Chapter 5: Scheduling

CPU Scheduling

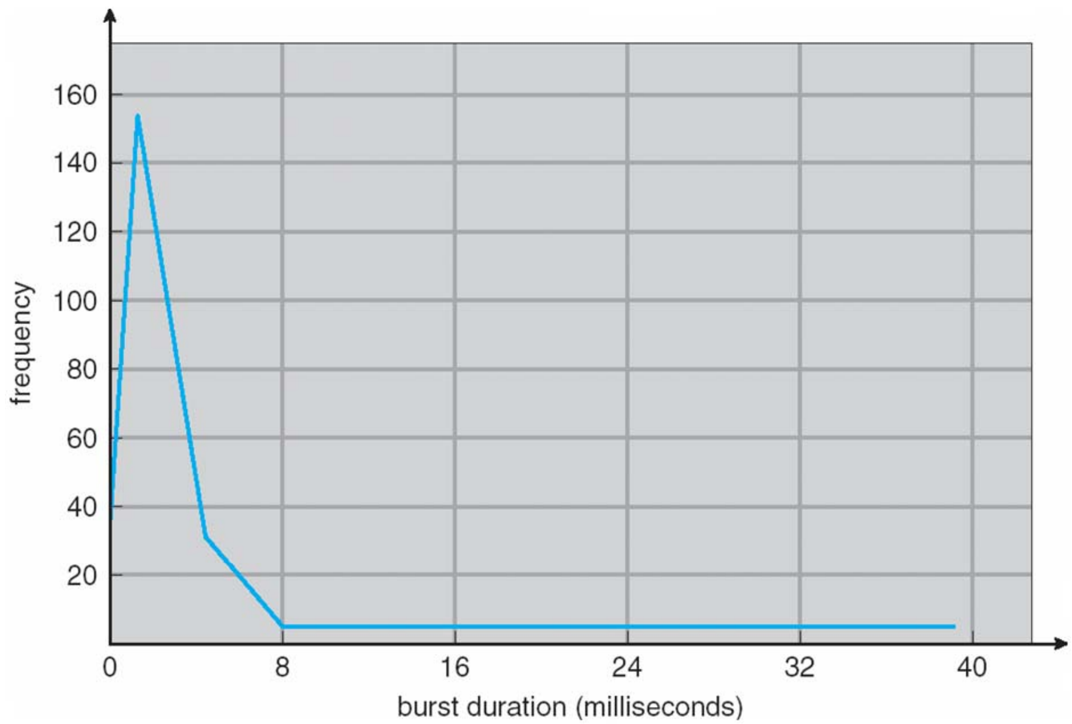
- Objectives
  - To introduce CPU scheduling, basis for multi-programmed operating systems
  - To describe selected CPU scheduling algorithms
  - To discuss evaluation criteria for selecting a CPU scheduling

- Topics
  - Basic Concepts
  - Scheduling Criteria
  - Scheduling Algorithms
  - Multiple-Processor Scheduling
  - Operating Systems Examples

Basic Concepts

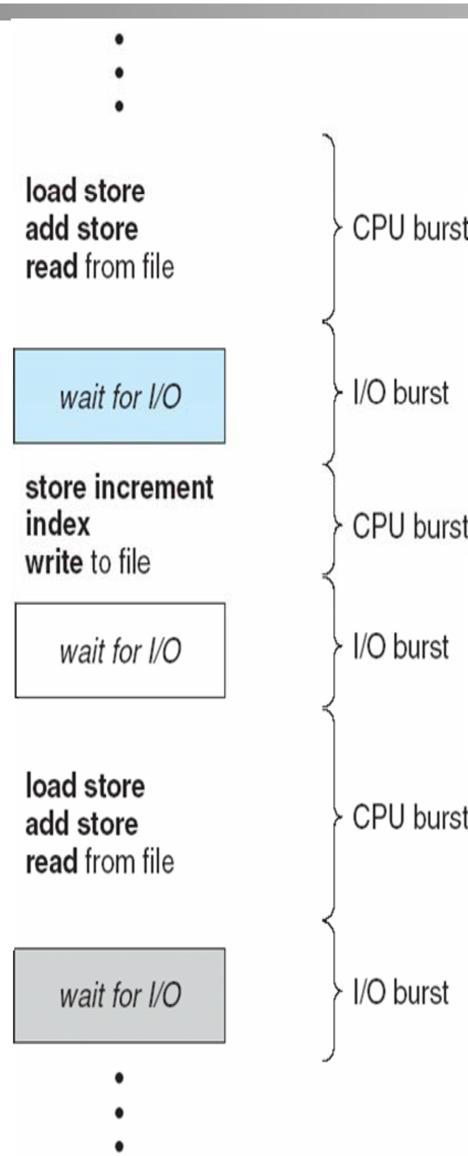
- Maximum CPU utilization only obtained with concurrent multi-programming

