Scheduling

Reading:

Silberschatz

chapter 6

Additional Reading:

Stallings

chapter 9

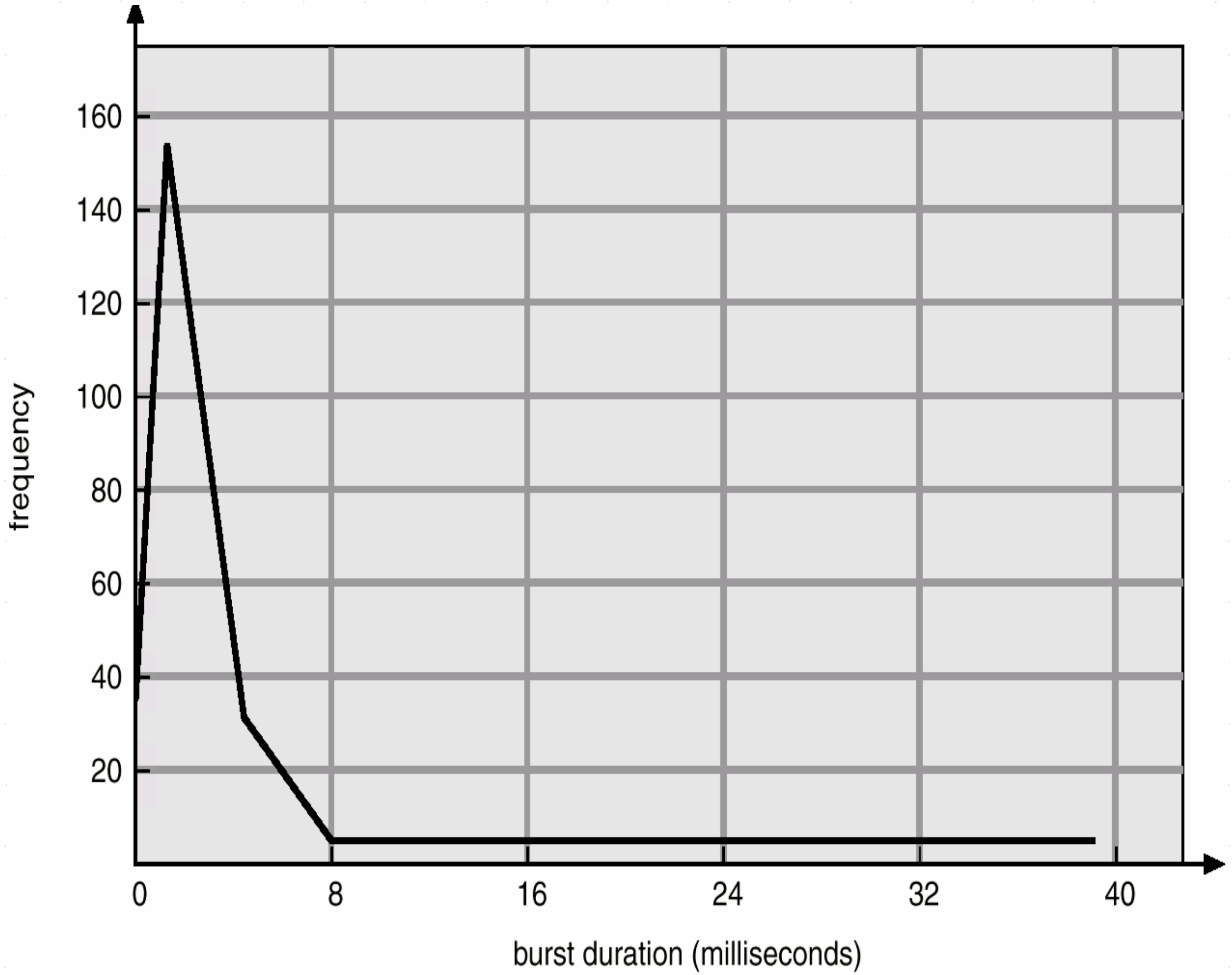
Outline

- Introduction
- Types of Scheduling
- Scheduling Criteria
- FCFS Scheduling
- Shortest-Job-First Scheduling
- Priority Scheduling
- Round Robin Scheduling
- Multilevel Queue Scheduling
- Multiprocessor Scheduling
  - Load Balancing
  - Symmetric Multithreading
- Algorithm Evaluation
- Real Time Scheduling
- Scheduling Examples
  - Windows XP, 2000
  - Linux

