OPERATING SYSTEMS

PROCESS SCHEDULING

Processor Scheduling

- Aim is to assign processes to be executed by the processor in a way that **meets system objectives**
  - response time
  - throughput
  - processor efficiency

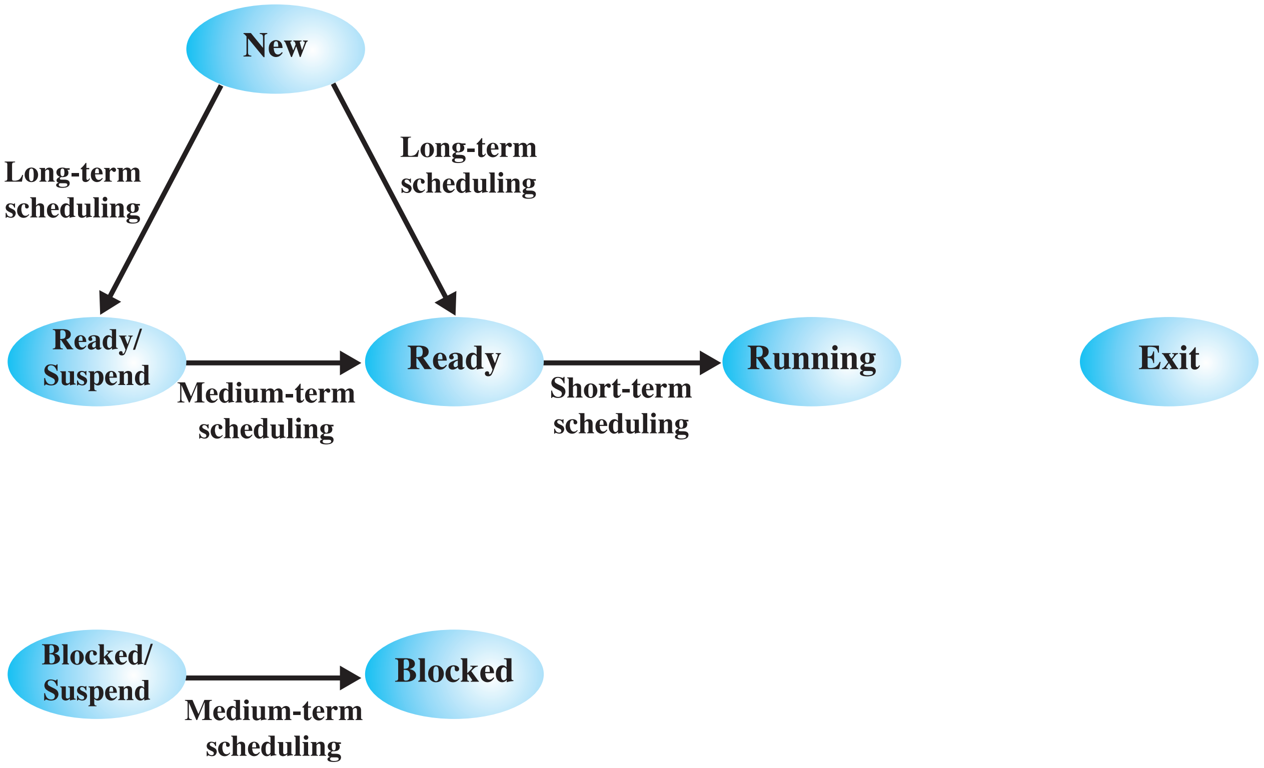
- Broken down into three separate functions

Objectives

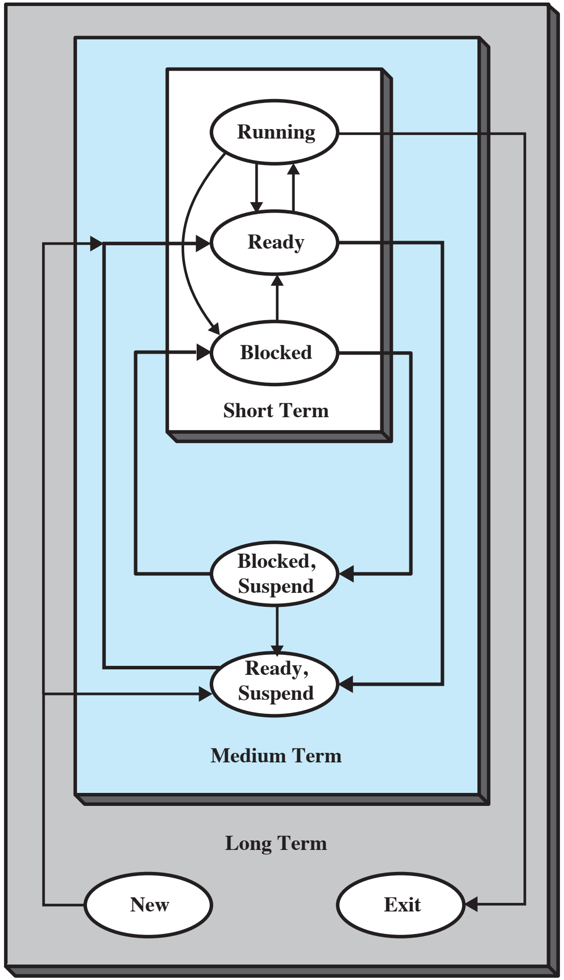
- CPU scheduling is the basis for multiprogrammed operating systems
- Various CPU-scheduling algorithms will be described
- Evaluation criteria for selecting a CPU-scheduling algorithm for a particular system will be discussed

