Uniprocessor Scheduling

Topics

- Types of Processor Scheduling
- Scheduling Algorithms
- Examples

Scheduling

- An OS must allocate resources amongst competing processes.
- The resource provided by a processor is execution time
  - The resource is allocated by means of a schedule

CPU - I/O Cycle

- Maximum CPU utilization is obtained with context switching -> CPU Scheduling
- CPU-I/O Cycle – Process execution consists of a cycle of
  - CPU execution (CPU burst time), and
  - I/O wait (I/O burst time)

Scheduling Objectives

- The scheduling function should
  - Share time ***fairly*** among processes
  - Prevent starvation of a process
  - Use the processor efficiently
  - Have low overhead
  - Prioritise processes when necessary (e.g. real time deadlines)

When to Schedule

- Required on two occasions:
  - When a process exits
  - When a process blocks on I/O or a semaphore (more on this later)
- Other occasions:
  - When a new process is created
  - When an I/O interrupt occurs

Types of Scheduling

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

Five-State Process Model

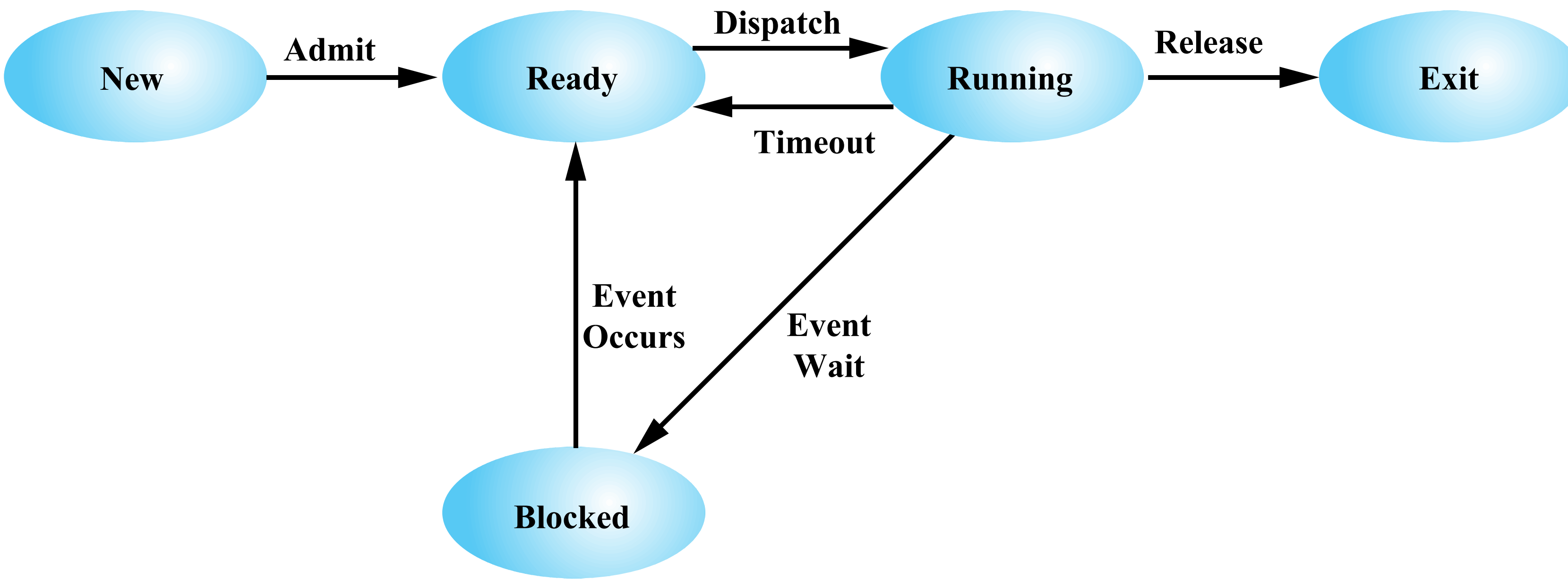
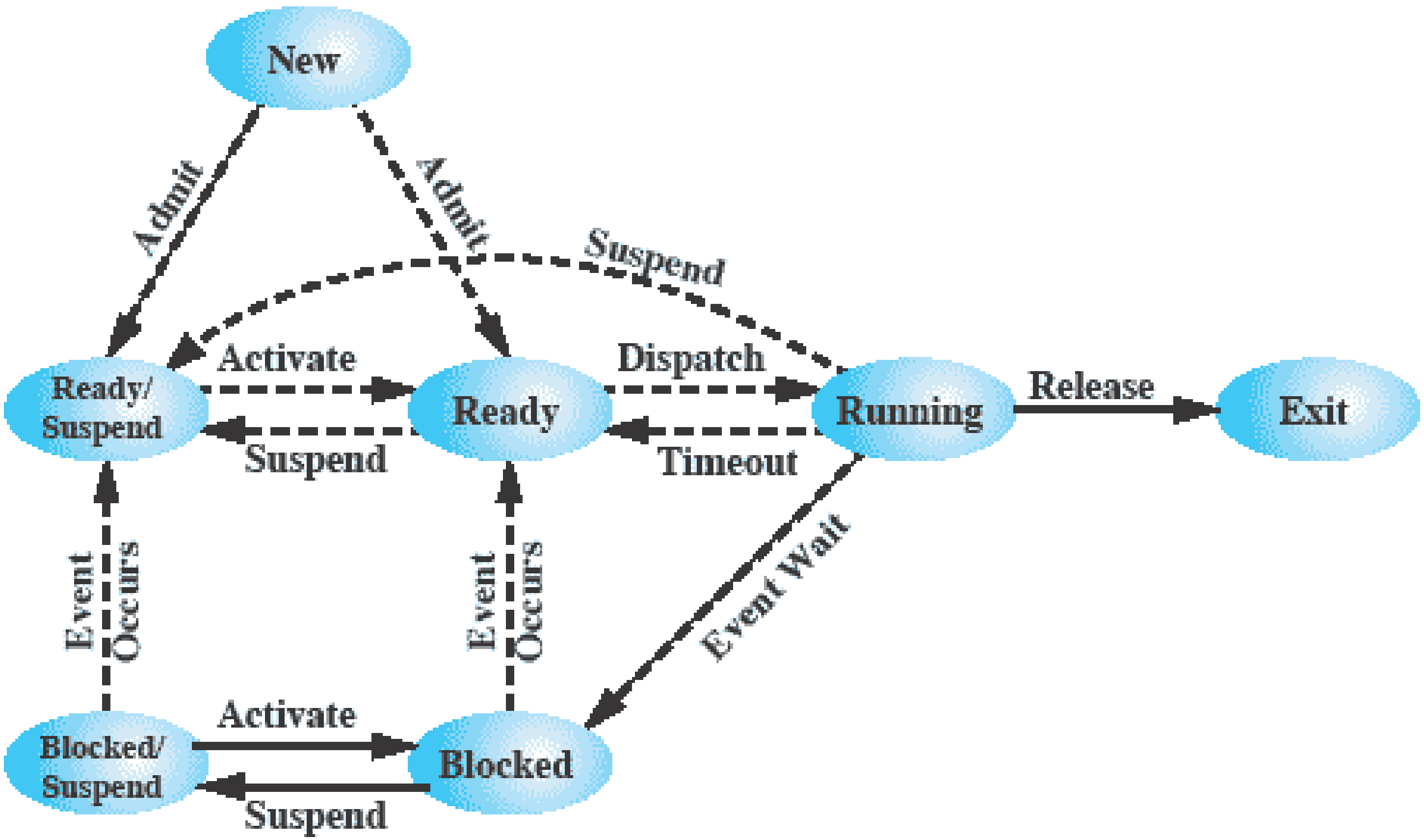


Figure 3.6 Five-State Process Model

Two Suspend States



(b) With Two Suspend States

Scheduling and Process State Transitions

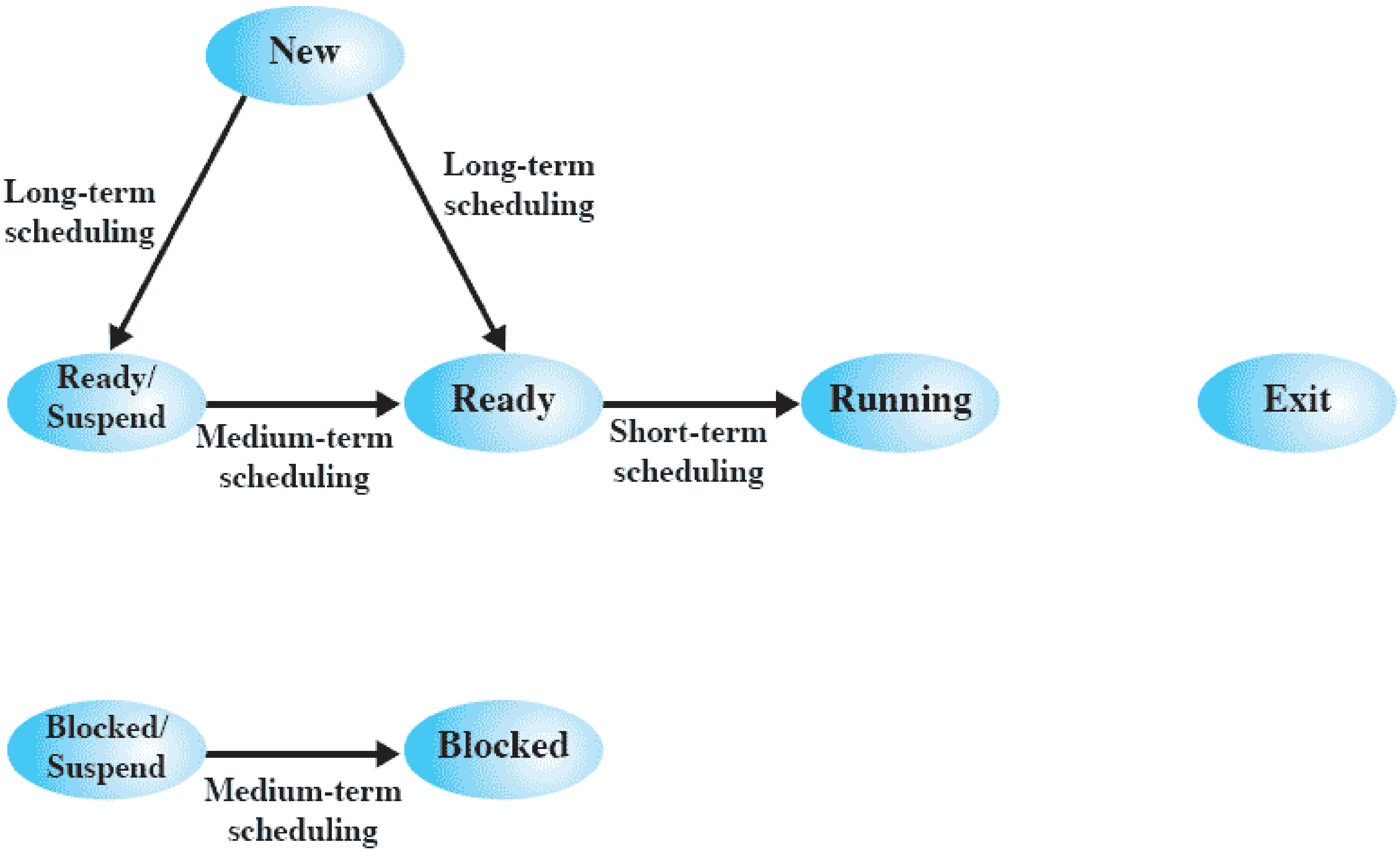
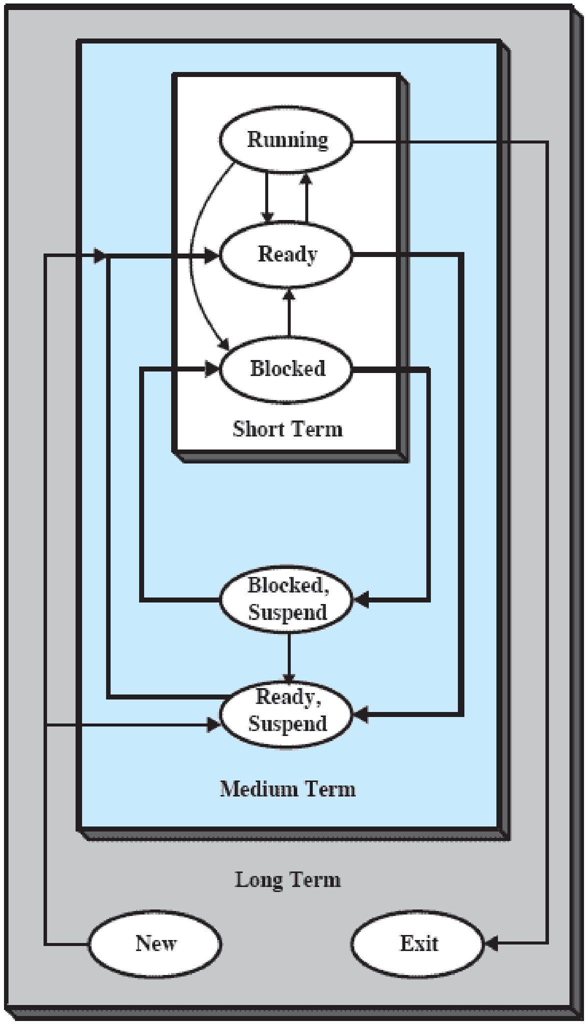