- CPU–I/O Burst Cycle
  – Process execution consists of a cycle of CPU execution and I/O wait



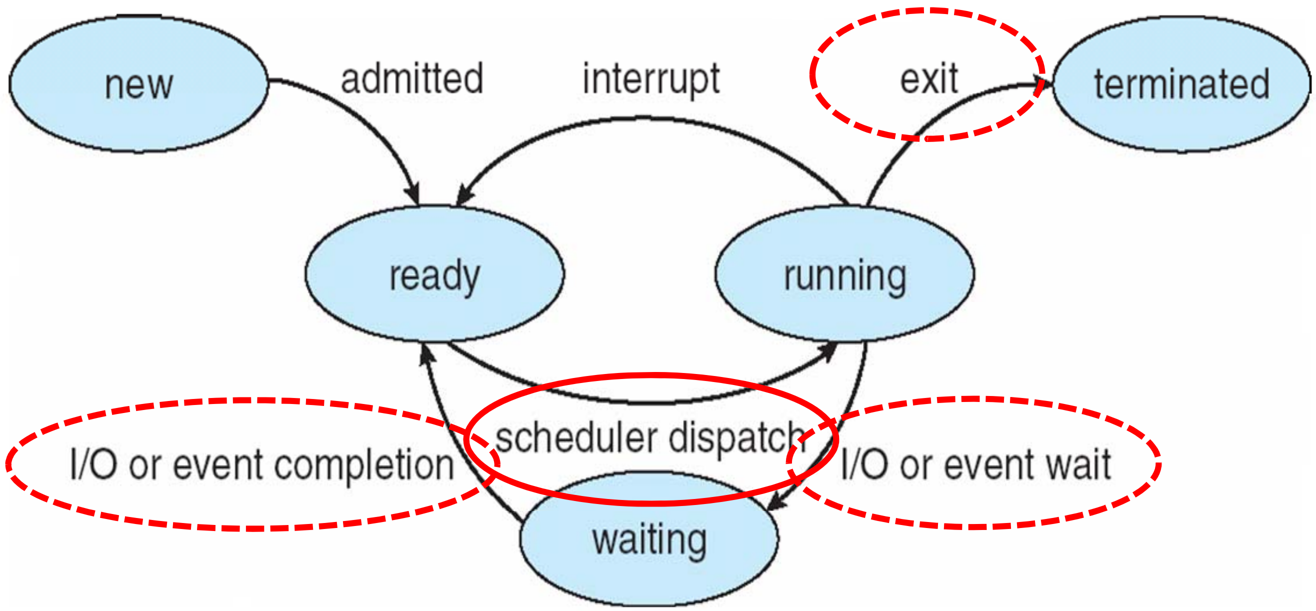
- CPU burst distribution
  – Measured extensively
  – Vary greatly from processes, computers, applications

Alternating Sequence of CPU and I/O Bursts



Process States (Reminder)

- As a process executes, it frequently changes states
  - new: the process is being created
  - running: instructions are being executed
  - waiting: the process is waiting for some event to occur
  - ready: the process is waiting to be assigned to a processor
  - terminated: the process has finished execution



CPU Scheduler

- Selects from among processes in memory that are ready to execute and allocates CPU to one of them
- 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 ready
  4. Terminates
- Scheduling under 1 and 4 is non-preemptive
  – Processes cooperate by giving up the CPU voluntarily
- All other scheduling is preemptive
  – Running CPU execution interrupted, triggered by outside event

Dispatcher Module

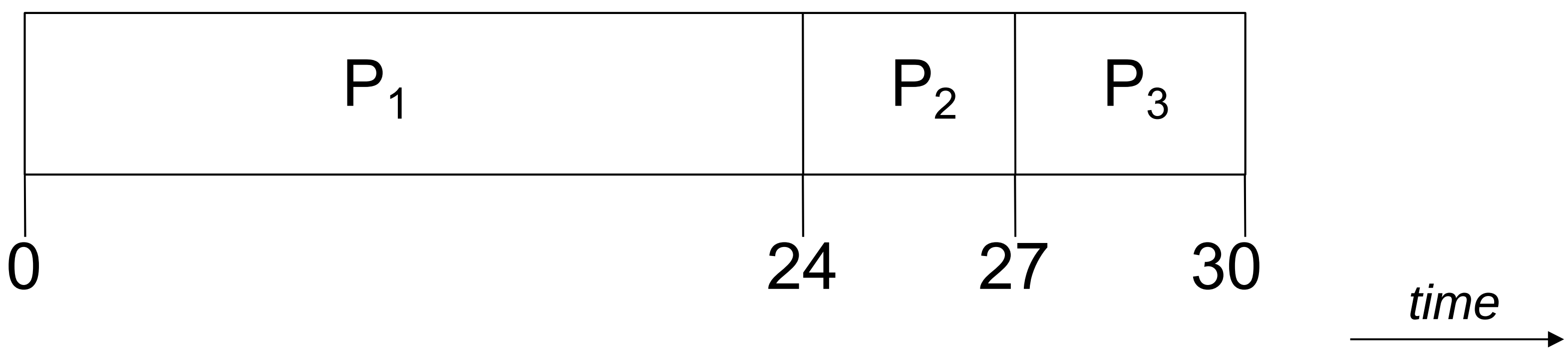
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler
- This involves:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program
- Dispatch latency
  - Time it takes for the dispatcher to stop one process and start another running
  - Must be very low to avoid overhead

Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- Waiting time – amount of time a process has been waiting in the ready queue
- 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)

First Come First Serve (FCFS) Scheduling

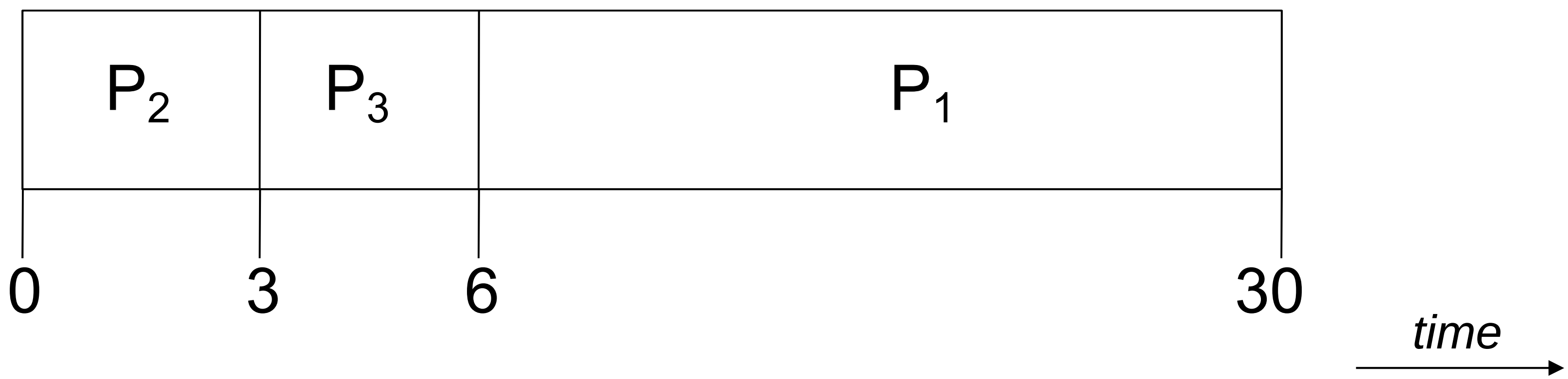
Process	Burst Time
P_1	24
P_2	3
P_3	3



- Suppose that processes arrive in order P_1 , P_2 , P_3
- The Gantt chart (bars for start/end times) 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 (2)

- Suppose that processes arrive in order P_2, P_3, P_1

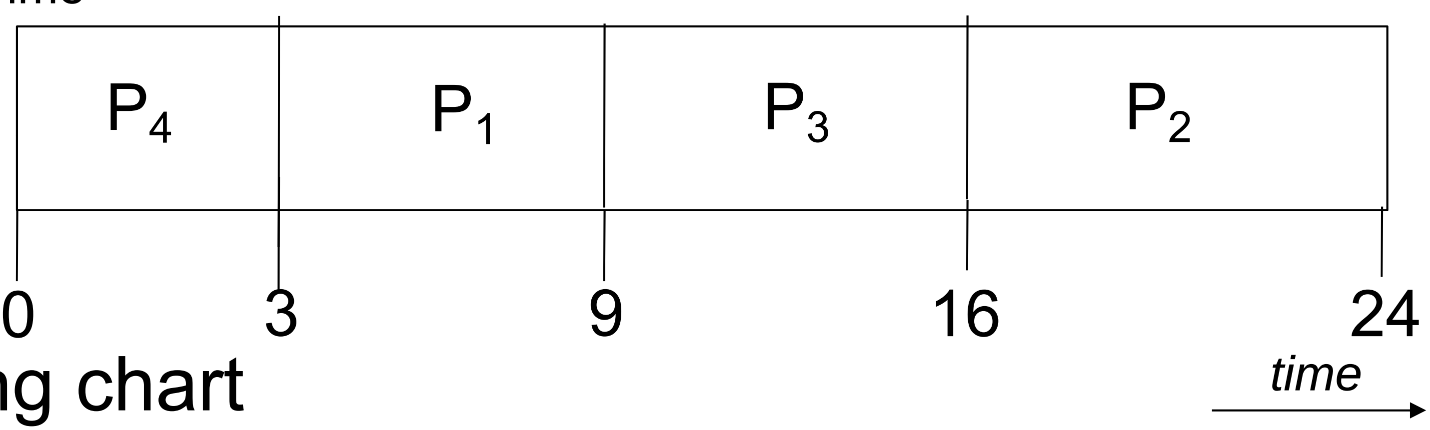


- The Gantt chart for the schedule is
  - Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
  - Average waiting time: $(6 + 0 + 3)/3 = 3$
    - Much better than previous case
  - Convoy effect on short processes behind long processes