Basic concepts

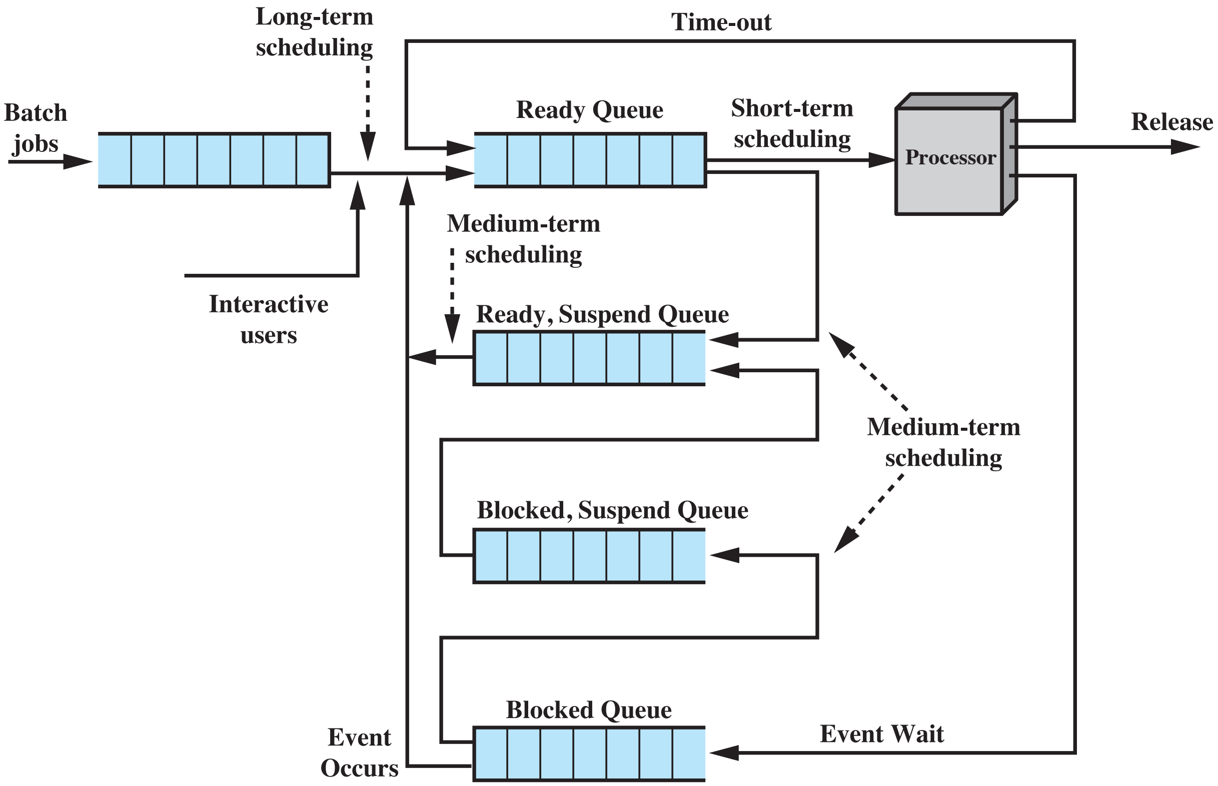
Scheduling and process state transitions



