Shortest Remaining Time



- Preemptive version of shortest process next policy
- Must estimate processing time

Highest Response Ratio Next (HRRN)



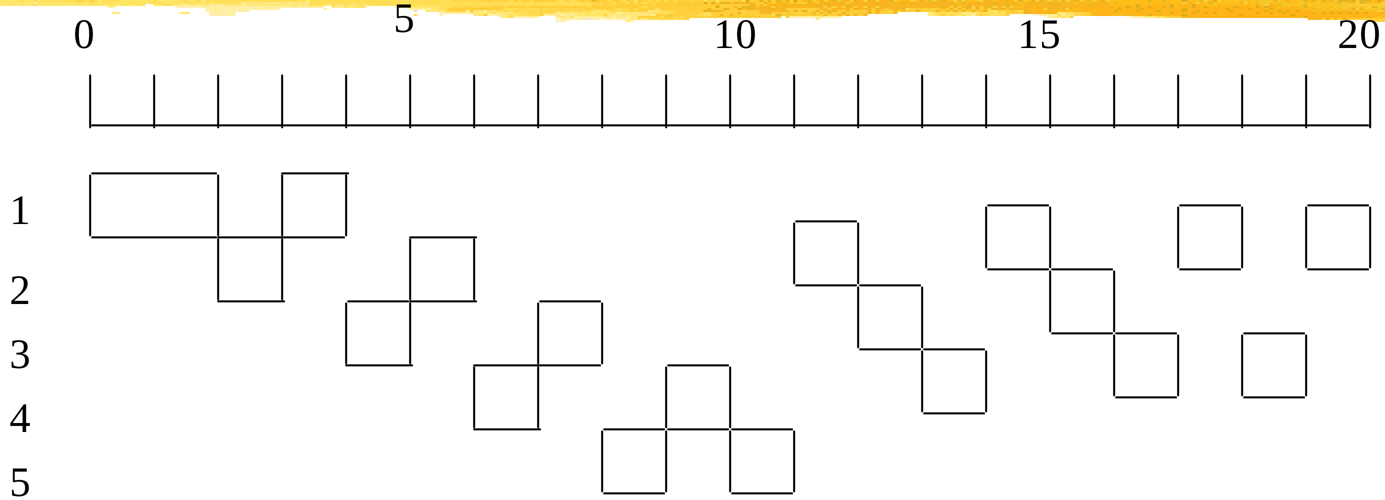
- Choose next process with the highest ratio

$$\frac{\textbf{time spent waiting + expected service time}}{\textbf{expected service time}}$$

Highest Response Ratio Next (HRRN)

- Minimum value of ratio is 1.0
- Count for the age of the process
  - generally 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

Feedback



- Used when we don’t know remaining time process needs to execute
  - decision based on the past
  - penalize jobs that have been running longer

Feedback

- Process is demoted to the next lower-priority queue each time it returns to the ready queue
- Longer processes drift downward
- To avoid starvation, we can vary the preemption times according to the queue