Introduction

- Basic Points
  - *Process Scheduling, Thread Scheduling*
  - *Max CPU Utilization $\rightarrow$ Multiprogramming*
  - *CPU Burst $\leftrightarrow$ I/O Burst*
  - *CPU Burst Distribution*

Histogram – CPU Burst Duration



Scheduling Types

- CPU Scheduling Decisions
  - Running → Waiting state
    - e.g. I/O Request, wait by Parent
  - Running → Ready state
    - e.g. Interrupt
  - Waiting → Ready state
    - e.g. Completion of I/O
  - Process Termination
- Nonpreemptive Scheduling
- Preemptive Scheduling
  - Associated Cost
  - Design of OS Kernel
    - Process → Kernel, wait for sys call or I/O completion *before* context switch

Scheduling Criteria

- CPU Utilization – How busy is the CPU?
- Throughput – Number of processes that are completed per unit time
- Turnaround Time – How long to execute a process? Submission ↔ Completion
- Waiting Time – Sum of periods spent in ready queue
- Response Time – Process Request → First response

Optimization Criteria

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

- Conflicting goals! Requires careful balance
- Average, Min/Max, Variance

FCFS Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Arrivals in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule:

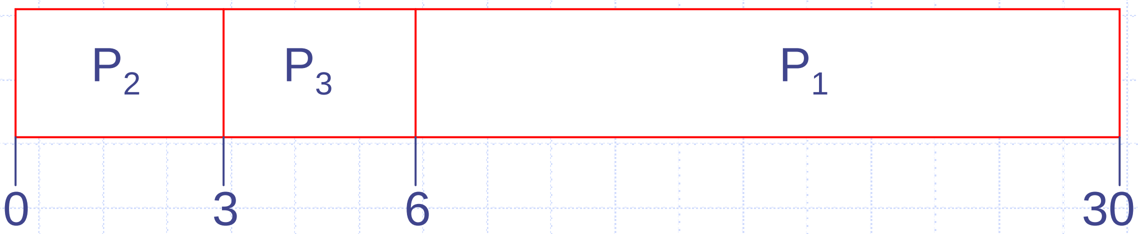
- Waiting time → P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time → (0 + 24 + 27)/3 = 17

FCFS Scheduling

Say, if the processes arrive in the order

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule:



- Waiting time → $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time → (6 + 0 + 3)/3 = 3, *Previously 17* ↑
- *Convoy effect* → Short process behind long process
- Nonpreemptive → Problem for time sharing systems

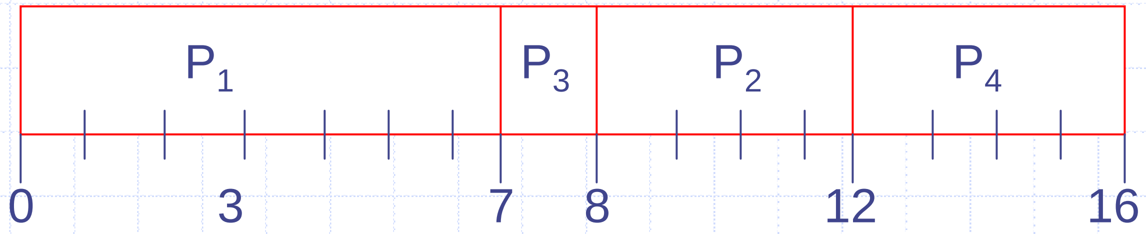
SJF Scheduling

- CPU assigned to process with smallest next CPU burst, Tie $\rightarrow$ FCFS
- *Shortest-next-CPU-burst algorithm*
- Major difficulty
  - Estimating the processing time of each job, Predicting the Next!
  - Long running jobs may starve, steady supply of short jobs to CPU
- SJF is optimal – minimum average waiting time