Levels of scheduling



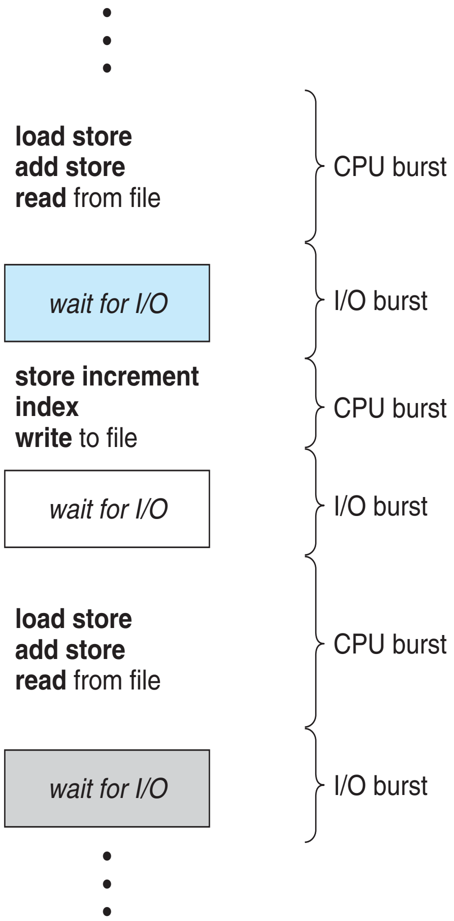
Queueing diagram for scheduling



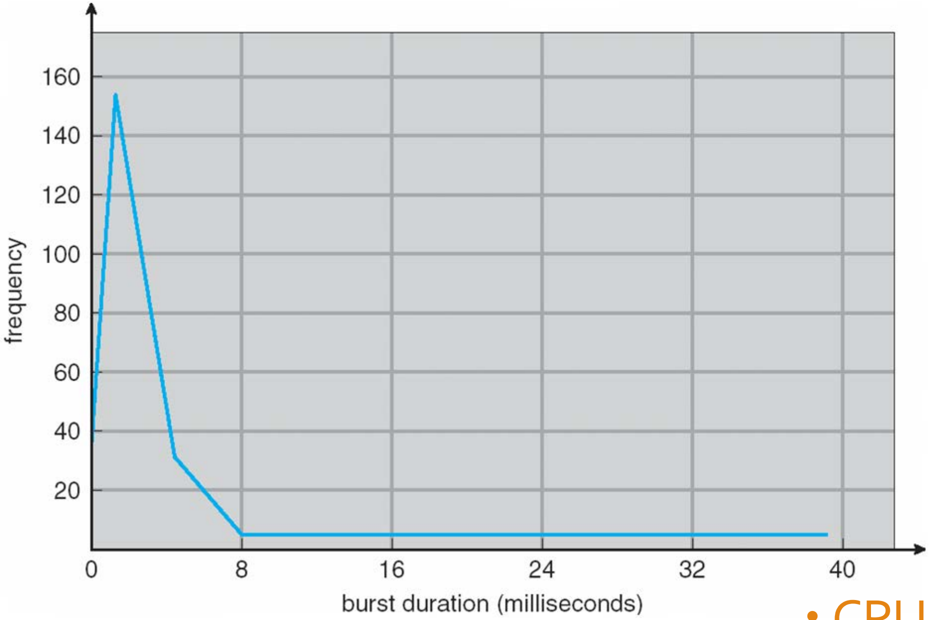
Short term scheduling

Basic Concepts

- Multiprogramming allows attaining maximum CPU utilization
- **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



Process characterization in terms of CPU burst times



- CPU bound processes
  - A small number of long CPU bursts
- I/O bound processes
  - A large number of short CPU bursts

CPU Short-term 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
- The short-term scheduler decision 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 ready
  4. Terminates
- Scheduling under 1 and 4 is non pre-emptive
- All other scheduling is preemptive
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

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

Scheduling Criteria

Short Term Scheduling Criteria

- Main objective is to allocate processor time to optimize certain aspects of system behavior
- Criteria to evaluate the scheduling policy

User-oriented criteria

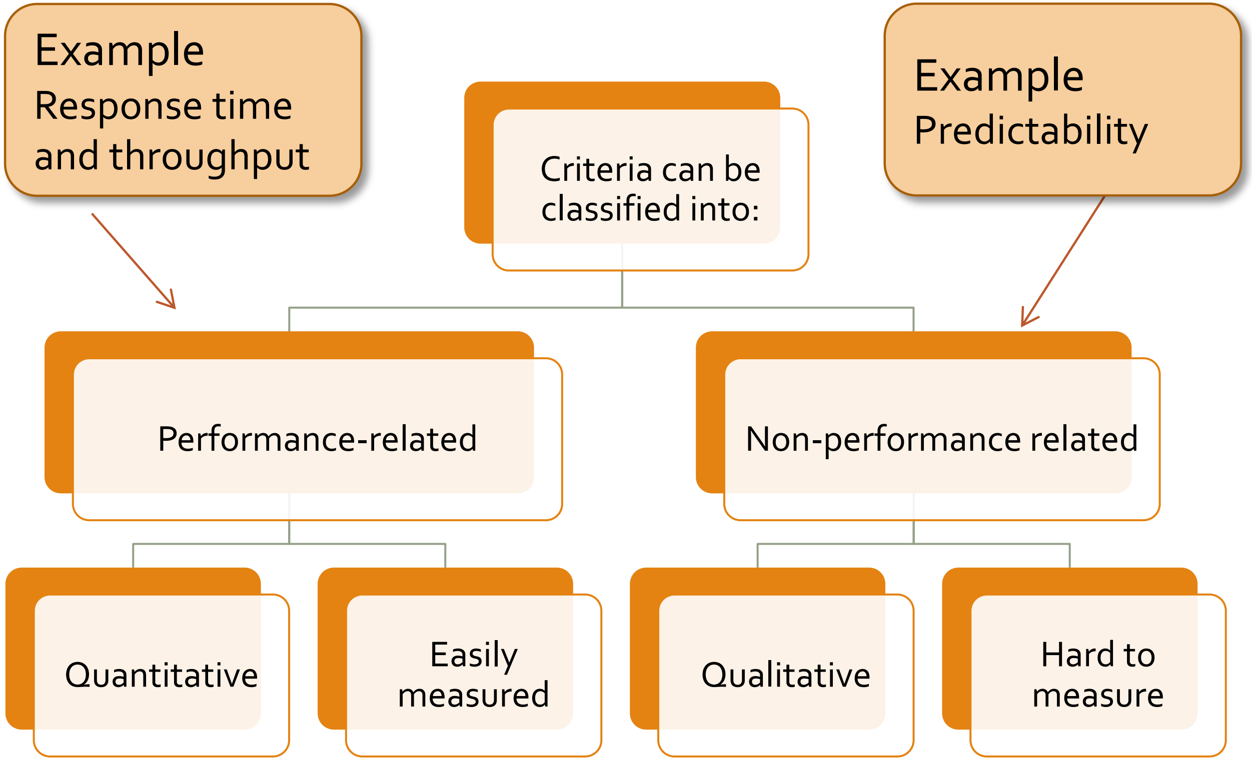
Relate to the behavior of the system as **perceived by the individual user** or process (such as **response time in an interactive system)**

System-oriented criteria

Focus is on **effective** and **efficient utilization** of the **processor** (rate at which processes are completed)

Generally of minor importance on single-user systems

Short-Term Scheduling Criteria: Performance



Scheduling Criteria

- **Max CPU utilization**
  - keep the CPU as busy as possible
- **Max Throughput**
  - # of processes that complete their execution per time unit
- **Min Turnaround time**
  - amount of time to execute a particular process
- **Min Waiting time**
  - amount of time a process has been waiting in the ready queue
- **Min Response time**
  - amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithms

Notes on the examples

- All the following examples show how scheduling algorithms work when a set of processes are in execution in the system.
- At a generic time *t=0* we will consider
  - the state of the ready queue
  - the time at which each process joins the ready queue
  - the length of the next CPU burst
- We will measure the performance in terms of average waiting time and average turnaround time