Figure 9.1 Scheduling and Process State Transitions

Nesting of Scheduling Functions



Queuing Diagram

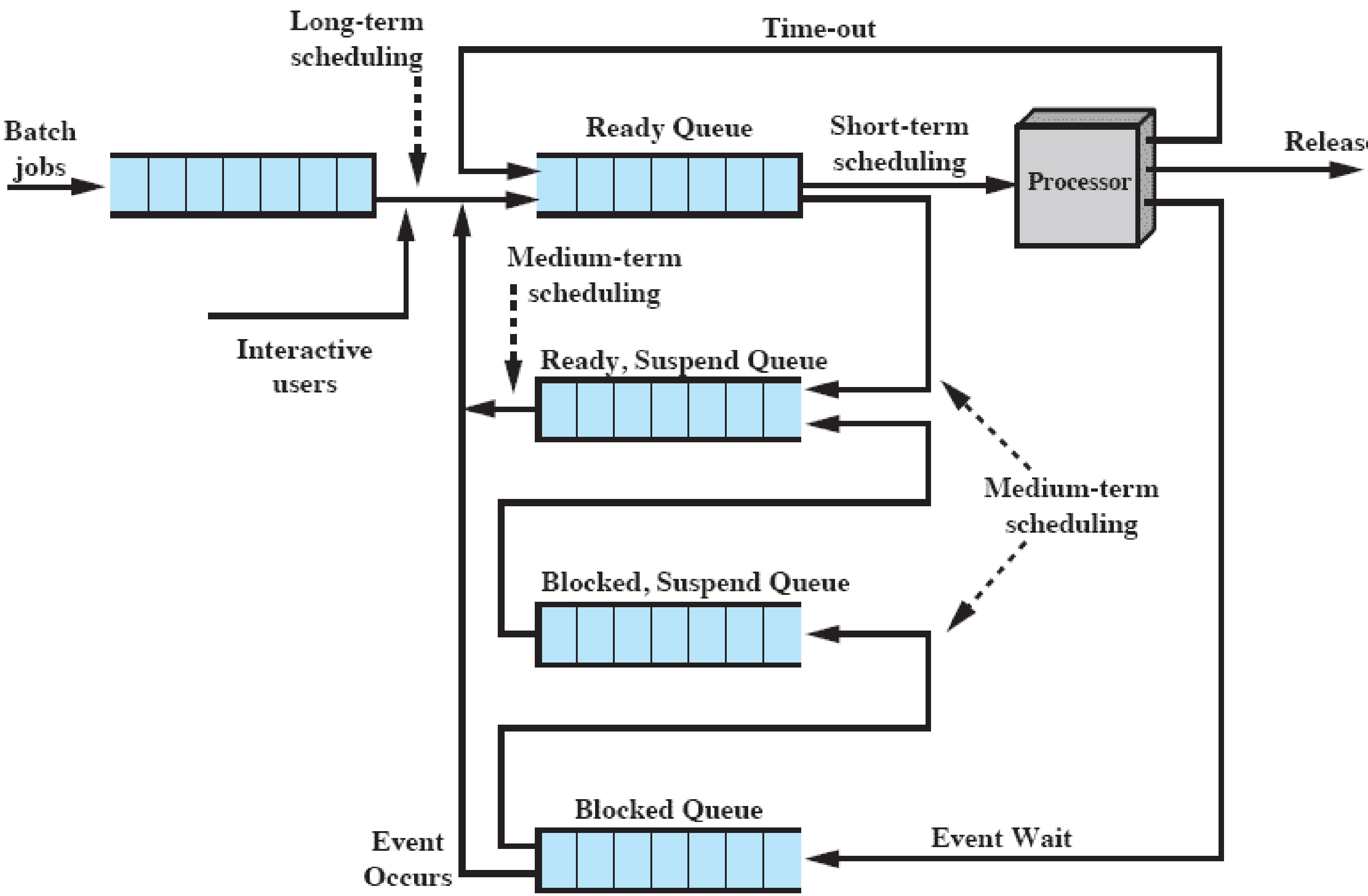


Figure 9.3 Queuing Diagram for Scheduling

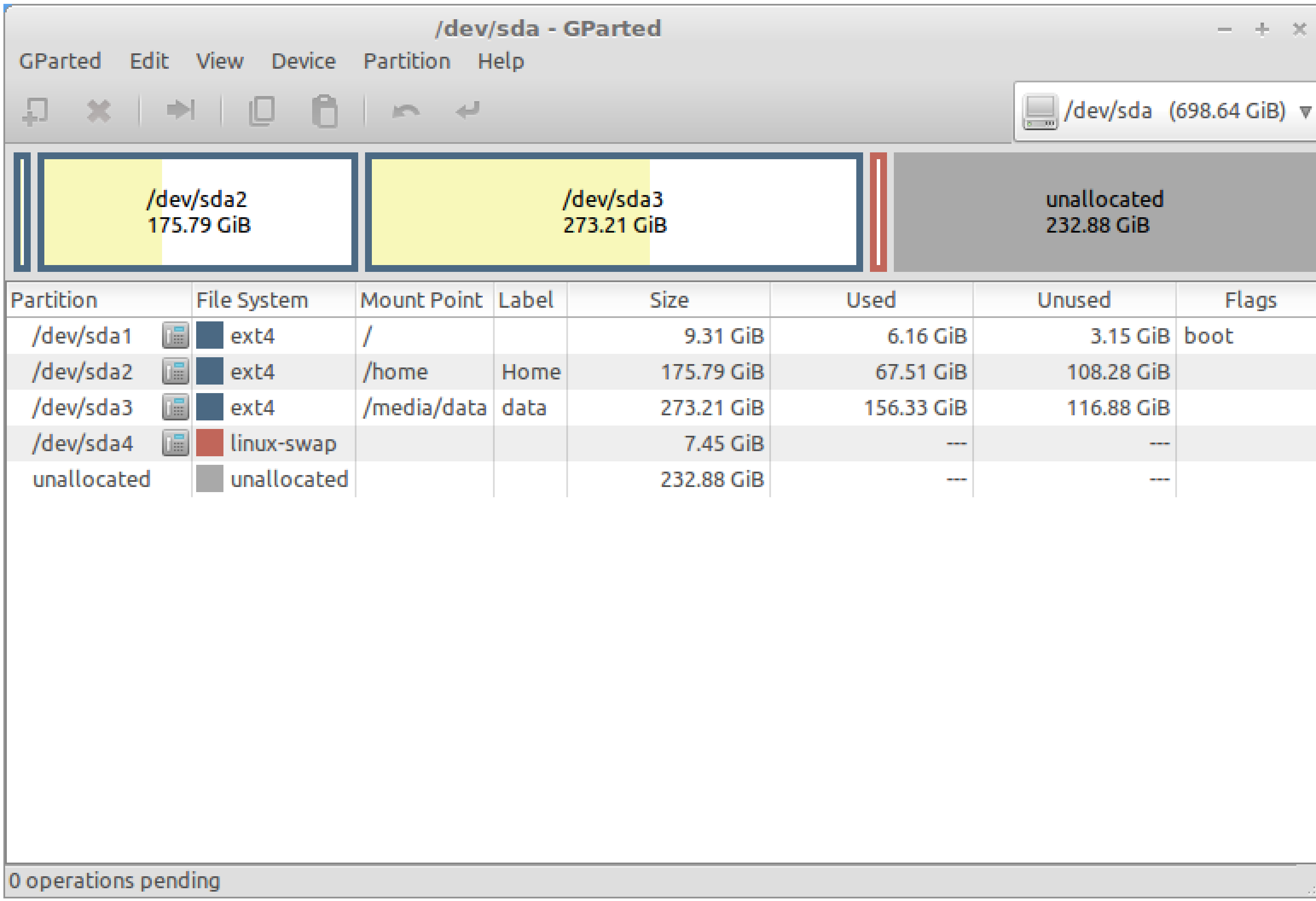
Long-Term Scheduling

- 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

Medium-Term Scheduling

- Part of the swapping function
- Swapping-in decisions are based on the need to manage the degree of multiprogramming

GParted


/dev/sda - GParted
GParted Edit View Device Partition Help
/dev/sda (698.64 GiB)
/dev/sda2
175.79 GiB
/dev/sda3
273.21 GiB
unallocated
232.88 GiB
Partition File System Mount Point Label Size Used Unused Flags
/dev/sda1 ext4 / 9.31 GiB 6.16 GiB 3.15 GiB boot
/dev/sda2 ext4 /home Home 175.79 GiB 67.51 GiB 108.28 GiB
/dev/sda3 ext4 /media/data data 273.21 GiB 156.33 GiB 116.88 GiB
/dev/sda4 linux-swap 7.45 GiB --- ---
unallocated unallocated 232.88 GiB --- ---
0 operations pending

Short-Term Scheduling

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

Topics

- Types of Processor Scheduling
- Scheduling Algorithms
- Examples

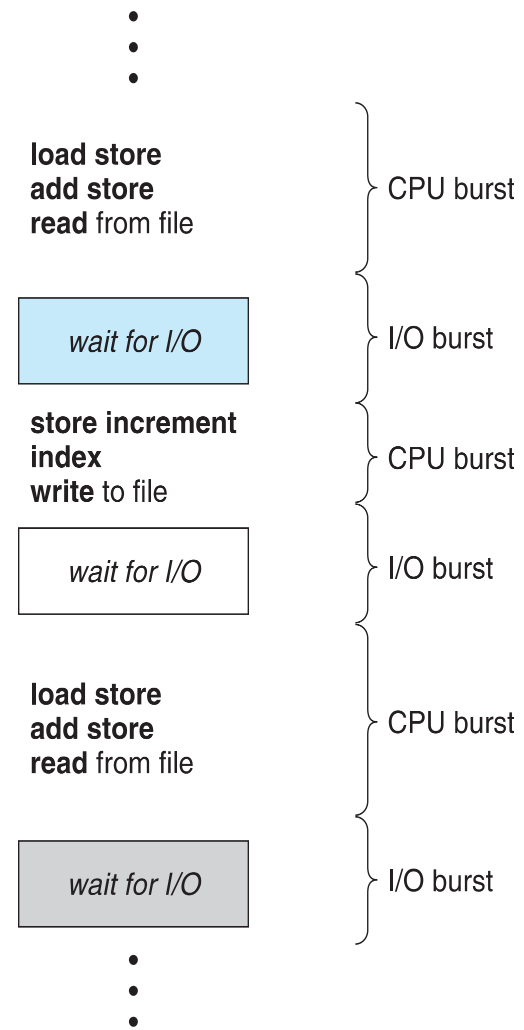
Behavior of Processes in Execution

- Which do you think is better: Having the scheduler favor I/O-bound processes or CPU bound processes or neutral?