SJF Scheduling

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

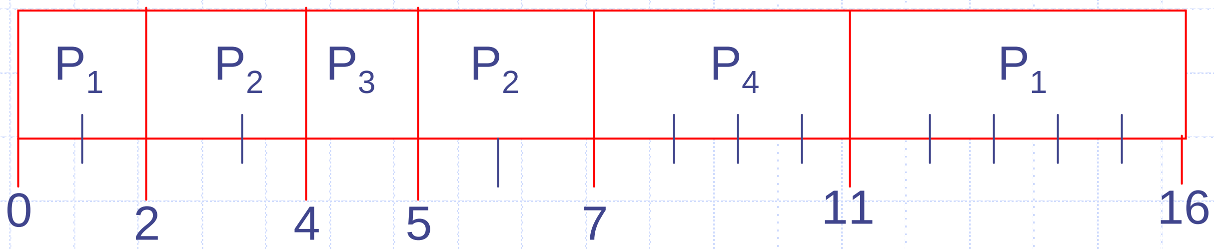


- Average waiting time → (0 + 6 + 3 + 7)/4 = 4

SJF Scheduling

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Priority Scheduling

- A priority number (integer) associated with each process
  - SJF – A Priority scheduling
  - Equal Priority - FCFS
- CPU $\rightarrow$ Process with the highest priority, High $\leftrightarrow$ Low
  - Preemptive
  - Nonpreemptive
- Defining Priorities
  - Internally, Measurable Quantities
    - Memory required, time limits, # open files, ratio of avg I/O to CPU burst, *etc*.
  - Externally, Outside OS
    - Importance of Process, type/amount of funds, *etc*.
- Starvation
  - Low priority processes may never execute
- Solution?
  - Aging

Round Robin (RR)

- Each process gets a small unit of CPU time
  - Time Quantum (*time-slice*)
    - usually 10-100 milliseconds
  - Time elapsed → Preempted
    - If not completed → end of the ready queue
- RR reduces penalty for short jobs in FCFS
- Critical Issue → Length of quantum, *q*
  - *q* large → FIFO or FCFS
  - *q* small → Context switch overhead

Round Robin (RR)

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

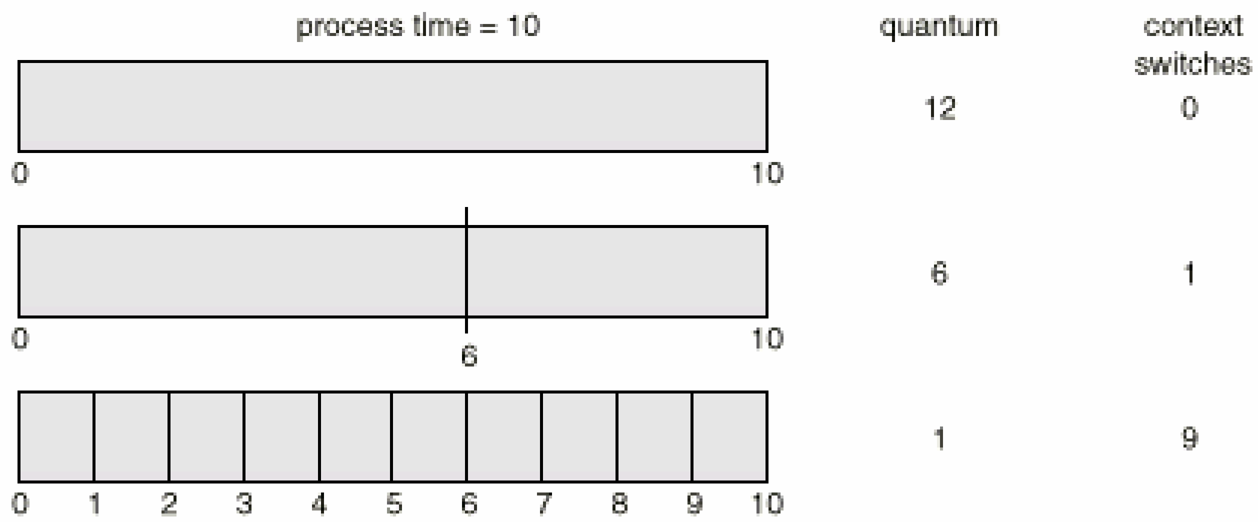
- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