First-Come-First-Served (FCFS)

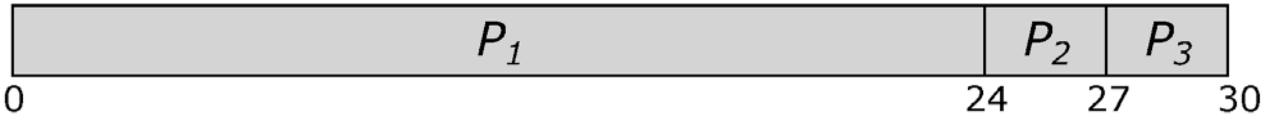
- a.k.a. first-in-first-out (FIFO) or a strict queuing scheme
- This is the simplest scheduling policy
  - easy implementation and fast execution
- When the currently running process ceases to execute, the process that has been in the ready queue the longest is selected for running
- Performs much better for long processes than short ones
- Tends to favor CPU-bound processes over I/O-bound processes
  - no pre-emption

FCFS example

Process	CPU burst
P_1	24
P_2	3
P_3	3

Let us assume that the three processes join the reqdy queue in the following order

P_1 P_2 P_3



P_2 has to wait 24 ms and P_3 has to wait 27 ms

Average waiting time:17 ms

FCFS example

Processo	CPU burst
P_1	24
P_2	3
P_3	3

Let us assume that the three processes join the reqdy queue in the following order

P_2 P_3 P_1

P_3 has to wait 3 ms and P_1 has to wait 6 ms

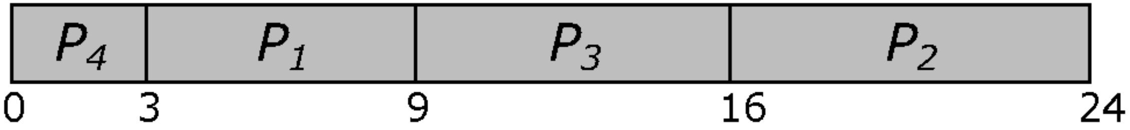
Average waiting time: 3 ms

Shortest Process Next (SPN)

- The original name of the algorithm was *Shortest Job First*
- The process with the shortest expected processing time is selected next

Process	CPU burst
P_1	6
P_2	8
P_3	7
P_4	3

Average waiting time
SPN 7 ms
FCFS 10.25 ms



SPN Performances

- SPN aims maximizing the throughput
- A short process will jump to the head of the queue
  - low predictability
- Possibility of starvation for longer processes
- One difficulty is the need to estimate the required processing time of each process
- If the programmer's estimate is substantially under the actual running time, the system may abort the job

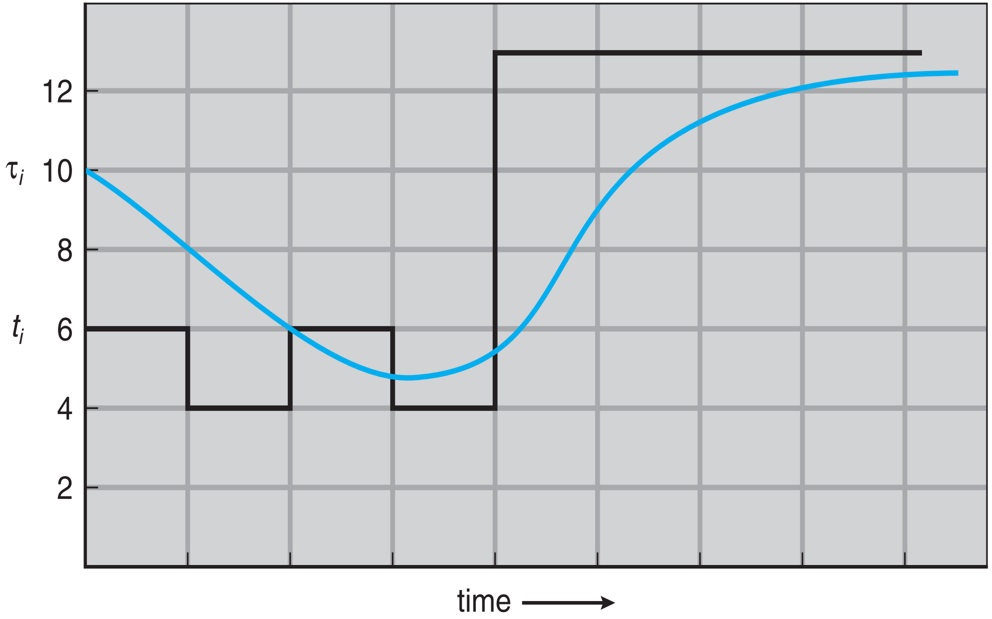
Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging

1. $t_n =$ actual length of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define: $\tau_{n=1} = \alpha\, t_n + (1 - \alpha)\tau_n.$

- Typically, α set to ½

Prediction of the Length of the Next CPU Burst



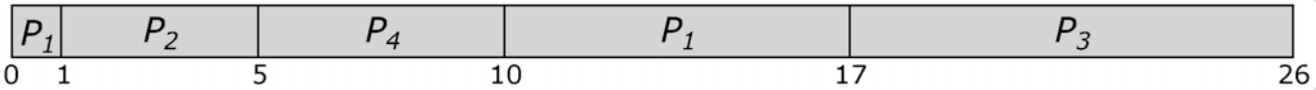
CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

Shortest Remaining Time

- This is the pre-emptive version of SPN
- The running process can be pre-empted by the new process joining the ready queue, if its CPU-burst is smaller than the CPU-burst of the running process

Process	Arrival time	CPU burst
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Average waiting time
SRT 6.5 ms
SPN 7.75 ms