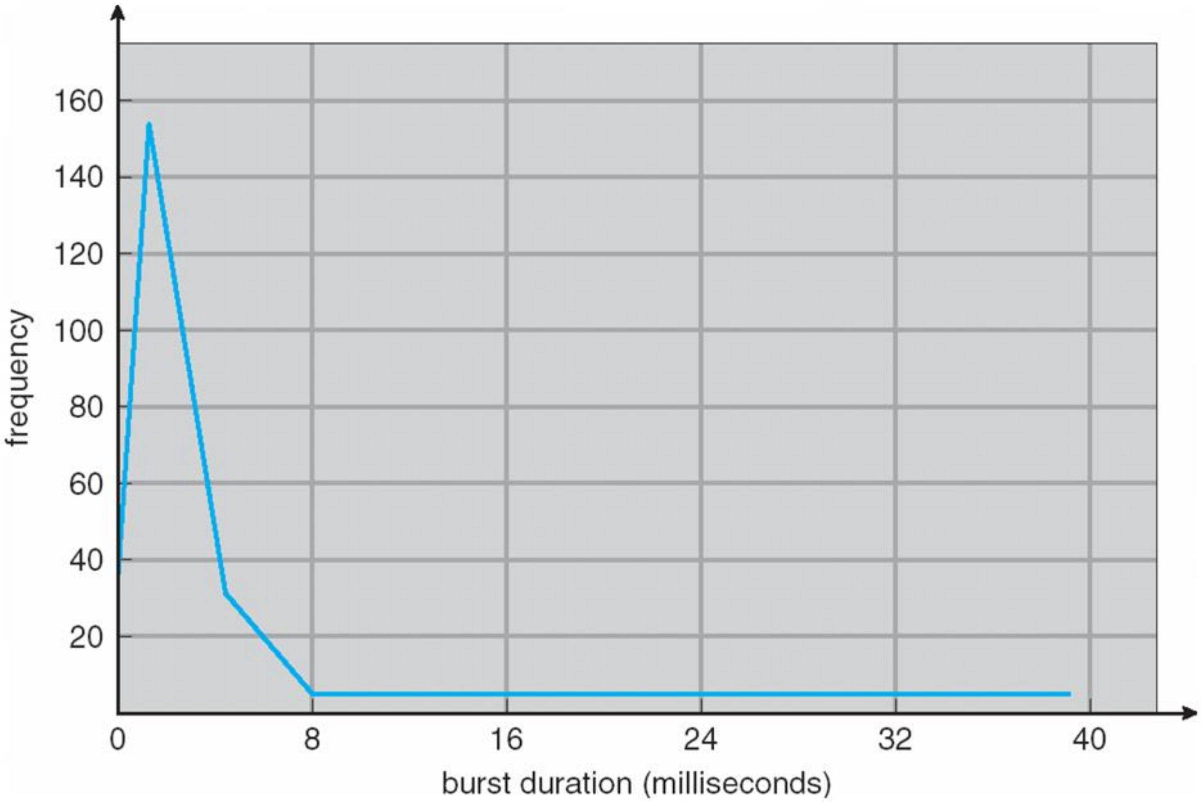
- Necessary to determine as quickly as possible the nature (CPU-bound or I/O-bound) of a process, since usually not known in advance.

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait
- CPU burst followed by I/O burst
- CPU burst distribution is of main concern



Histogram of CPU-burst Times



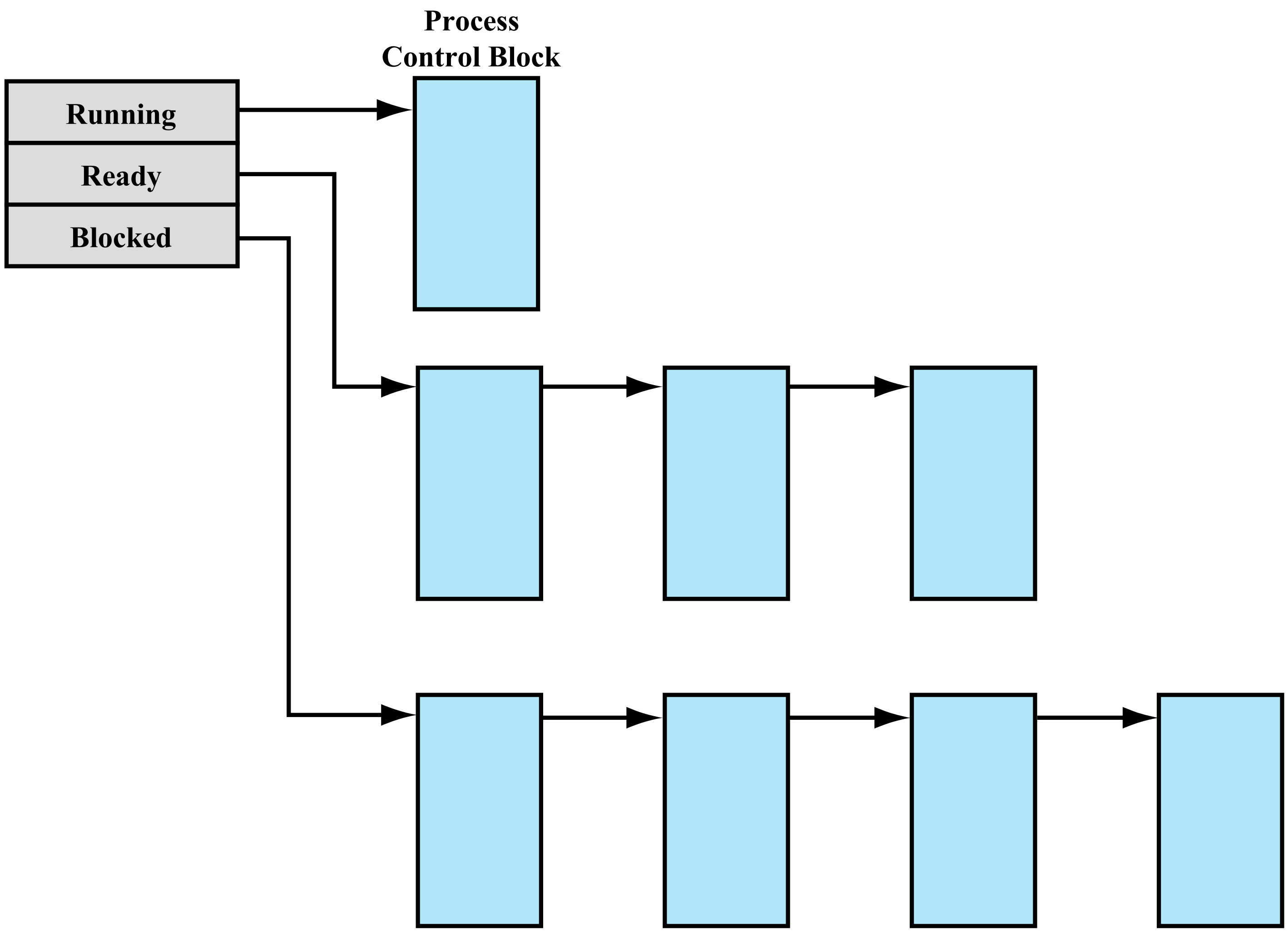


Figure 3.14 Process List Structures

CPU Scheduler

- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways

Short-Term Scheduling Criteria: User vs System

- We can differentiate between user and system 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: User vs System

- We can differentiate between user and system 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

CPU Scheduler

- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to read
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

When to Schedule

- Non-preemptive
  - Picked process runs until it voluntarily relinquishes CPU
    - Blocks on an event e.g., I/O or waiting on another process
    - Process terminates
- Preemptive
  - Picked process runs for a maximum of some fixed time; or until it terminates.
  - Picked process voluntarily relinquishes CPU
  - Requires a clock interrupt to occur at the end of the time interval to give control of the CPU back to the scheduler

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

Dispatcher process activity

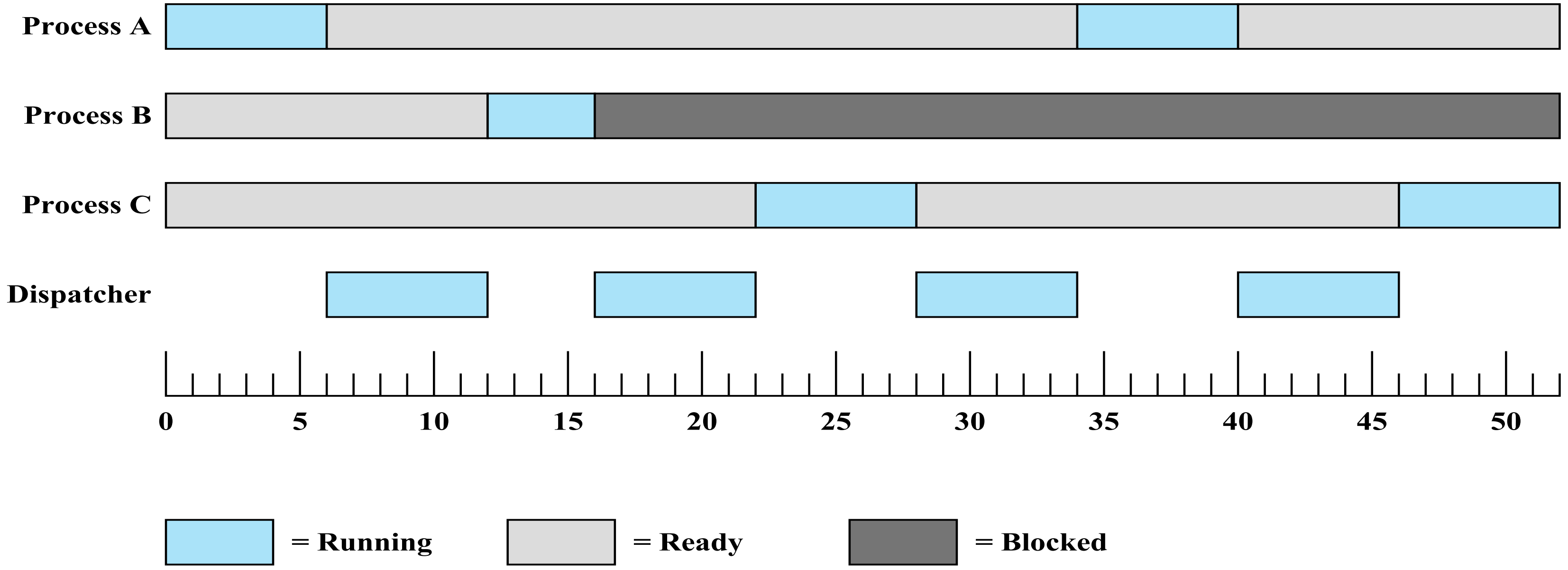


Figure 3.7 Process States for Trace of Figure 3.4

Scheduling Evaluation Metrics

- Many quantitative criteria for evaluating a scheduling algorithm:
  - CPU utilization: Percentage of time the CPU is not idle
  - Throughput: Completed processes per time unit
  - Turnaround time: Submission to completion
  - Waiting time: Time spent on the ready queue
  - Response time: Response latency. Amount of time it takes from when a request was submitted until the first response is produced.
  - Predictability: Variance in any of these measures.
  - Fairness: No process suffers starvation.