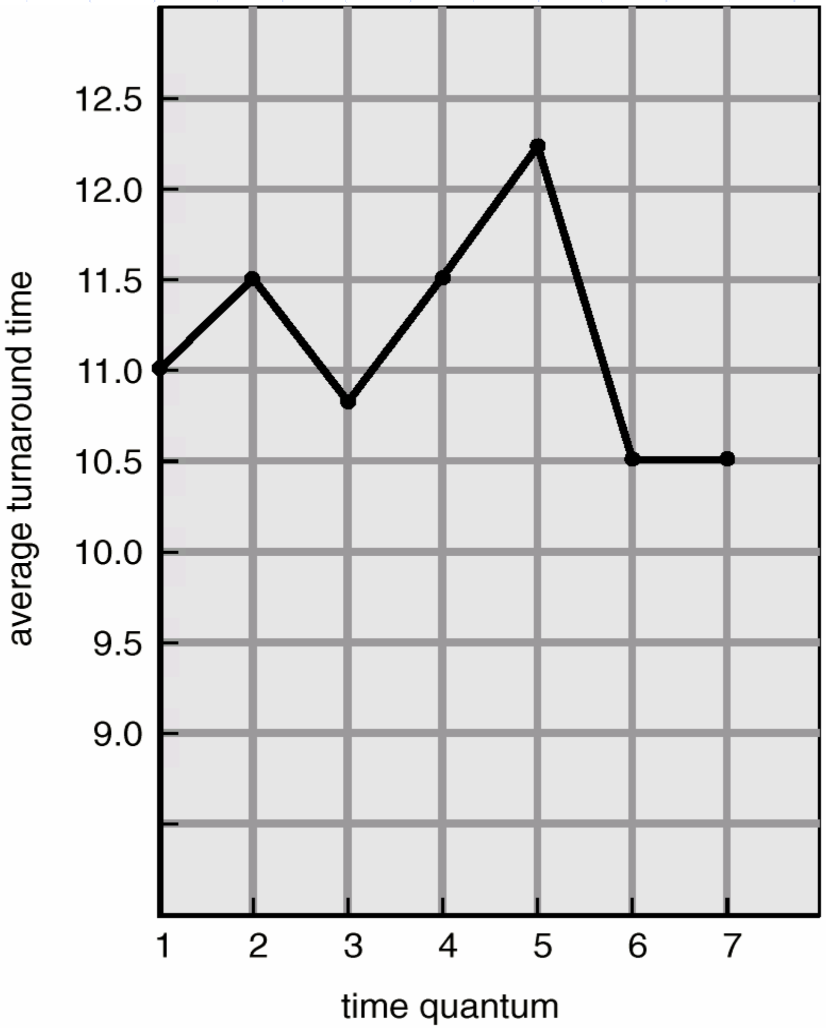
0 20 37 57 77 97 117 121 134 154 162

- Typically, higher average turnaround than SJF, but better *response*.