Shortest Job First (SJF) Scheduling

- Associate with each process the length of its next CPU burst and use these lengths to schedule the process with the shortest time to run and complete first
- SJF is optimal
  - Gives minimum average waiting time for given set of processes
  - The difficulty is knowing the length of the next CPU request

Process	Burst Time
P1	6
P2	8
P3	7
P4	3



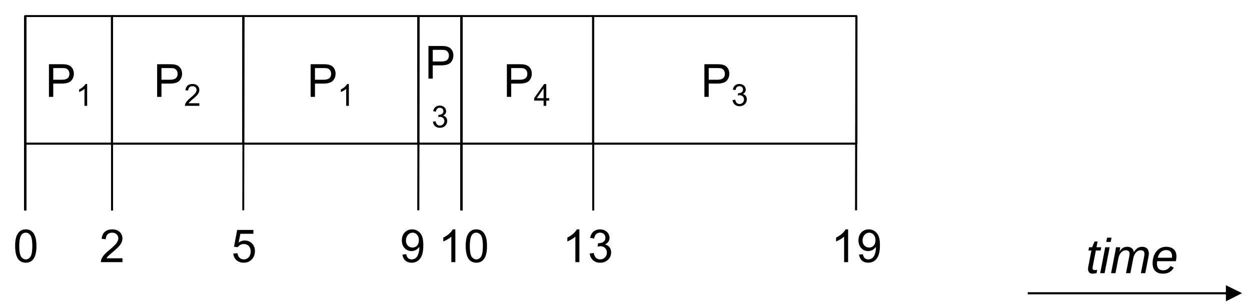
- SJF scheduling chart
  - Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Shortest Remaining Time (SRT) Scheduling

- Preemptive SJF scheduling, allowing *new* processes to interrupt the currently running one