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Scheduler options

- Priorities
  - May use priorities to determine who runs next
  - Dynamic vs. Static algorithms
    - Dynamically alter the priority of the tasks while they are in the system (possibly with feedback)
    - Static algorithms typically assign a fixed priority when the job is initially started.
- Preemptive vs. Nonpreemptive
  - Preemptive systems allow the task to be interrupted at any time so that the O.S. can take over again.

Priority Queuing

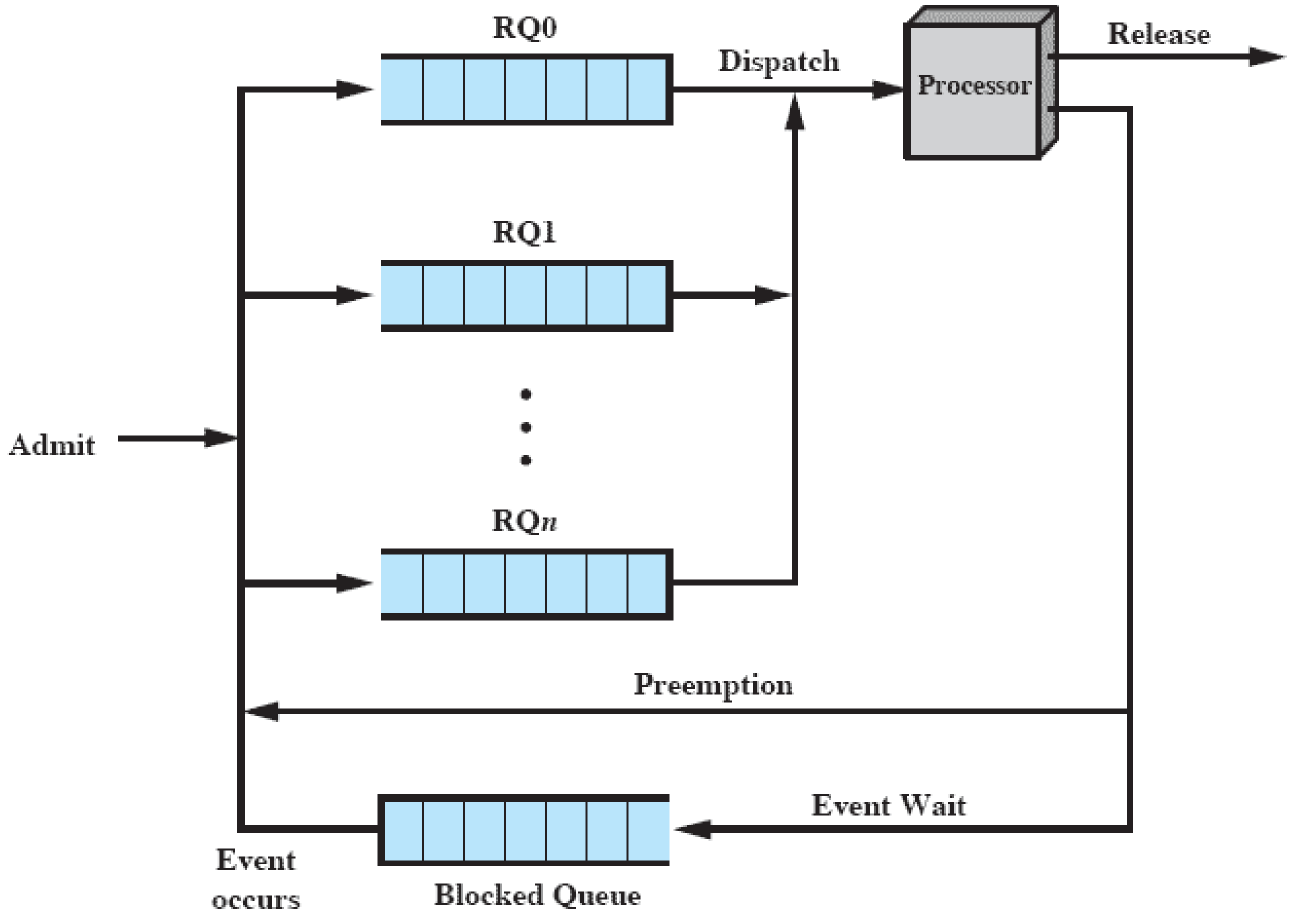


Figure 9.4 Priority Queuing

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

First-Come, First-Served (FCFS)

- The process that requests the CPU first is allocated the CPU first
- The code for FCFS scheduling is simple to write and understand
- We will illustrate the use of FCFS with three processes
- Nonpreemptive

First-Come, First-Served (FCFS) Scheduling

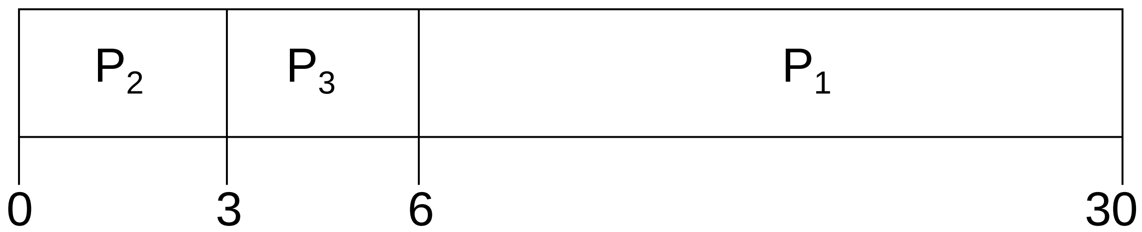
Process	Burst Time
P1	24
P2	3
P3	3

- Suppose the processes arrive at the same time and are executed in the order: P1 , P2 , P3 The Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling

- Suppose that the processes arrive at the same time and are executed in the order P_2 , P_3 , P_1
- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- Convey effect short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

FCFS Scheduling

- Consider a scenario with one CPU-bound process and many I/O bound processes
  - Assume the CPU-bound process gets and holds the CPU
  - Meanwhile, all other processes finish their I/O and move into the ready queue to wait for the CPU
    - Leaves the I/O queues idle
  - CPU-bound process finishes its CPU burst and moves to an I/O device
  - All the I/O-bound processes (short CPU bursts) execute quickly and move back to the I/O queues
  - CPU is idle
  - The above repeats!
  - Are the I/O devices and CPU utilized as much as they could be?
- Not used in modern operating systems

LIFO Scheduling

- Last-In First-Out (LIFO)
  - New processes are placed at head of ready queue
  - Improves response time for newly created processes
- Problem:
  - May lead to starvation – early processes may never get CPU

Shortest-Job-First (SJF) Scheduling

- Estimated CPU burst time is associated with each process.
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request

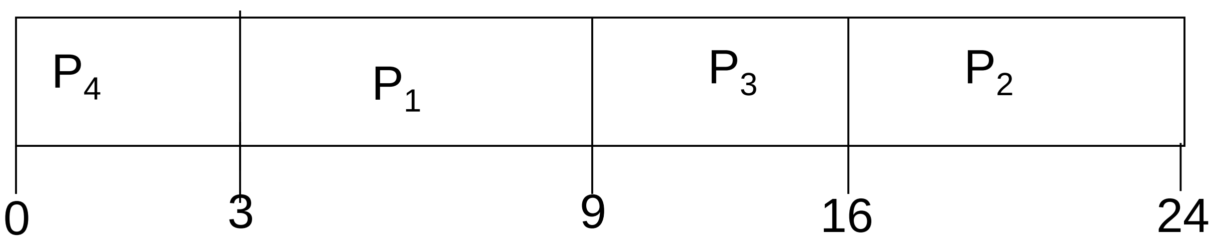
Shortest-Job-First (SJF) Scheduling

- The shortest process will jump to the head of the queue.
- Running time for batch jobs that run frequently are easier to estimate.
- Interactive processes will require statistics to determine average of each burst for each process.

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Shortest Job First Prediction

- There is no way to know the length of the next CPU burst.
- We may be able to predict it.
- We expect that the next CPU burst will be similar in length to the previous one.
- By computing the length of the next CPU burst we can determine the process with the shortest predicted CPU burst.

Shortest Job First Prediction (cont.)

- Approximate next CPU-burst duration
  - from the durations of the previous bursts
    - The past can be a good predictor of the future
- No need to remember entire past history
- Use exponential average:

  t_n duration of the n^{th} CPU burst

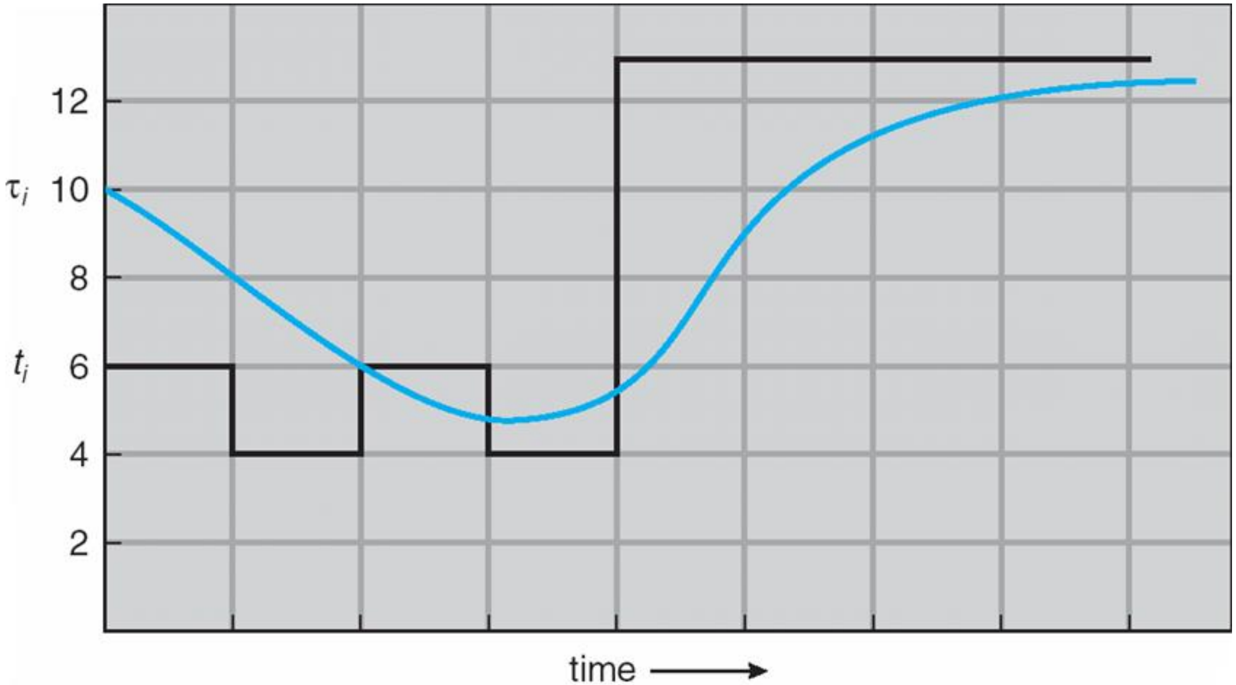
  τ_{n+1} predicted duration of the $(n+1)^{st}$ CPU burst

  $$\tau_{n+1} = \alpha\ t_n + (1-\ \alpha)\ \tau_n$$

  where $0 \le \alpha \le 1$

  α determines the weight placed on past behavior

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\begin{aligned}\tau_{n+1} = \alpha\, t_n &+ (1 - \alpha)\alpha\, t_{n-1} + \ldots \\ &+ (1 - \alpha)^j \alpha\, t_{n-j} + \ldots \\ &+ (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Exponential Smoothing Coefficients