Round Robin (RR)



Round Robin (RR)

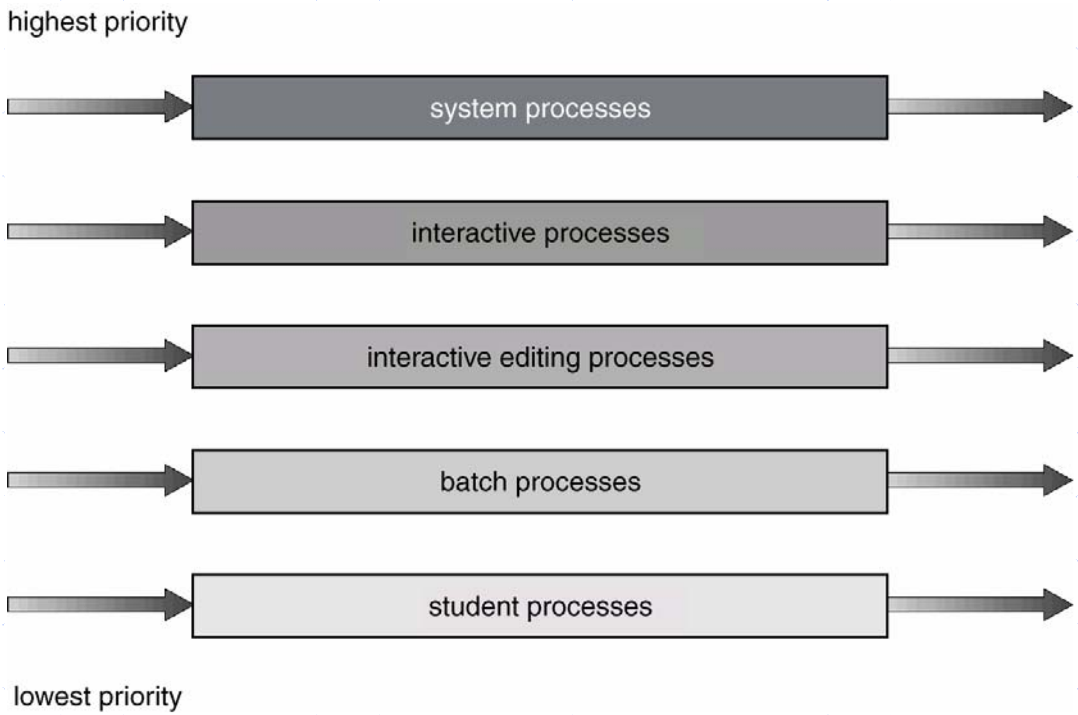


process	time
P_1	6
P_2	3
P_3	1
P_4	7

Multilevel Queue

- Ready queue $\rightarrow$ separate queues
  - foreground (interactive)
  - background (batch)
- Each queue $\rightarrow$ own scheduling algorithm, *e.g.*
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues
  - Fixed priority scheduling; (i.e., serve all from foreground then from background), *Starvation*
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