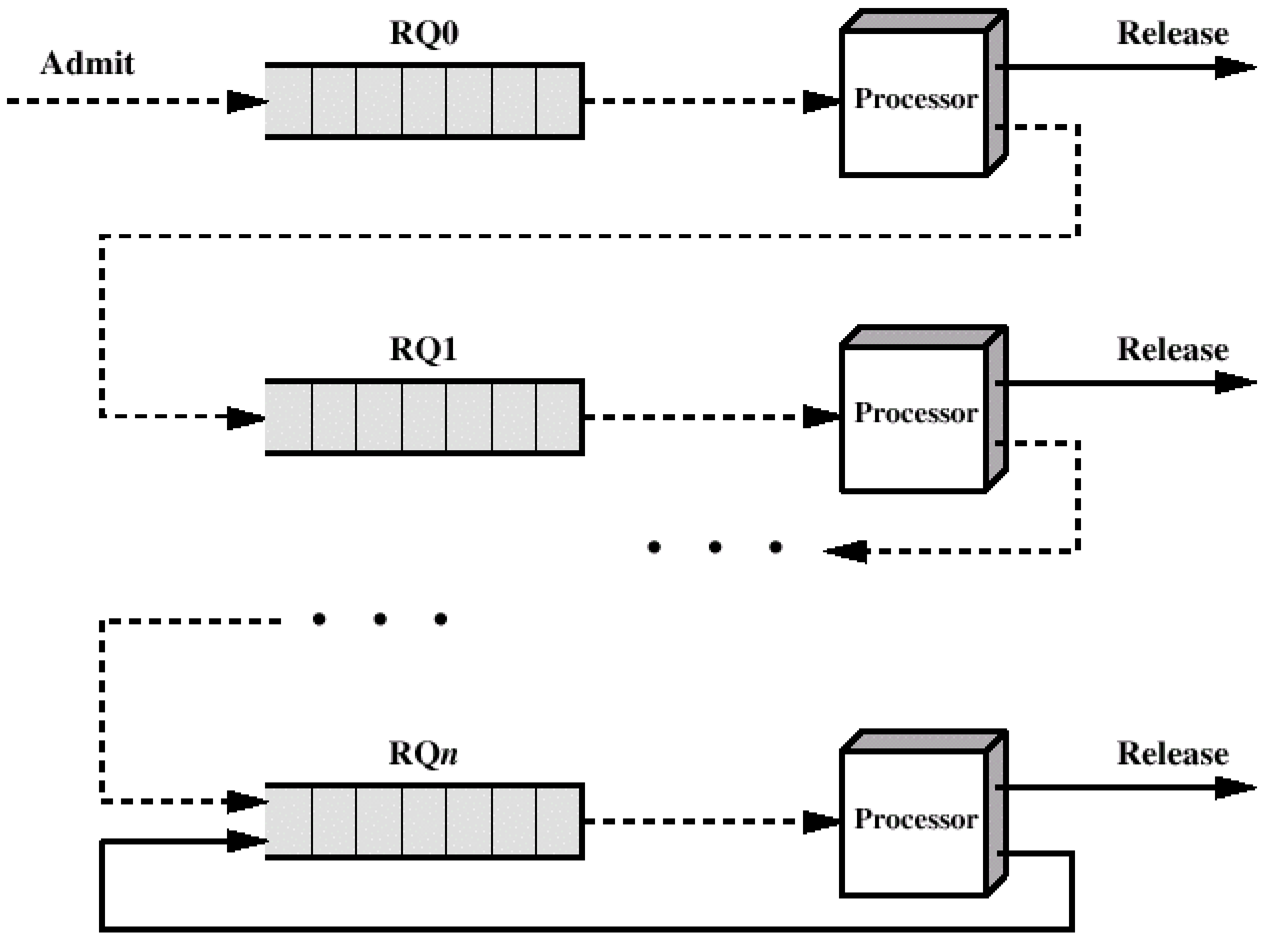


Figure 9.10 Feedback scheduling

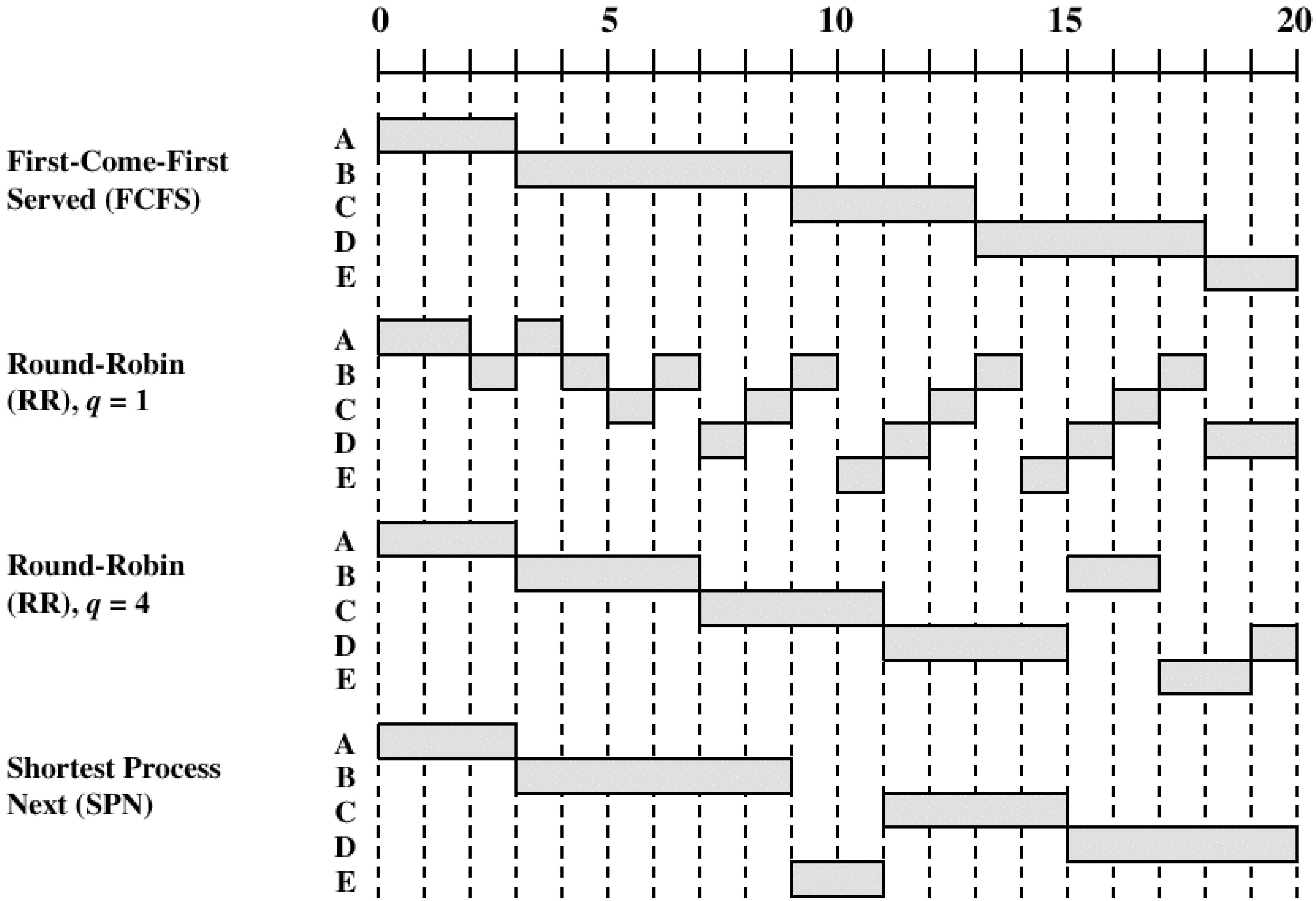


Figure 9.5 A comparison of scheduling policies

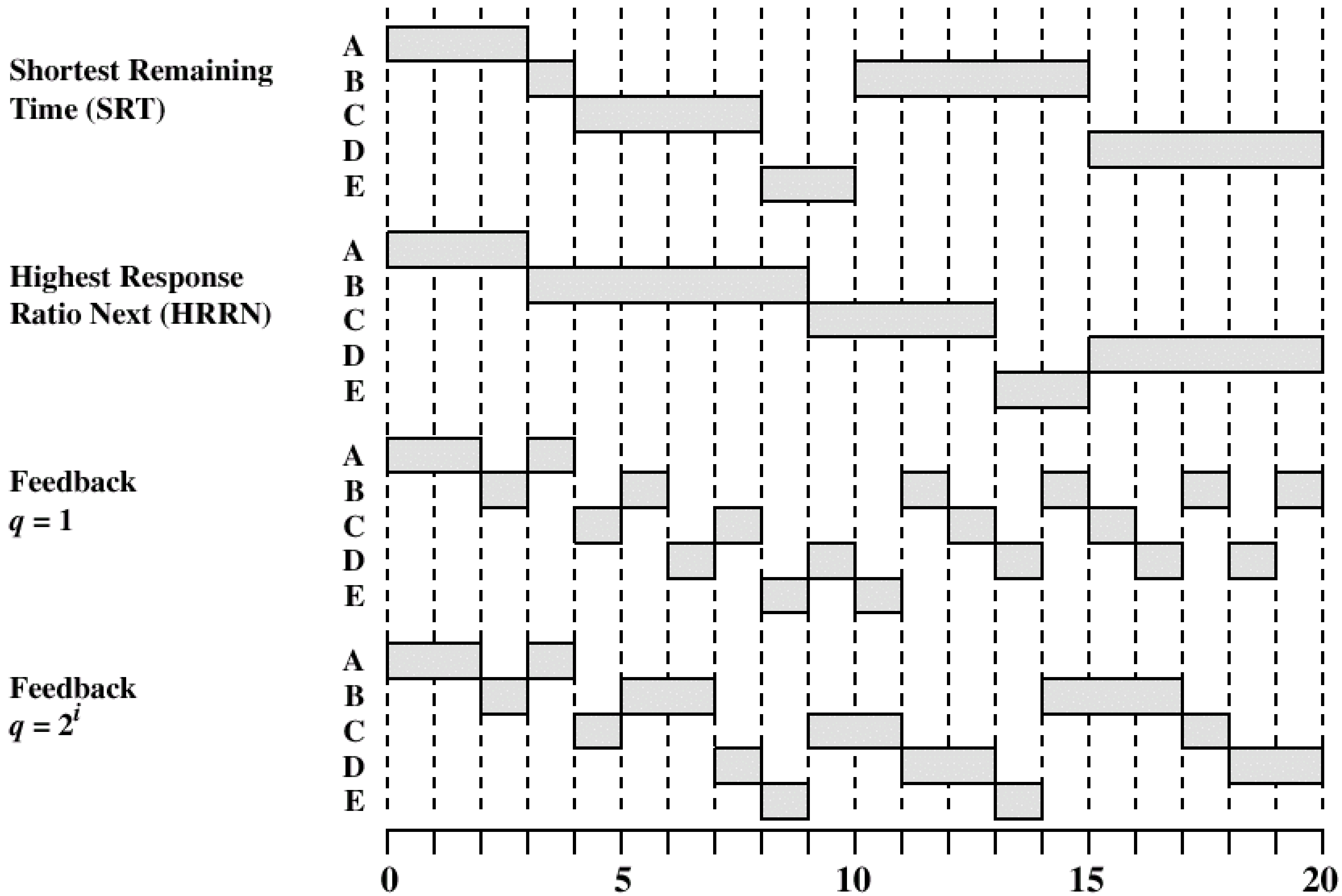


Figure 9.5 A comparison of scheduling policies

Fair-share Scheduling

- User's application runs as a collection of processes (threads)
- User is concerned about the performance of the application
- Need to make scheduling decisions based on groups of processes

Time	Process A			Process B			Process C		
	Priority	Process	Group	Priority	Process	Group	Priority	Process	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			**Group 2**					