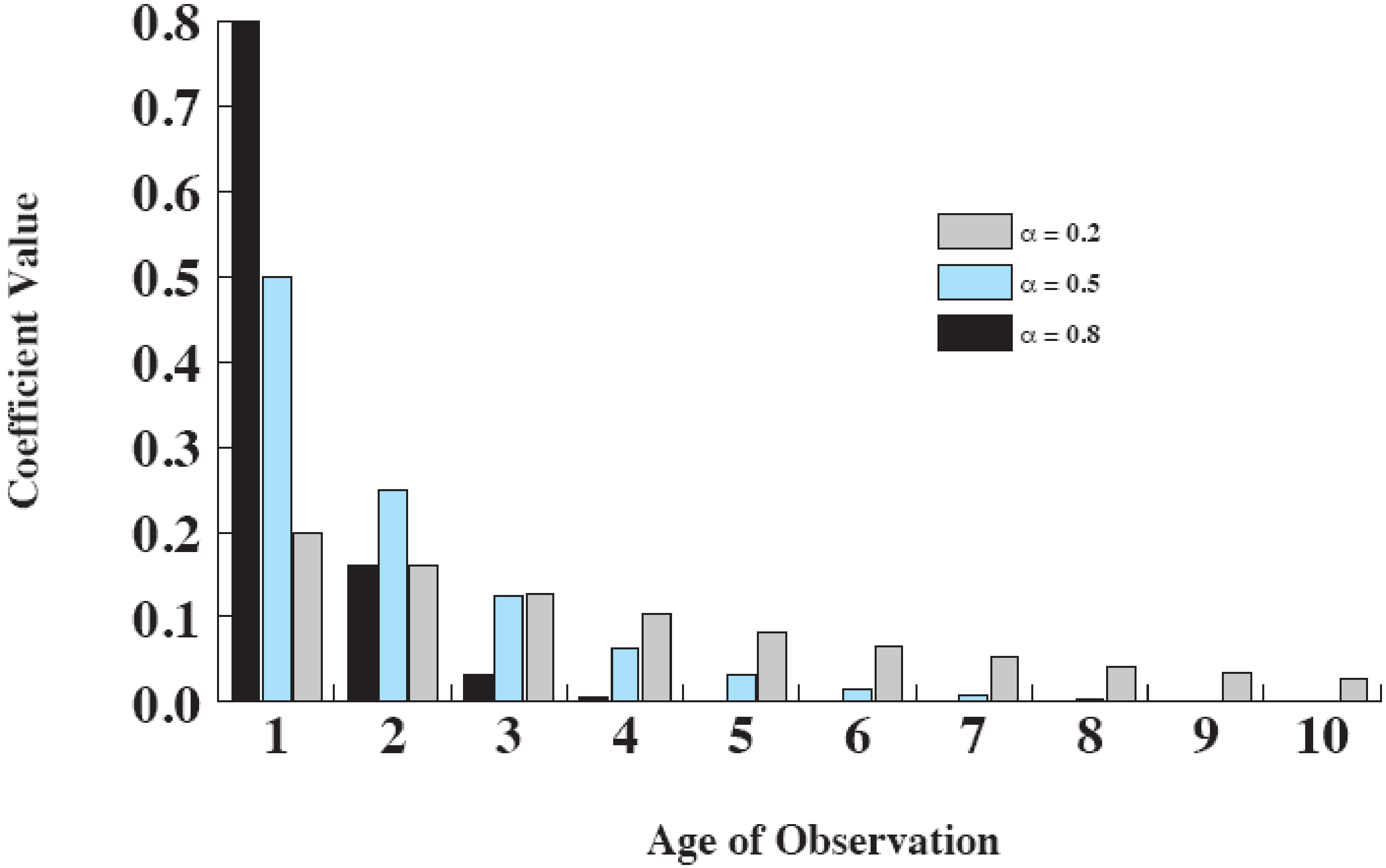


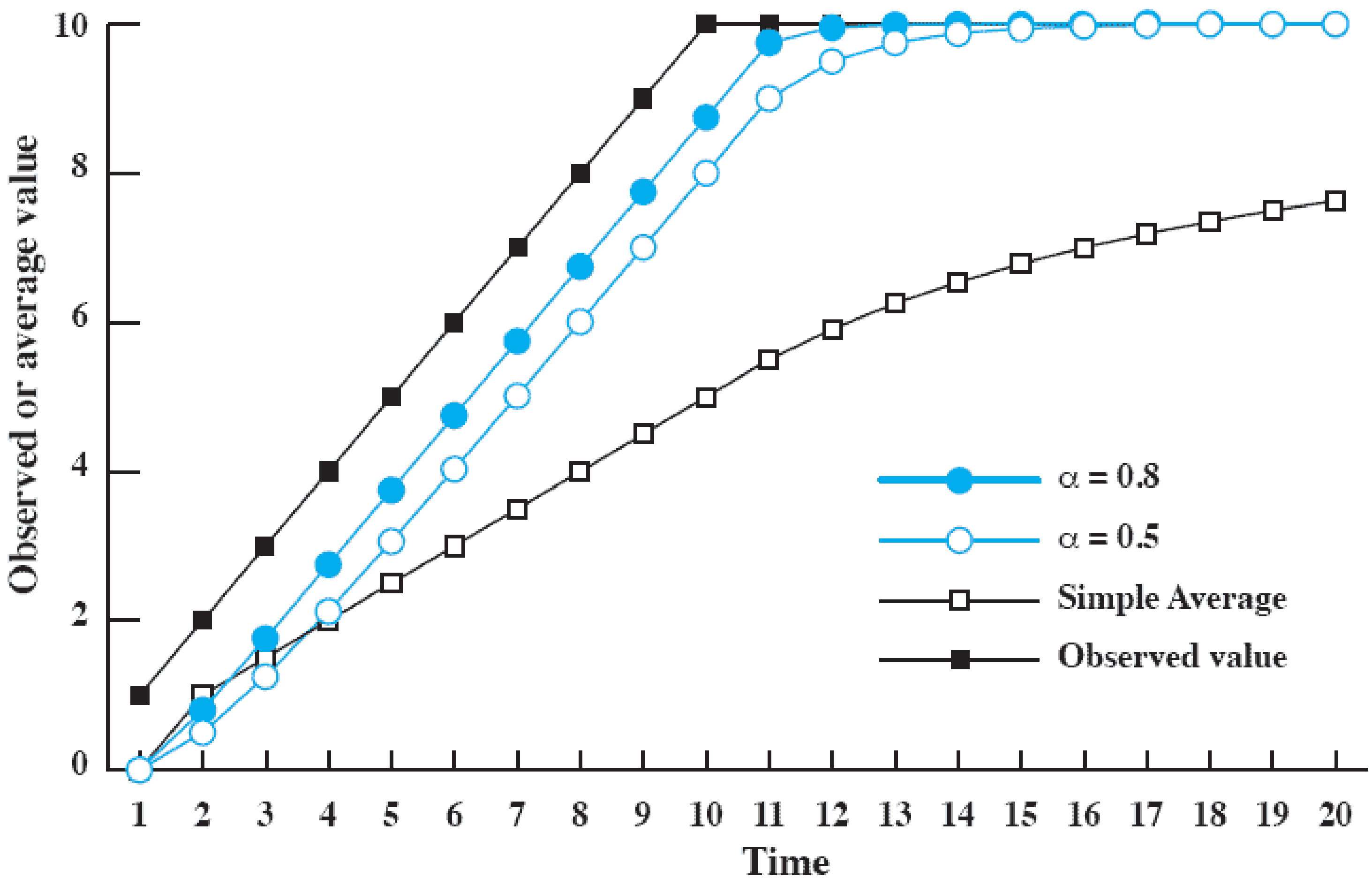
Figure 9.8 Exponential Smoothing Coefficients

Exponential Coefficients

- The larger the value of α, the greater the weight given to more recent observations.
  - α = 0.8, almost all the weight is given to the four most recent observations.
    - Advantage: Average will quickly reflect a rapid change in the observed quantity.
    - Disadvantage: If there is a brief increase in the value of the observed quantity followed by a value around the average, the use of a large value of α will result in sudden change in the average.
  - α = 0.2, the averaging is effectively spread out over the eight most recent observations.

Use of Exponential Averaging

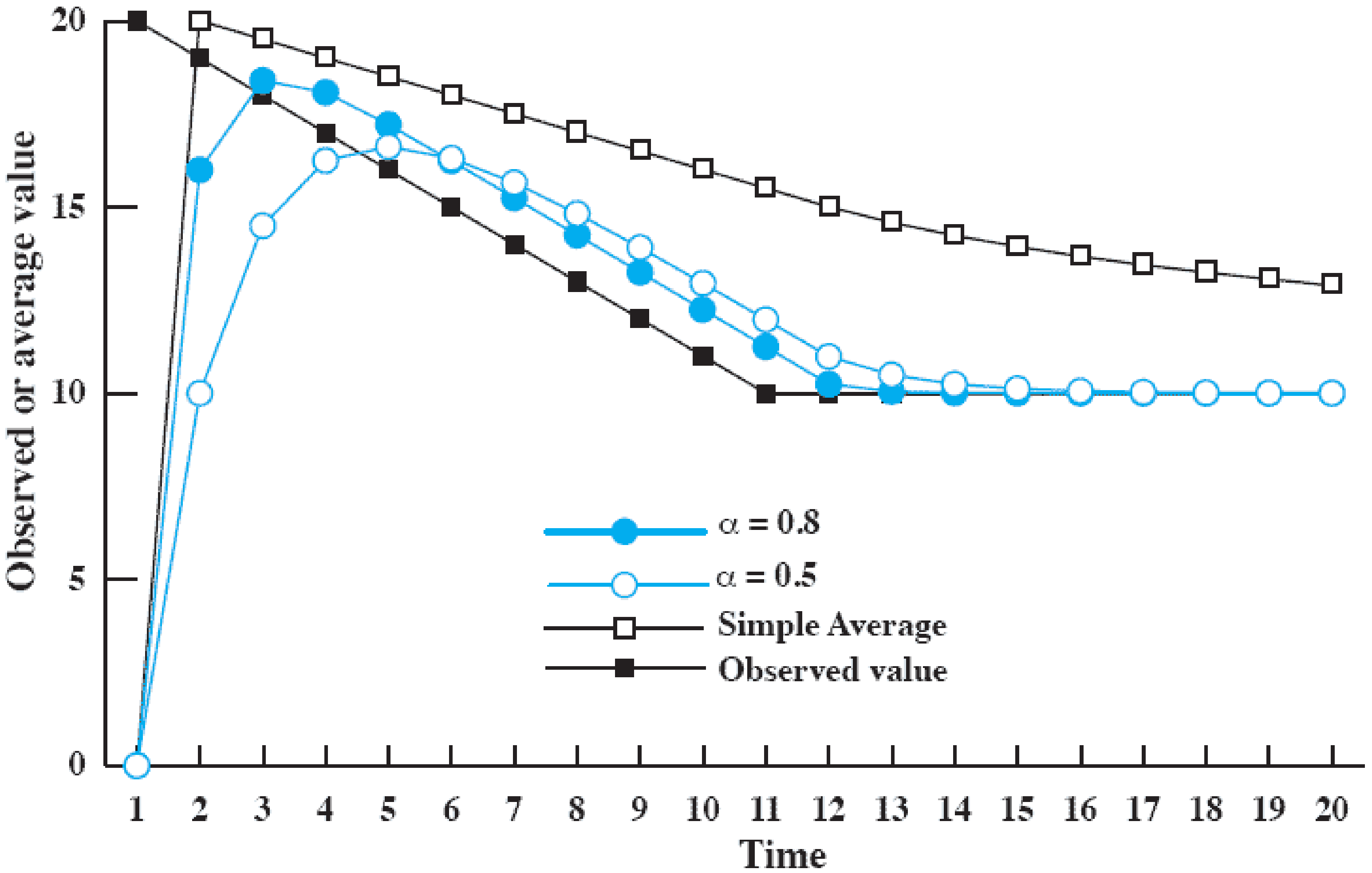
Observed value begins at 1 and grows gradually to a value of 10 and stays there.