Priority Scheduling

- A priority number (integer) is associated with each process
  - computed by the OS, such as for SPN scheduling, where priority is the inverse of predicted next CPU burst time
  - set by the user
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- Problem ≡ Starvation
  low priority processes may never execute
- Solution ≡ Aging
  as time progresses increase the priority of the process

Priority Scheduling

Process	CPU burst	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

0 Highest Priority

5 Lower Priority

Average waiting time 8.2 ms

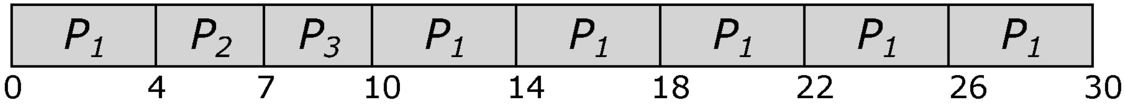
Round Robin (RR)

- Circular scheduling
- Each process gets a small unit of CPU time (time quantum q), usually 10-100 milliseconds.
- After this time has elapsed, the process is preempted and added to the end of the ready queue. The next process in the queue is scheduled.
- If there are n processes in the ready queue and the time quantum is q, no process waits more than (n-1)q time units.
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

RR example

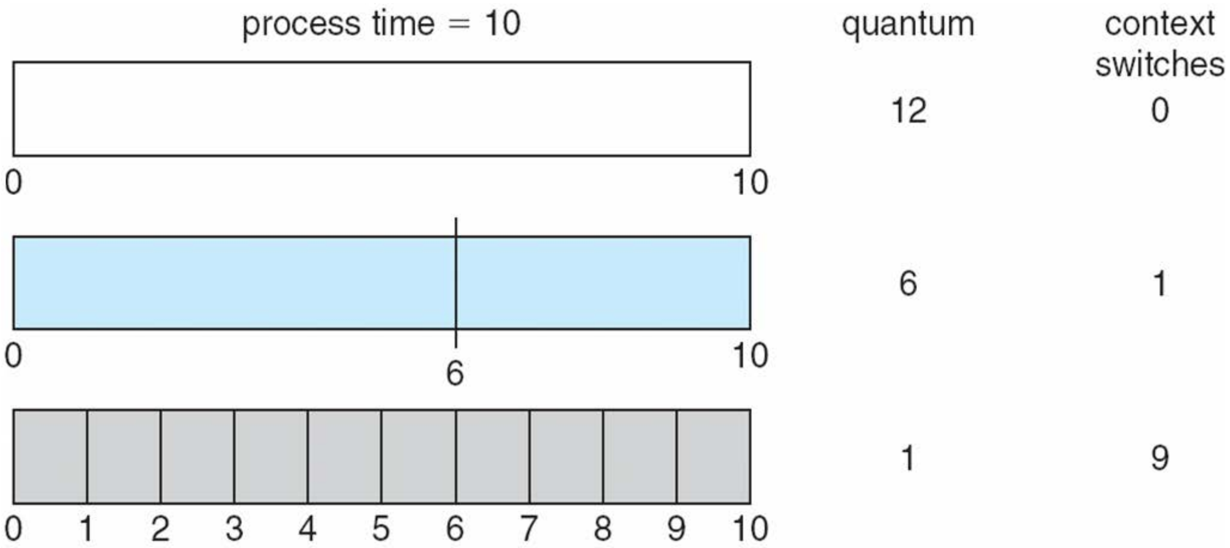
Process	CPU burst
P_1	24
P_2	3
P_3	3

time quantum q = 4ms

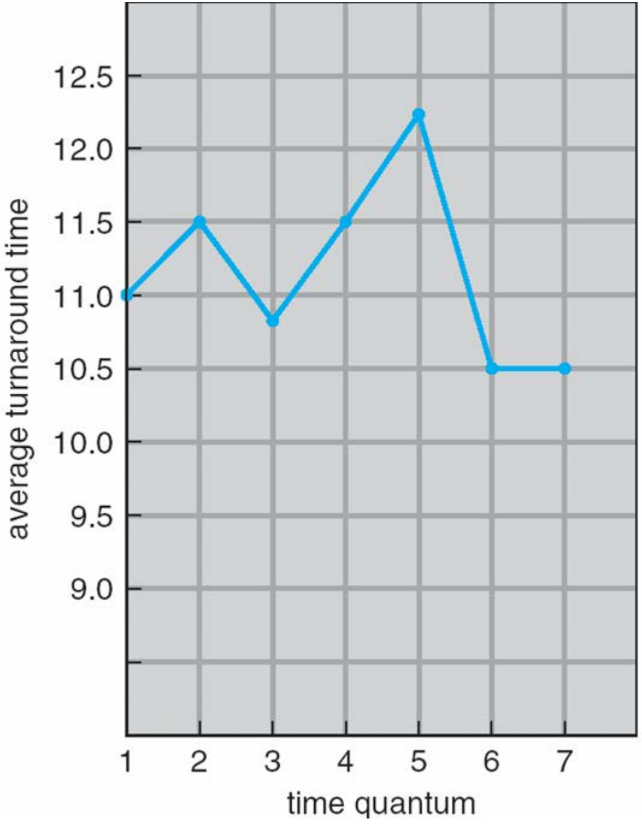


Average waiting time 5.66 ms

Time Quantum and Context Switch Time



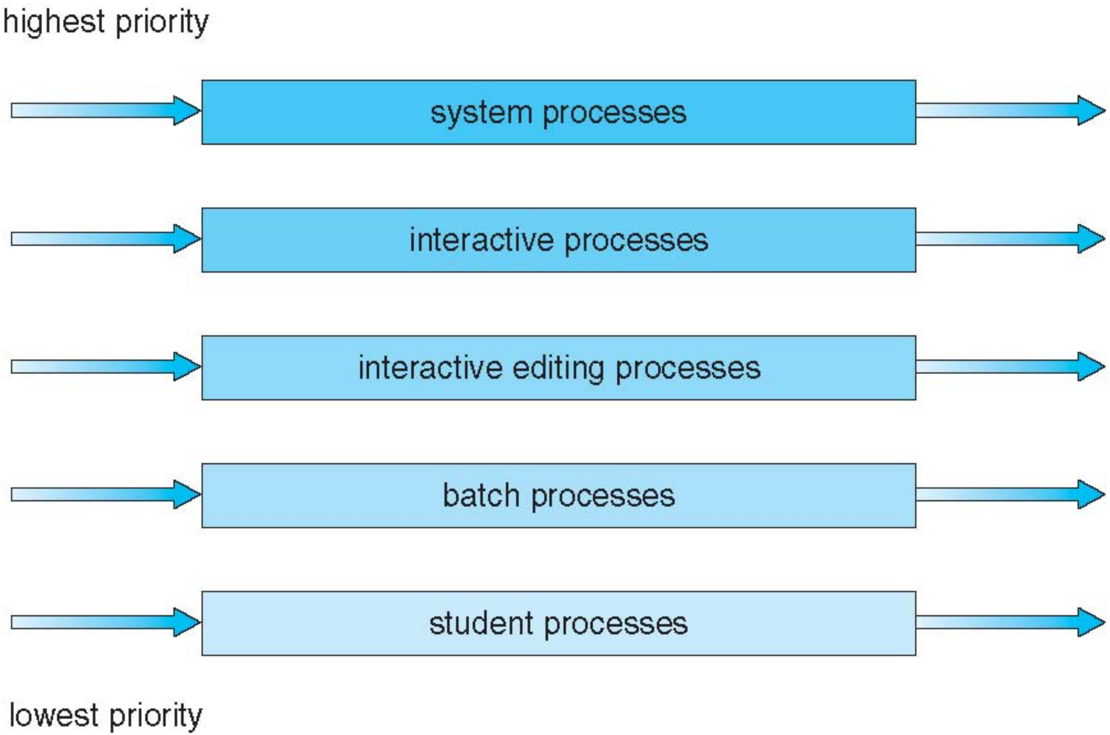
Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q

Multilevel Queue Scheduling



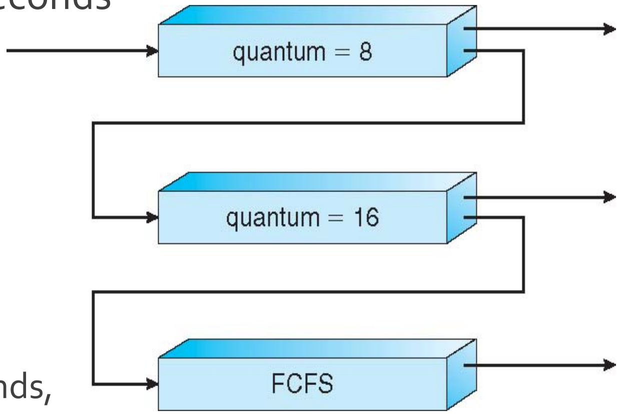
For each queue, the most appropriate scheduling algorithm is chosen