Process	Arrival Time	Burst Time
P_1	0.0	6
P_2	2.0	3
P_3	4.0	7
P_4	10.0	3

- SRT scheduling chart

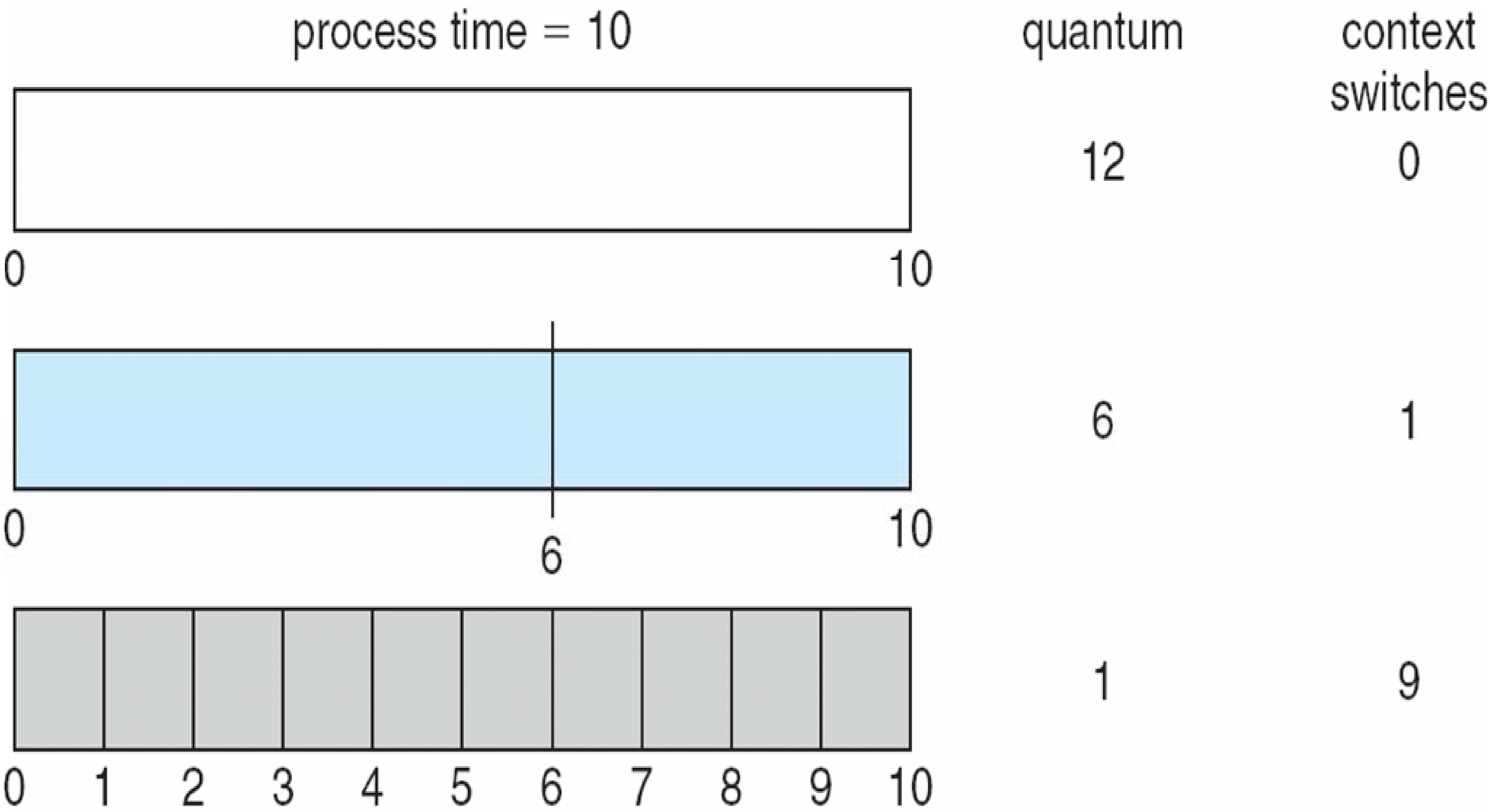


– Average waiting time = (3 + 0 + 5+3 + 0) / 4 = 2.75

Round Robin (RR) Scheduling

- Process gets a small unit of CPU time (*time quantum*),
  - After time elapsed (usually 10-100 ms), process 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, 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
  - n processes run in "parallel" at $1/n$ of available CPU speed
- Performance
  - q too large $\Rightarrow$ FIFO
  - q too small $\Rightarrow$ many context switches, overhead is too high
    - q must be large with respect to context switch costs

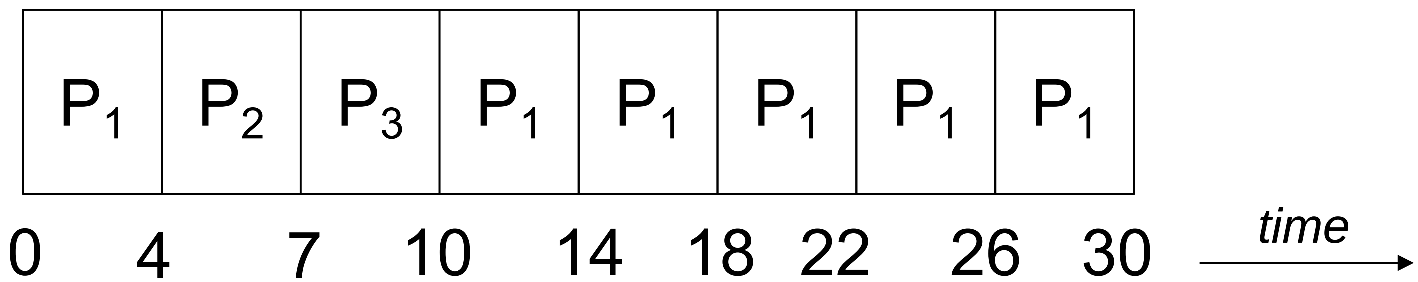
Time Quantum and Context Switch Time



Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is



- Typically, higher average turnaround than SJF, but better response time
  – Average RR turnaround time (30 + 7 + 10) / 3 = 15.6
  – Comparison to SJF turnaround time (3 + 6 + 30) / 3 = 13