(a) Increasing function

Use of Exponential Averaging

The observed value begins at 20, declines gradually to 10 and stays there.



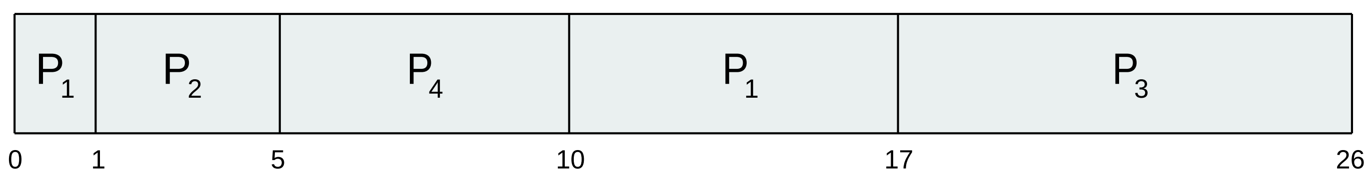
(b) Decreasing function

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis.

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive SJF Gantt Chart*



- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer $\equiv$ highest priority)
  - Preemptive
  - Non-preemptive

Priority Scheduling

- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem: Starvation
  - Low priority processes may never execute

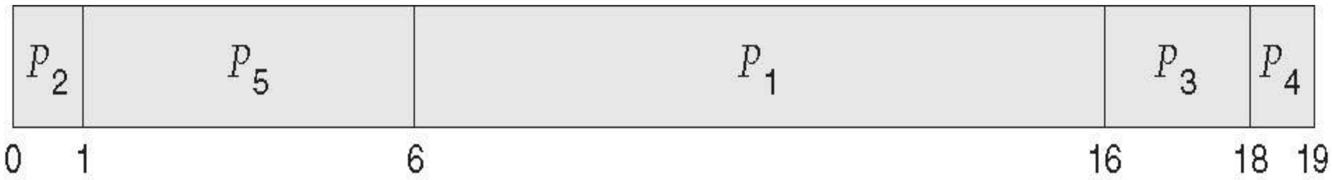
unfinished low-priority processes. (Rumor has it that when they shut down the IBM 7094 at MIT in 1973, they found a low-priority process that had been submitted in 1967 and had not yet been run.)

- Solution :*Aging*
  - As time progresses increase the priority of the process

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

Round Robin (RR)

- 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 (RR)

- Each process gets a small unit of CPU time (time quantum), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.

Round Robin (RR)

- Timer interrupts every quantum to schedule next process
- Performance
  - q is too large $\Rightarrow$ FIFO
  - q is too small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Process Scheduling Example

- Example set of processes, consider each a batch job

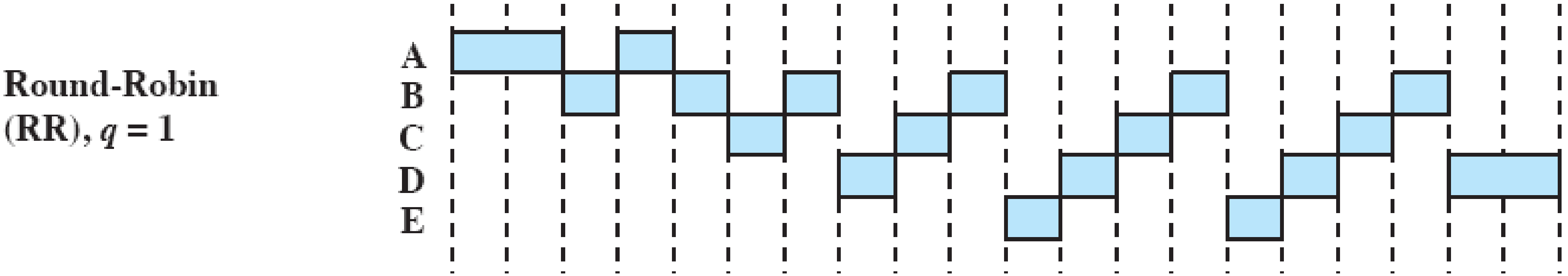
Table 9.4 Process Scheduling Example

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

  – Service time represents total execution time

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.



Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

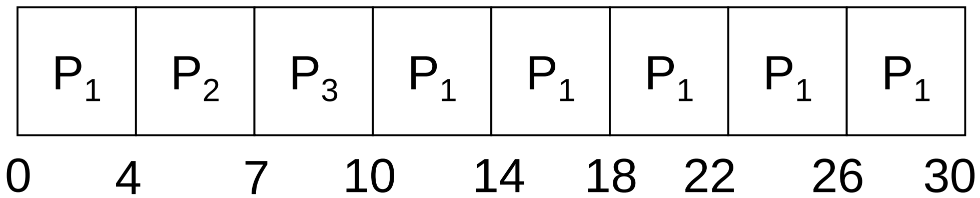
0 4 7 10 14 18 22 26 30

- Typically, higher average turnaround than SJF, but better response
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10μsec

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

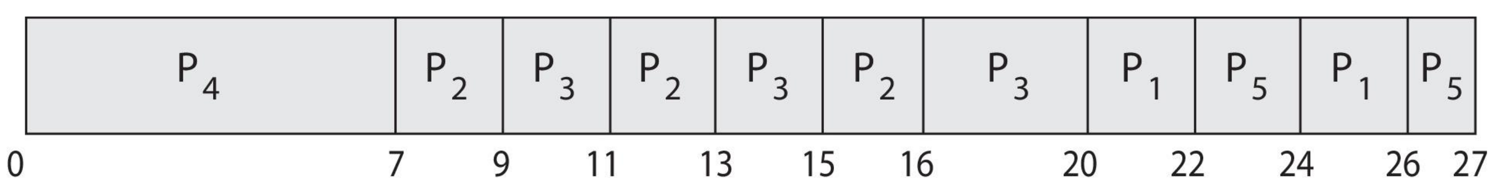


- Average waiting time = [(10-4)+(4)+(7)]/3 = 17/3 = 5.66 msec

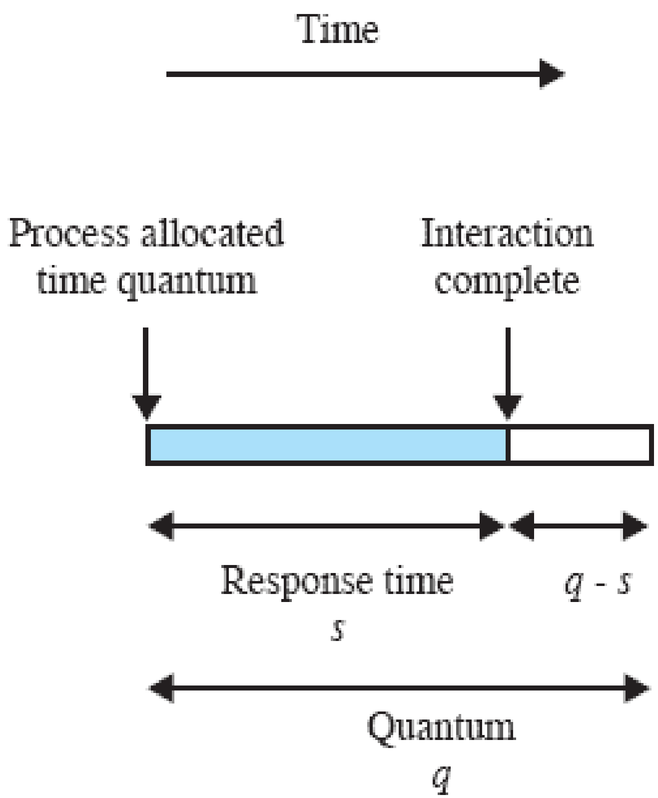
Priority Scheduling w/ Round-Robin

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- Run the process with the highest priority. Processes with the same priority run round-robin
- Gantt Chart with time quantum = 2

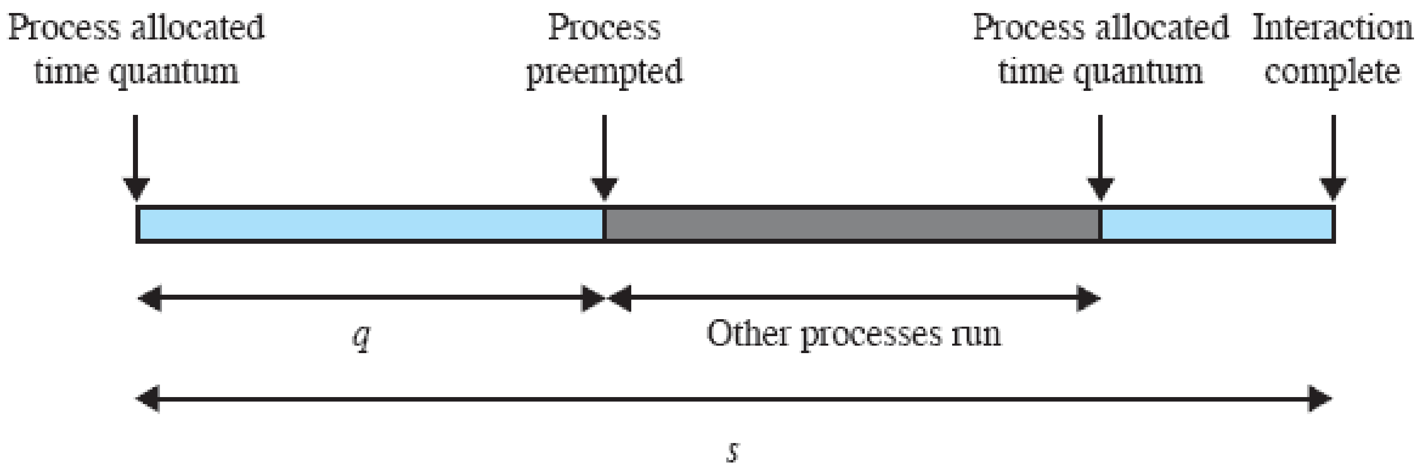


Effect of Size of Preemption Time Quantum



(a) Time quantum greater than typical interaction

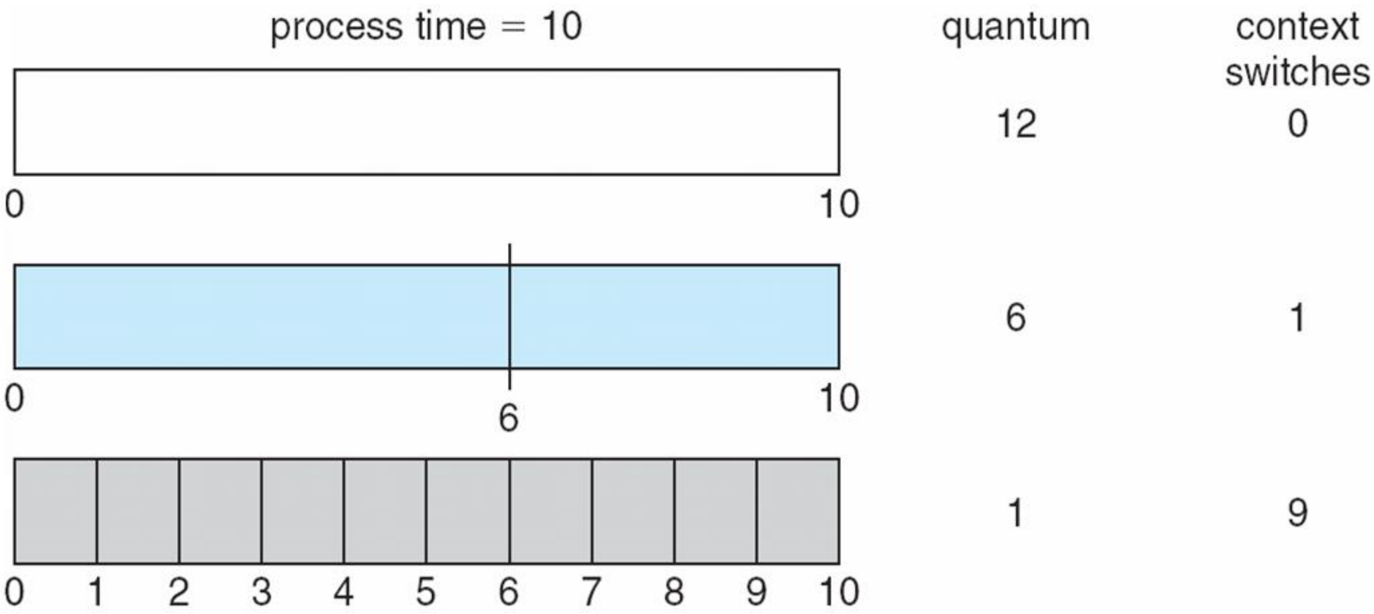
Effect of Size of Preemption Time Quantum