Multilevel Queue

- Ready queue is partitioned into separate queues, e.g.
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS
- 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

Example of Multilevel Feedback Queue

- Three queues:
  - Q0 – RR with time quantum 8 milliseconds
  - Q1 – RR time quantum 16 milliseconds
  - Q2 – FCFS



- Scheduling
  - A new process enters queue Q0
    - When it gains CPU, it receives 8 milliseconds
    - If it does not finish in 8 milliseconds, it is moved to queue Q1
  - At Q1 the process receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q2

Thread Scheduling

Thread Scheduling

- **When threads supported, threads scheduled, not processes**
- Distinction between user-level and kernel-level threads
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - Known as process-contention scope (PCS) since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is system-contention scope (SCS) – competition among all threads in system

Multiprocessor and Multicore Scheduling

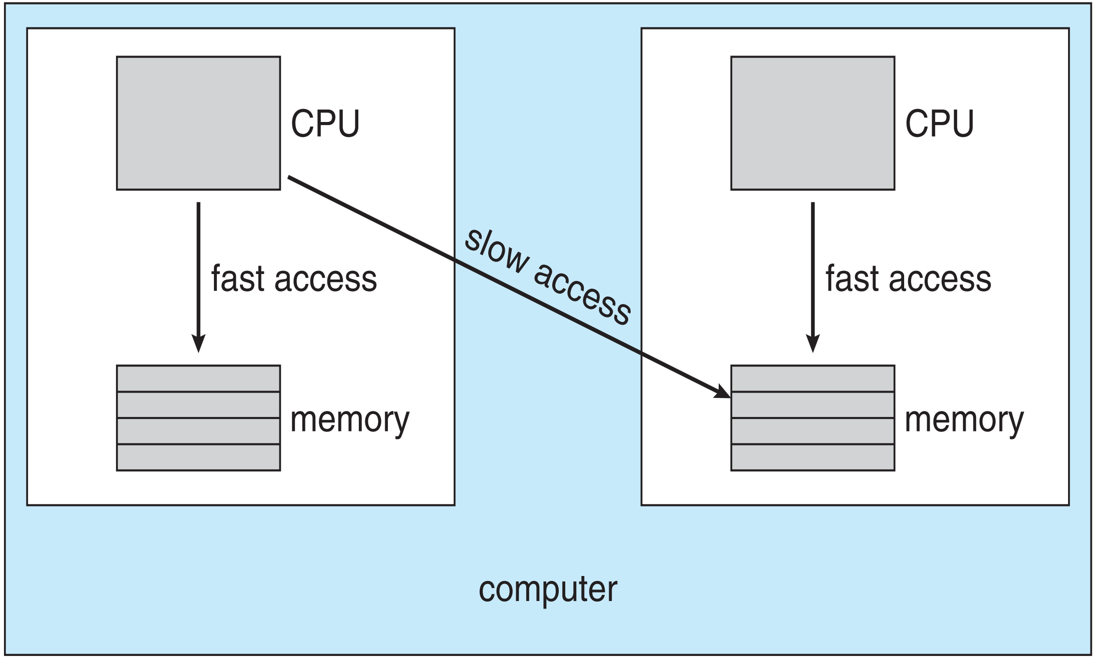
Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- **Homogeneous** processors within a multiprocessor
- Asymmetric multiprocessing
  only one processor accesses the system data structures, alleviating the need for data sharing
- Symmetric multiprocessing (SMP)
  each processor is self-scheduling, all processes share the ready queue, or each processor has its own private queue of ready processes
  - Currently, most common solution