Shaded rectangle represents executing process

Figure 9.16 Example of Fair Share Scheduler Three Processes, Two Groups

UNIX Scheduling

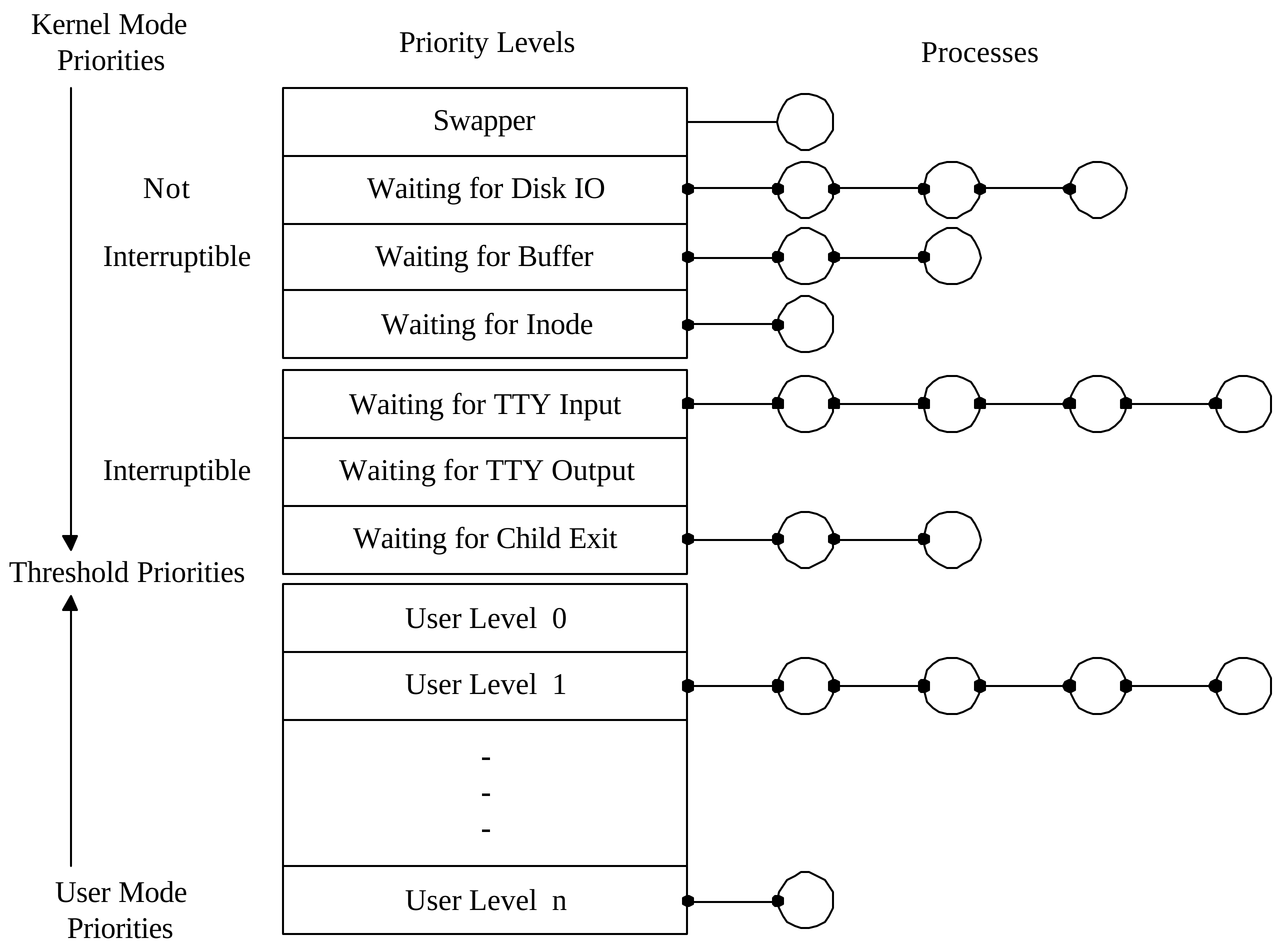
- Multilevel feedback using round-robin within each of the priority queues
- Priorities are recomputed once per second
- Base priority divides all processes into fixed bands of priority levels

UNIX Scheduling

- Bands in decreasing order of priority
  - Swapper
  - Block I/O device control
  - File manipulation
  - Character I/O device control
  - User processes

Bands of process priorities

user and kernel priorities



UNIX Scheduling

- Formulas to calculate the priority

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

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j$$

$CPU_j(i-1)$ = Measure of processor utilization by process j through interval i
$P_j(i)$ = Priority of process j at beginning of interval i: lower values equal higher priorities
$Base_j$ = Base priority of process j
$nice_j$ = user-controllable adjustment factor

Time	Process A Priority	Process A CPU Count	Process B Priority	Process B CPU Count	Process C Priority	Process C 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

Shaded rectangle represents executing process

Figure 9.17 Example of Traditional UNIX Process Scheduling