(b) Time quantum less than typical interaction

Figure 9.6 Effect of Size of Preemption Time Quantum

Time Quantum and Context Switch Time



'Virtual Round Robin'

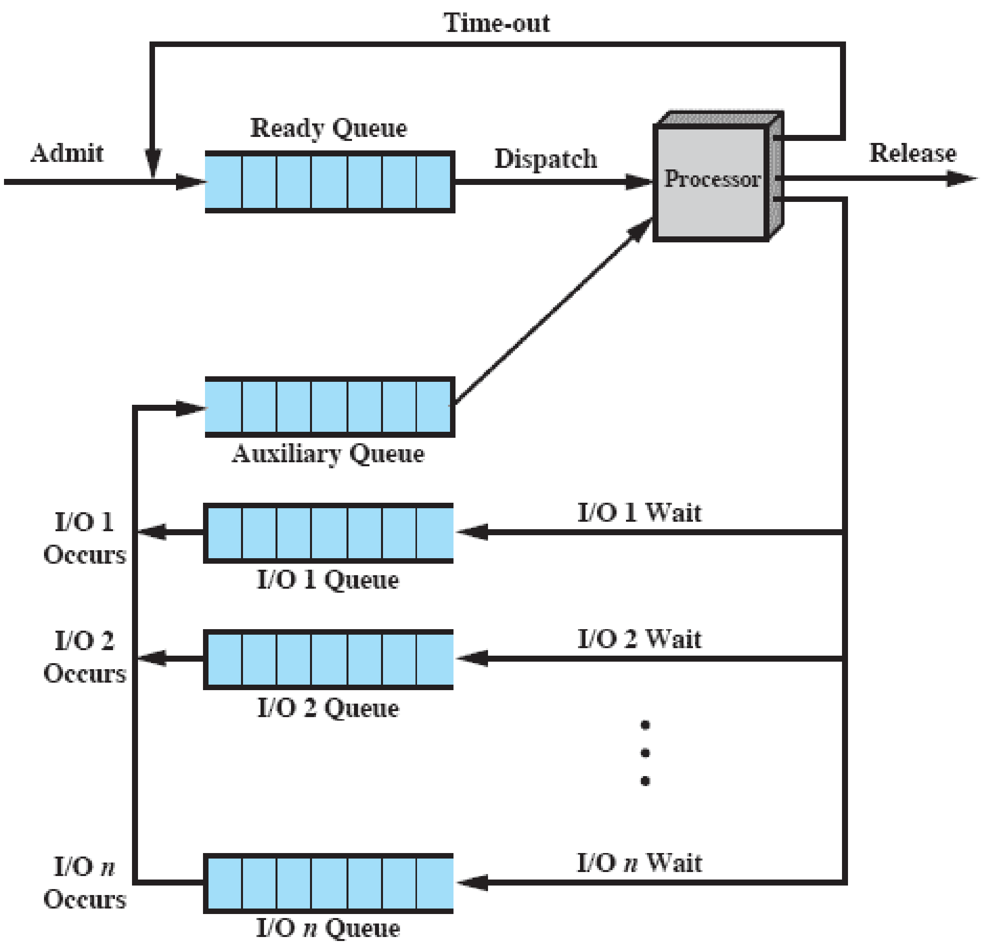
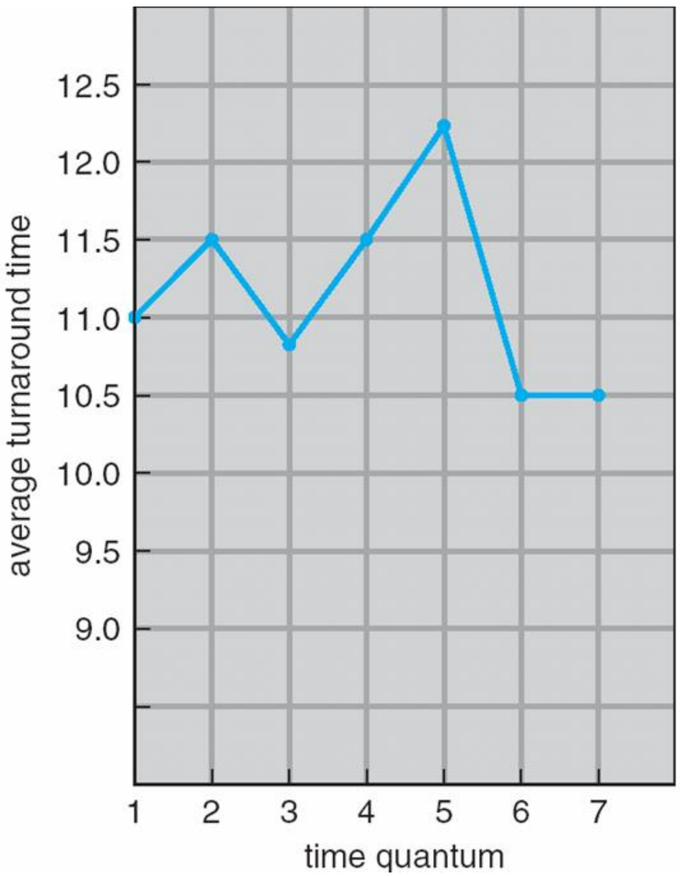


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

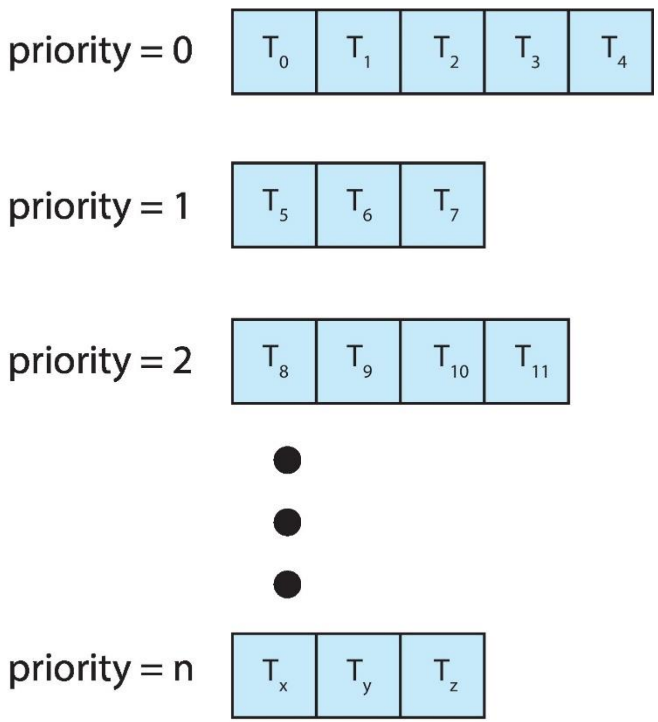
80% of CPU bursts
should be shorter than q

Multilevel Feedback Queue Scheduling

- A process can move between queues
- Separate processes according to the characteristics of the CPU bursts
  - If a process uses too much CPU time, it will be moved to a lower-priority queue
  - Leave I/O bound and interactive processes in the higher-priority queues
  - In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue

Multilevel Queue

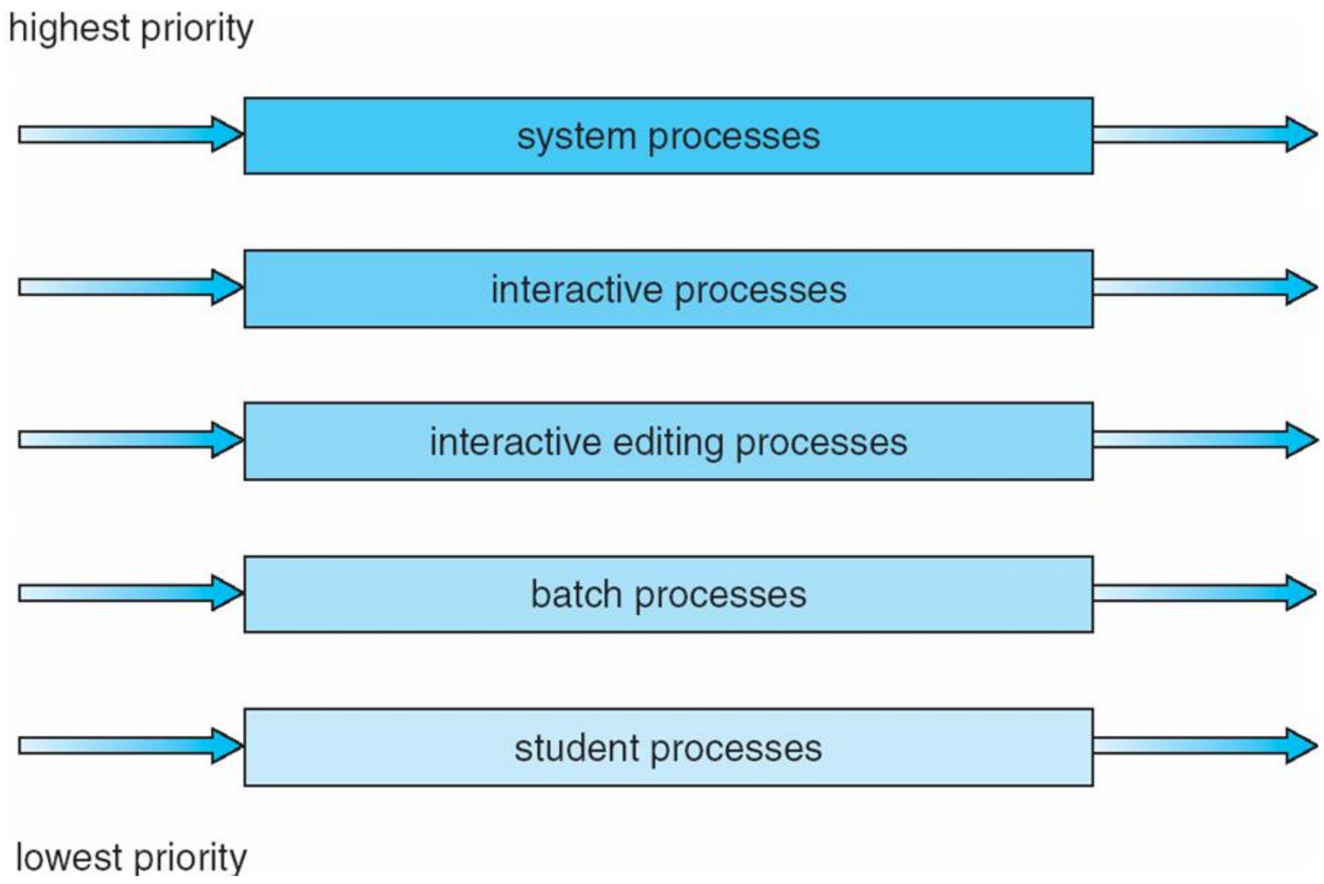
- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!