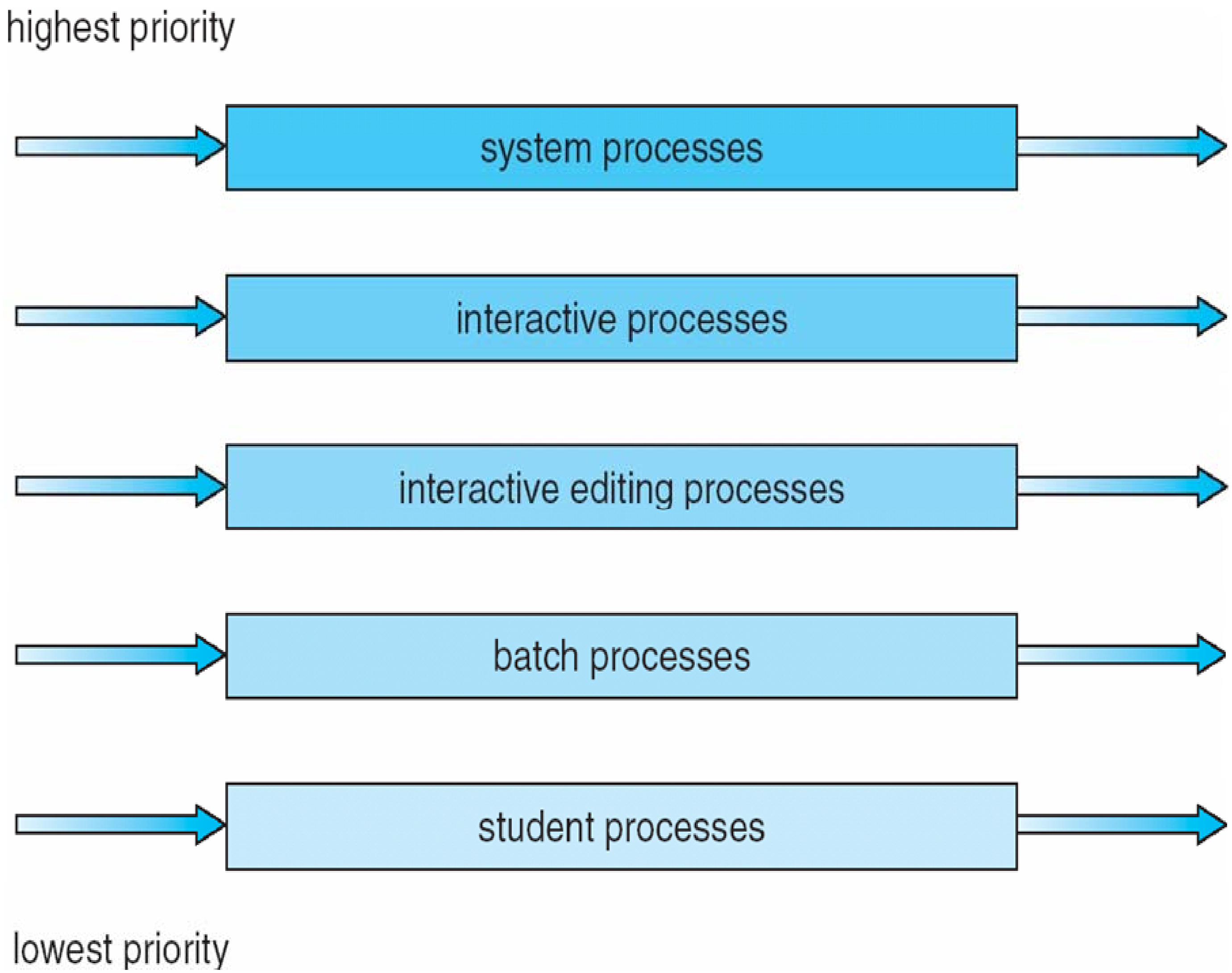
Priority Scheduling

- Priority number (integer) is associated with each process
- CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
- FCFS is a priority scheduling, where priority is the process' arrival time: Non-preemptive
- SJF is a priority scheduling, where priority is the (predicted) next CPU burst time: Preemptive
- Problem ≡ Starvation
  - Low priority processes may never execute
- Solution ≡ Aging
  - As time progresses increase the priority of the process

Multi-level (ML) Queue Scheduling

- Ready queue is partitioned into separate queues, *e.g.*:
  - Foreground (interactive)
  - Background (batch)
- 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
    - *I.e.*, 80% to foreground in RR and 20% to background in FCFS

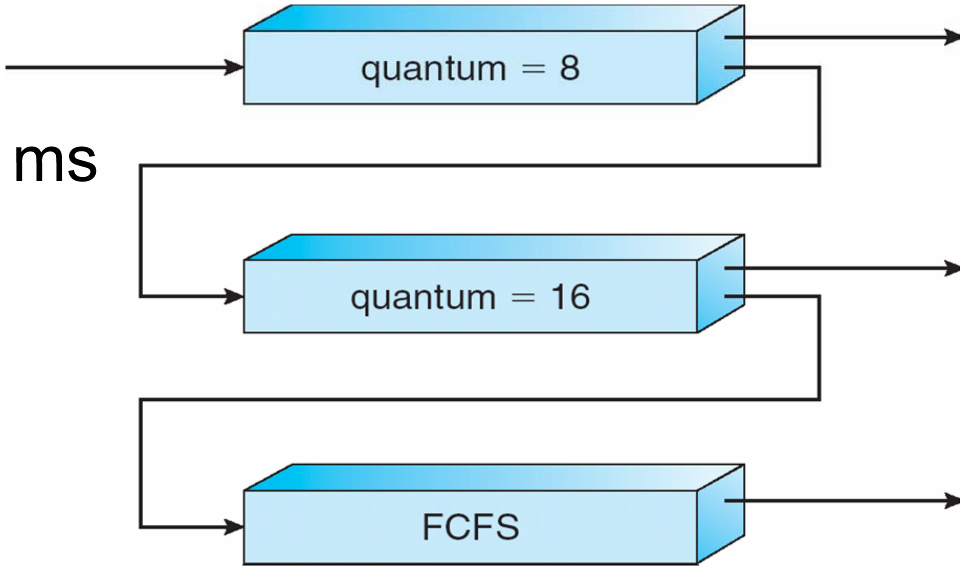
Example for ML Queue Scheduling



Multi-level Feedback Queue (MLF)