Multiple-Processor Scheduling

- Processor affinity
  process has affinity for processor on which it is currently running
  - soft affinity, when the OS tries to bundle the process to the processor
  - hard affinity, when the OS ensure that each process always run on the same processor
  - Variations including processor sets
- Moving a process from one processor to another requires moving the associated cache content

NUMA and CPU Scheduling



Non Uniform Memory Access

Note that memory-placement algorithms can also consider affinity

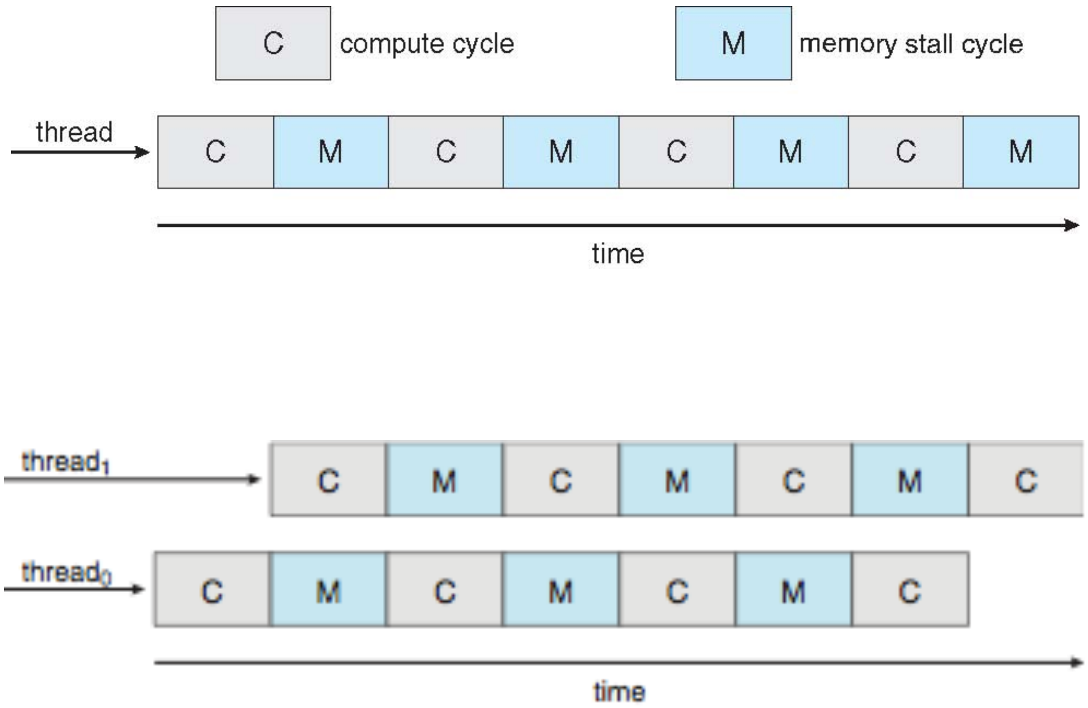
Multiple-Processor Scheduling – Load Balancing

- If SMP need to keep all CPUs loaded for efficiency
- Load balancing attempts to keep workload evenly distributed
- Push migration
  periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- Pull migration
  idle processors pulls waiting task from busy processor

Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

Multithreaded Multicore System



Operating System Examples

Linux Scheduling Through Version 2.5

- Prior to kernel version 2.5, ran variation of standard UNIX scheduling algorithm
- Version 2.5 moved to constant order O(1) scheduling time
  - Preemptive, priority based
  - Two priority ranges: time-sharing and real-time
  - Real-time range from 0 to 99 and nice value from 100 to 140
  - Map into global priority with numerically lower values indicating higher priority
  - Higher priority gets larger q
  - Task run-able as long as time left in time slice (active)
  - If no time left (expired), not run-able until all other tasks use their slices
  - All run-able tasks tracked in per-CPU runqueue data structure
    - Two priority arrays (active, expired)
    - Tasks indexed by priority
    - When no more active, arrays are exchanged
- Worked well, but poor response times for interactive processes