Multilevel Queue

- Scheduling must be done between the queues
  - Fixed priority scheduling; (i.e., serve all from foreground then from background).
    - Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes e.g.,
    - 80% to foreground in RR
    - 20% to background in FCFS

Multilevel Queue Scheduling



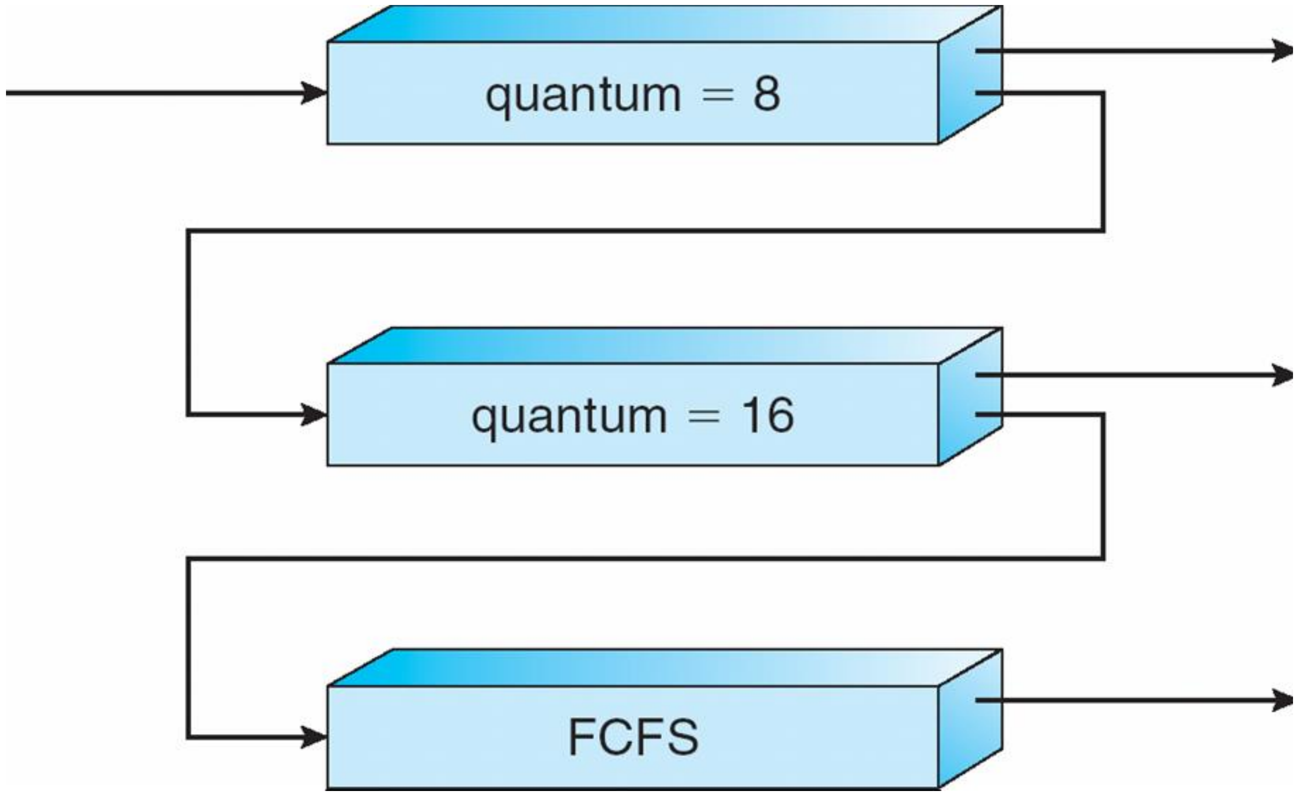
Multilevel Feedback Queue Scheduling

- Example: Assume three queues: Q0, Q1, Q2
  - The scheduler first executes all processes in Q0; it then proceeds to queue Q1 followed by queue Q2
  - A process entering the ready state is put in Q0
  - A process in Q0 is given a time quantum of 8 milliseconds

Multilevel Feedback Queue Scheduling

- Example: Assume three queues: Q0, Q1, Q2 (cont)
  - If it does not finish within this time it is moved to the tail of Q1.
  - If Q0 is empty the process at the head of Q1 is given a quantum of 16 milliseconds
  - If it does not complete, it is preempted and is put into Q2
  - Gives highest priority to any process with a CPU burst of 8 milliseconds

Multilevel Feedback Queues



Multilevel Feedback Queue

- A process can move between the various queues
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service
- Aging can be implemented using multilevel feedback

Process Scheduling Example

- Example set of processes, consider each a batch job

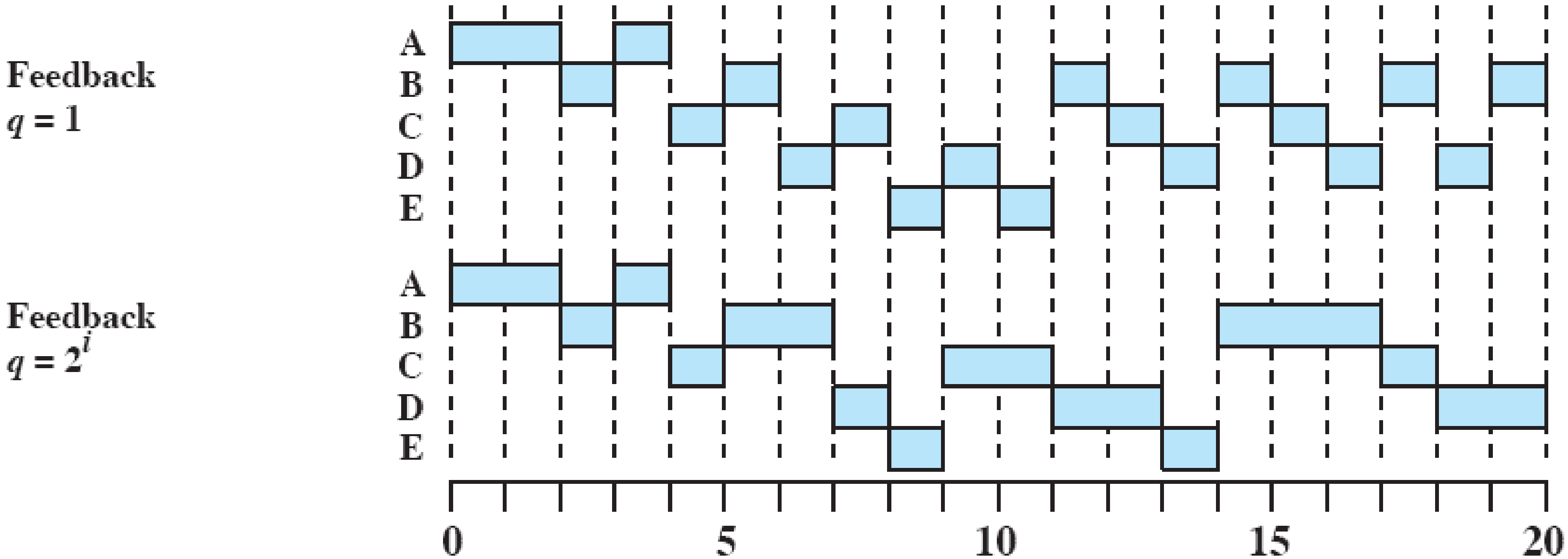
Table 9.4 Process Scheduling Example

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

  – Service time represents total execution time

Feedback Performance

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



Lottery Scheduling

- Scheduler gives each thread some lottery tickets.
- To select the next process to run...
  - The scheduler randomly selects a lottery number
  - The winning process gets to run
- Example

  Thread A gets 50 tickets → 50% of CPU

  Thread B gets 15 tickets → 15% of CPU

  Thread C gets 35 tickets → 35% of CPU

  There are 100 tickets outstanding.

Topics

- Types of Processor Scheduling
- Scheduling Algorithms
- Examples

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
- Priority is based on process type and execution history.
  - Increases over time if process blocks before end of quantum
  - Decreases over time if process uses entire quantum

Scheduling Formula

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

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

where

$CPU_j(i)$ = 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

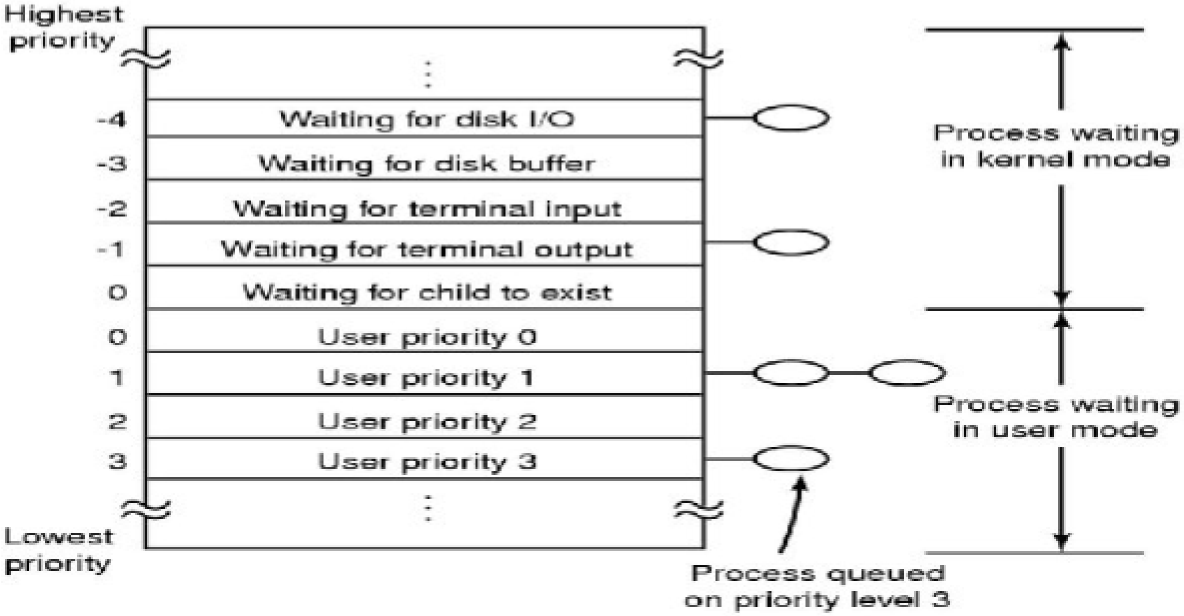
$Base_j$ = base priority of process j

$nice_j$ = user-controllable adjustment factor

Bands

- Priorities are recomputed once per second
- Base priority divides all processes into fixed bands of priority levels
  - Swapper (highest)
  - Block I/O device control
  - File manipulation
  - Character I/O device control
  - User processes (lowest)

UNIX Scheduler



Example of Traditional UNIX Process Scheduling

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

Colored rectangle represents executing process

Figure 9.17 Example of Traditional UNIX Process Scheduling

Examples

Operating System	Preemption	Algorithm
Amiga OS	Yes	Prioritized round-robin scheduling
classic Mac OS pre-9	None	Cooperative scheduler
FreeBSD	Yes	Multilevel feedback queue
Linux kernel 2.6.0–2.6.23	Yes	O(1) scheduler
Linux kernel after 2.6.23	Yes	Completely Fair Scheduler
Linux kernel before 2.6.0	Yes	Multilevel feedback queue
Mac OS 9	Some	Preemptive scheduler for MP tasks, and cooperative for processes and threads
macOS	Yes	Multilevel feedback queue
NetBSD	Yes	Multilevel feedback queue
Solaris	Yes	Multilevel feedback queue
Windows 3.1x	None	Cooperative scheduler
Windows 95, 98, Me	Half	Preemptive scheduler for 32-bit processes, and cooperative for 16-bit processes
Windows NT (including 2000, XP, Vista, 7, and Server)	Yes	Multilevel feedback queue

Summary

- Reviewed several scheduling algorithms