- A process can move between the various priority queues
  - Aging can be implemented this way
- Multi-level 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

Example of a 3-Level MLF



- Three queues
  - Q_0 – RR with time quantum 8 ms
  - Q_1 – RR time quantum 16 ms
  - Q_2 – FCFS
- Scheduling
  - New job enters queue Q_0 which is served RR. When it gains CPU, job receives 8 ms. If it does not finish in 8ms, job is preempted and moved to queue Q_1
  - At Q_1 job is again served RR and receives 16 additional ms. If it still does not complete, it is preempted and moved to queue Q_2
  - Final queue Q2 is served FCFS

Multiple Processor Scheduling (1)

- CPU scheduling is more complex when multiple CPUs are available
  - Homogeneous processors within a multi-processor assumed
- Asymmetric multiprocessing (AMP)
  - Only one processor accesses system data structures
  - Alleviating the need for data sharing
- Symmetric multiprocessing (SMP)
  - Each processor is self-scheduling
  - All processes in common ready queue or each has its own private queue of ready processes
  - Supported, *e.g.*, in Windows XP, Mac OS X, Linux, Solaris

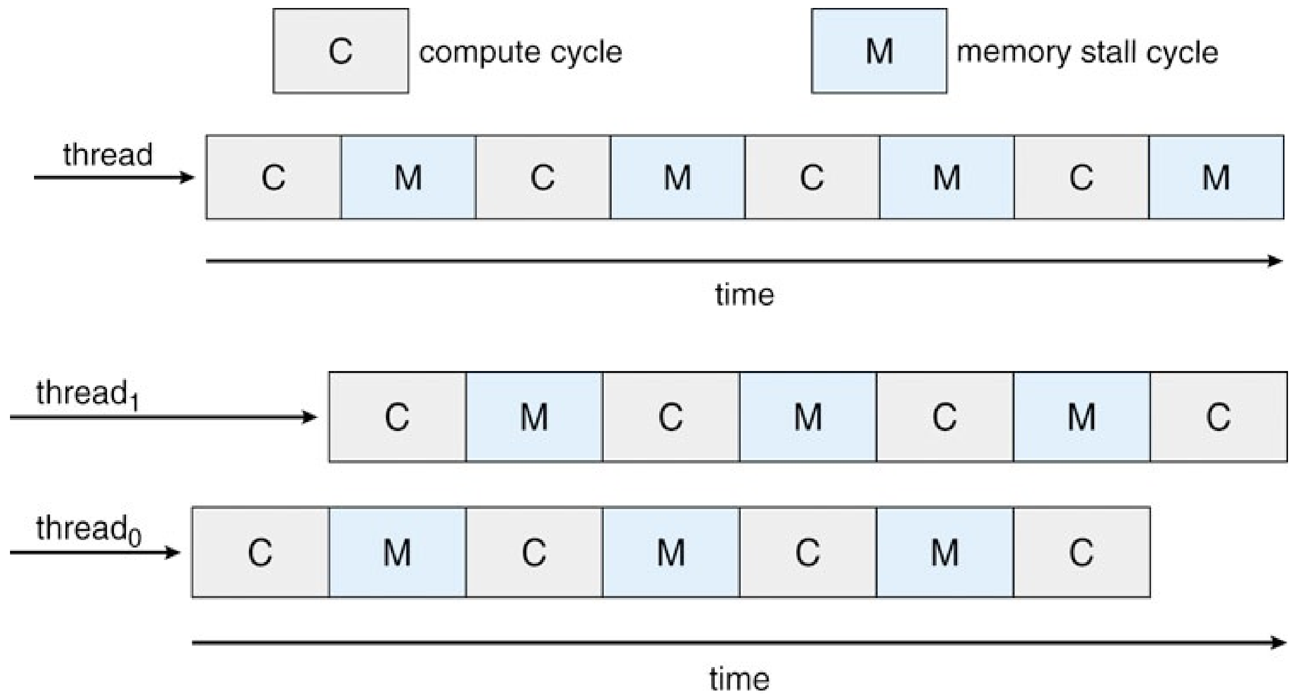
Multiple Processor Scheduling (2)

- Processor affinity
  - Process has affinity for processor on which it is currently running
    - Hard affinity
      - Process runs only on one particular CPU
    - Soft affinity
      - Process can migrate to another CPU, but reluctantly done

- Process migration involves cache invalidation on one CPU and repopulation on the other