Linux Scheduling in Version 2.6.23 +

- Completely Fair Scheduler (CFS)
- Scheduling classes
  - Each has specific priority
  - Scheduler picks highest priority task in highest scheduling class
  - Not quantum based, but based on proportion of CPU time
- Quantum calculated based on nice value from -20 to +19
  - Lower value is higher priority
  - Calculates target latency – interval of time during which task should run at least once
  - Target latency can increase if number of active tasks increases
- CFS scheduler maintains per task virtual run time in variable vruntime
  - Associated with decay factor based on priority of task – lower priority is higher decay rate
  - Normal default priority yields virtual run time = actual run time
- To decide next task to run, scheduler picks task with lowest virtual run time

Windows Scheduling

- Windows uses priority-based preemptive scheduling
- Highest-priority thread runs next
- Thread runs until (1) blocks, (2) uses time slice, (3) preempted by higher-priority thread
- Real-time threads can preempt non-real-time
- 32-level priority scheme
- Variable class is 1-15, real-time class is 16-31
- Priority 0 is memory-management thread
- Queue for each priority
- If no run-able thread, runs idle thread

Windows Priorities

	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
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Algorithm Evaluation

Algorithm Evaluation

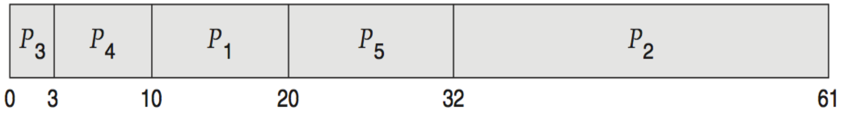
- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- **Deterministic modeling**
  Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

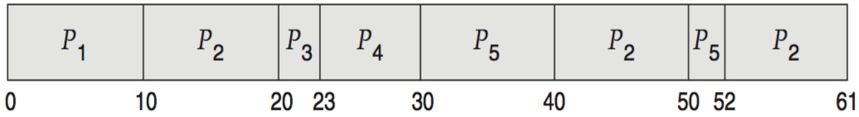
Deterministic Evaluation

- For each algorithm, calculate minimum average waiting time
- Simple and fast, but requires exact numbers for input, applies only to those inputs
  - FCFS is 28ms:

  - SPN is 13ms:



  - RR is 23ms:



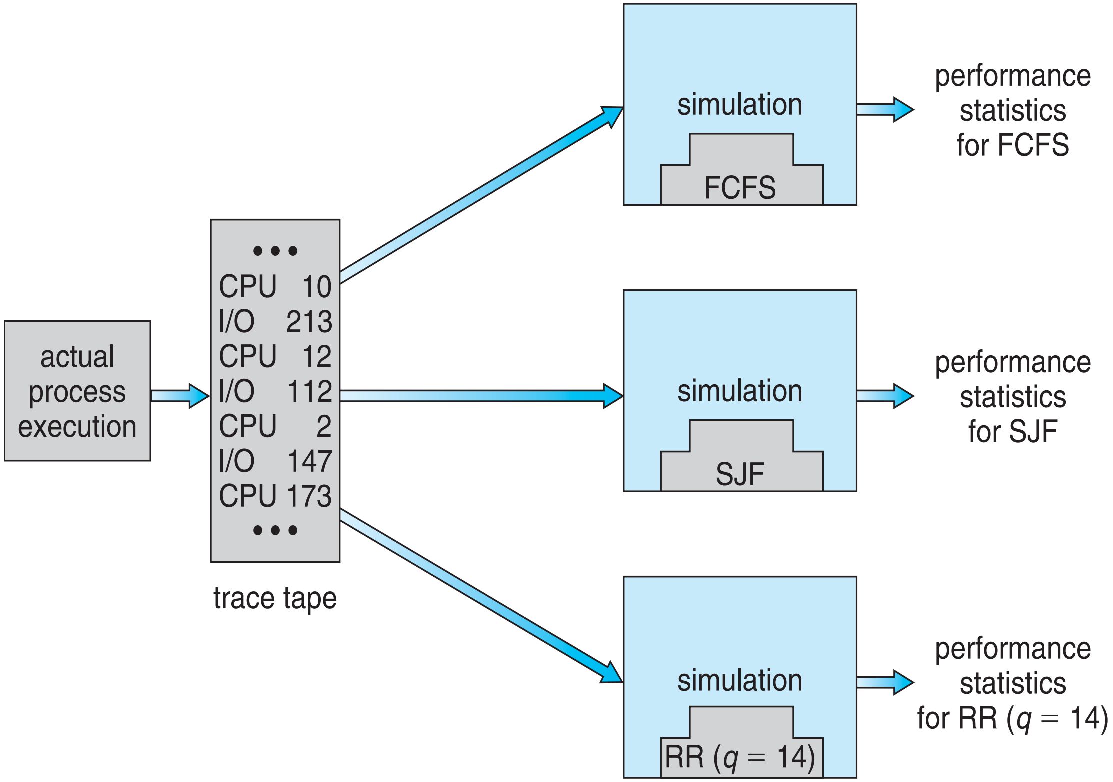
Queueing Models

- Describes the arrival of processes, and CPU and I/O bursts probabilistically
  - Commonly exponential, and described by mean
  - Computes average throughput, utilization, waiting time, etc
- Computer system described as network of servers, each with queue of waiting processes
  - Knowing arrival rates and service rates
  - Computes utilization, average queue length, average wait time, etc

Simulations

- Programmed model of computer system
- Clock is a variable
- Gather statistics indicating algorithm performance
- Data to drive simulation gathered via
  - Random number generator according to probabilities
  - Distributions defined mathematically or empirically
  - Trace tapes record sequences of real events in real systems

Evaluation of CPU Schedulers by Simulation



Implementation

- Even simulations have limited accuracy
- Just implement new scheduler and test in real systems
  - High cost, high risk
  - Environments vary
- Most flexible schedulers can be modified per-site or per-system
- Or APIs to modify priorities
- But again environments vary