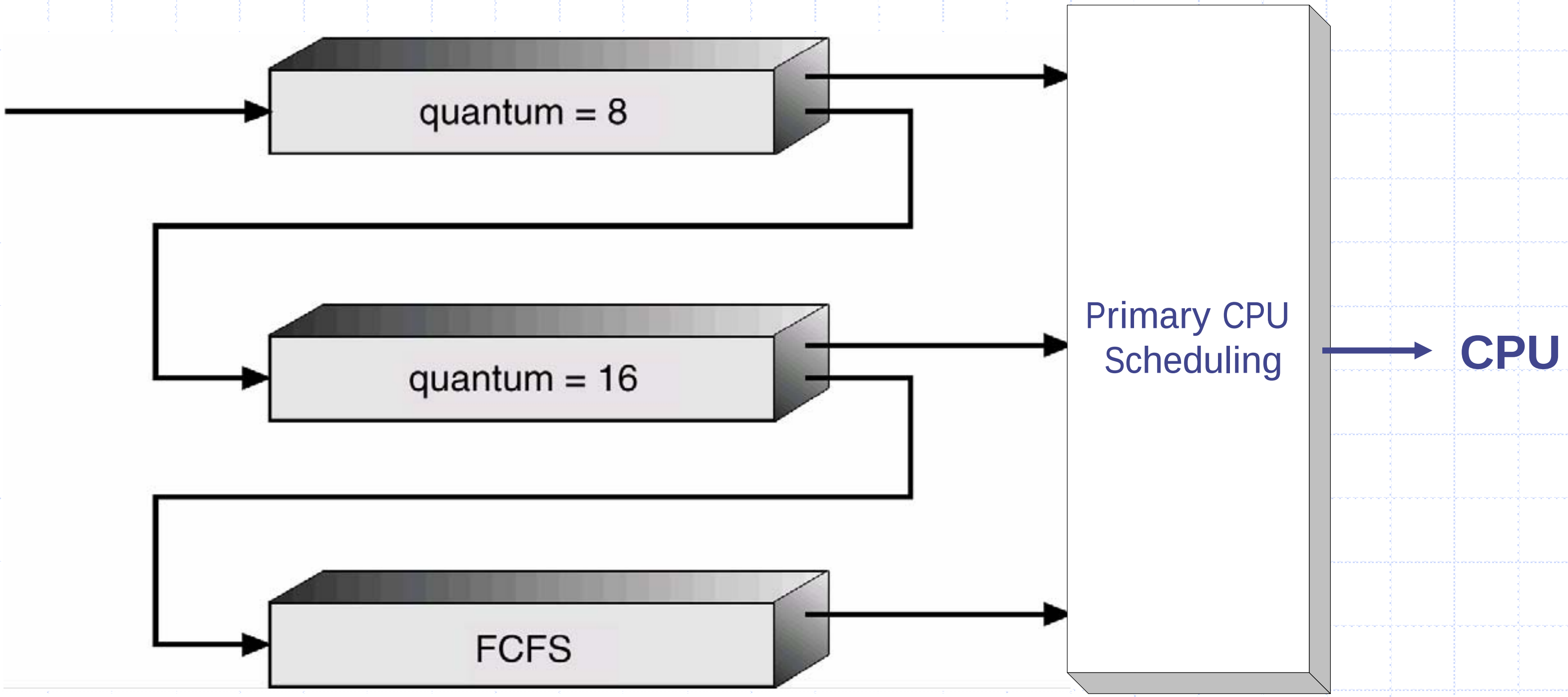
Multilevel Queue Scheduling



Multilevel Feedback Queue

- Separate processes $\rightarrow$ CPU burst characteristics
- Process moves *up* $\leftrightarrow$ *down* in queues
  - Too much time $\downarrow$
  - Aging $\uparrow$
- Key points
  - 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