Multi-core Processors

- Trend since 2010 to place multiple processor cores on same physical chip
  - Faster, consumes less power
- Multiple kernel threads per core also growing
  - Takes advantage of memory stall to make progress on another thread, while memory retrieve happens



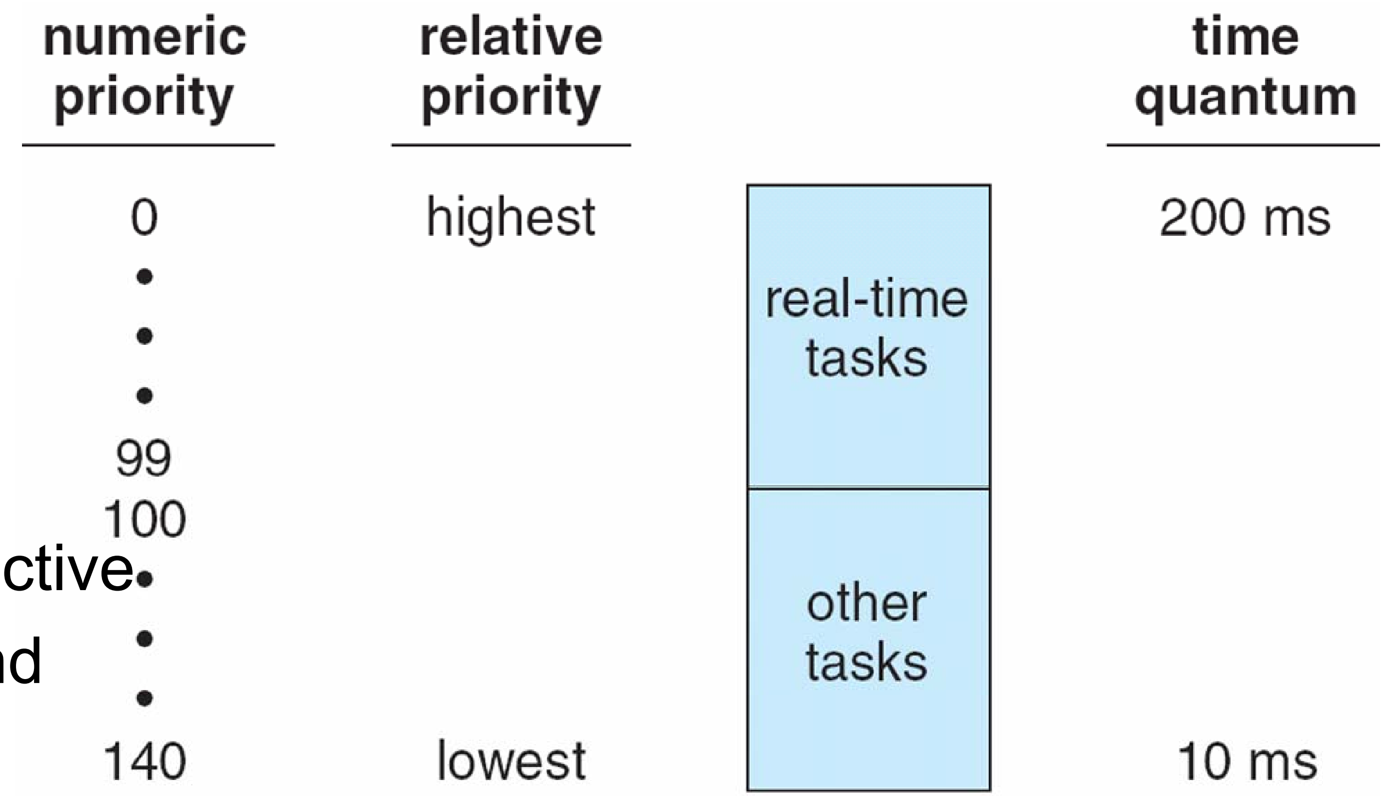
Example 1: Linux Scheduling/Priorities (1)

- Constant order O(1) scheduling time independent of number of processes
  - Suitable for large and SMP systems

- Two priority ranges
  - Real-time
    - Real-time range from 0 to 99
  - Time sharing (nice values)
    - Nice value from 100 to 140
  - Longer time quantum for higher priority

Example 1: Linux Scheduling/Priorities (2)

- Time sharing tasks have dynamic priorities, adjusted after expired time quantum
  - Level of interactivity determines ±5 adjustment
  - Long sleep times → more interactive
  - Short sleep times → less interactive
    - CPU bound



Example 2: Java Thread Scheduling

- JVM uses a preemptive, priority-based scheduling algorithm
  - FIFO queue is used if there are multiple threads with the same priority

- JVM schedules a thread to run when
  - The currently running thread exits the runnable state
  - A higher priority thread enters the runnable state

- Note: the JVM does not specify whether threads are time-sliced or not