Example of Multilevel Feedback Queue



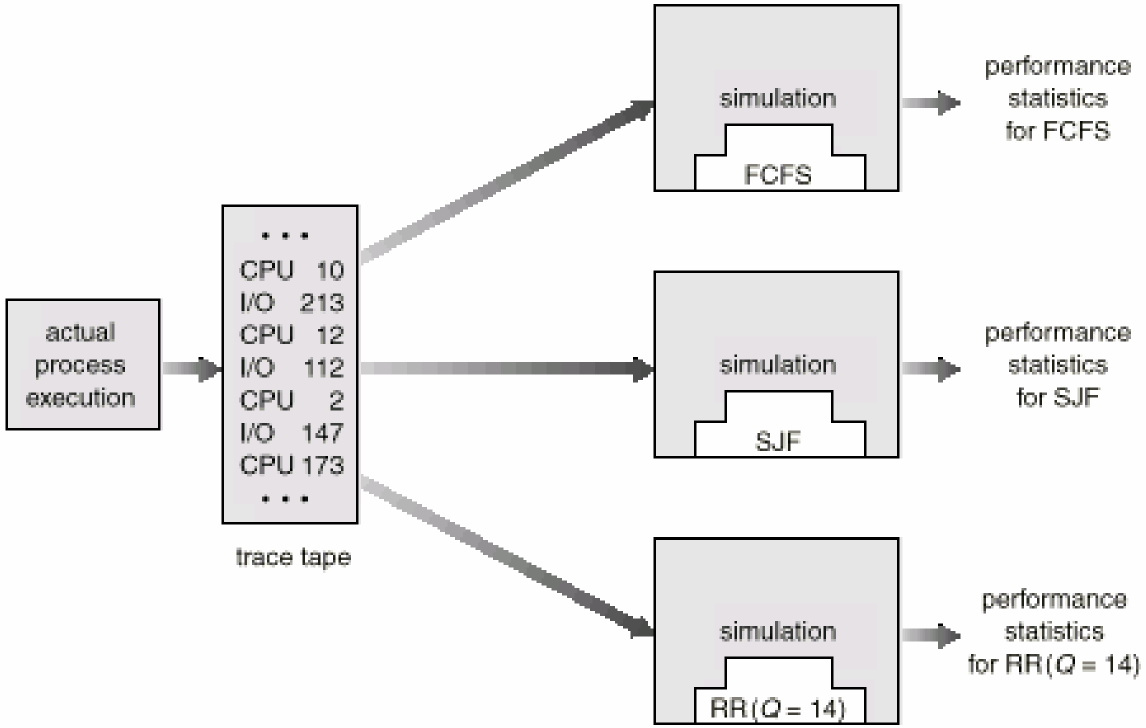
Multiple-Processor Scheduling

- Multiple CPUs → High scheduling complexity
- Homogeneous Processors
  - Asymmetric Multiprocessing
    - No data sharing, System data structures → one processor
  - Symmetric Multiprocessing
    - Self Scheduling, Ready queue
- Processor Affinity
  - Soft Vs Hard affinity
- Load Balancing
  - Push Migration
  - Pull Migration

Algorithm Evaluation

- Deterministic modeling
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Queueing models
  - Queue of network servers
  - Little's formula, $l = \lambda \times w$
    - λ - Avg arrival rate, w - Avg waiting time, l - Avg queue length
- Simulation
  - Model, clock
  - Simulation $\rightarrow$ modifies system with clock $\uparrow$
  - Distribution driven simulation
  - Only # instances of an event, order?

Evaluation of CPU Schedulers by Simulation



Real-Time Scheduling

- *Hard real-time* systems
  - Complete a *critical task* within a guaranteed time
  - Admit or Reject
  - Impossible with *SS*, *VM*
  - Resource Reservation
- *Soft real-time* computing
  - Critical processes receive priority over less fortunate ones
  - General-purpose systems $\rightarrow$ Multimedia, Graphics
  - *Priority Inversion*
  - *Priority-inheritance protocol*

Example: Windows XP, 2000

Scheduling

- Priority-based, preemptive scheduling
- Thread runs → preempted by higher priority thread, terminates, Qu
- Does <u>not</u> guarantee execution of a real-time thread within time-limit

Thread Priorities

- 32 level priority scheme
- *Real time class* → 16-32
- *Variable class* → 1-15
- Memory Management → Thread at 0 priority
-
- Six Classes (Win32 API) – 1 + 5
- Within each 6 classes – 7 relative priorities
- Currently selected foreground process → Scheduling Quantum ↑ 3

Windows XP, 2000 Priorities

Priority Classes →

Relative Priority ↓

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4 ← Base Priority
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Example: Linux

- Scheduling
  - Increased support for SMP, Scaling with # tasks
  - *Processor affinity*, *load balancing*
  - High priority tasks → longer quanta, vice-versa
  - Real time tasks – static priorities
  - Rest dynamic → nice values ± 5 (*interactivity*)

- Numeric Priorities
  - 0-140 level priority scheme
  - Real time → 0-99
  - *Nice* values → 100-140