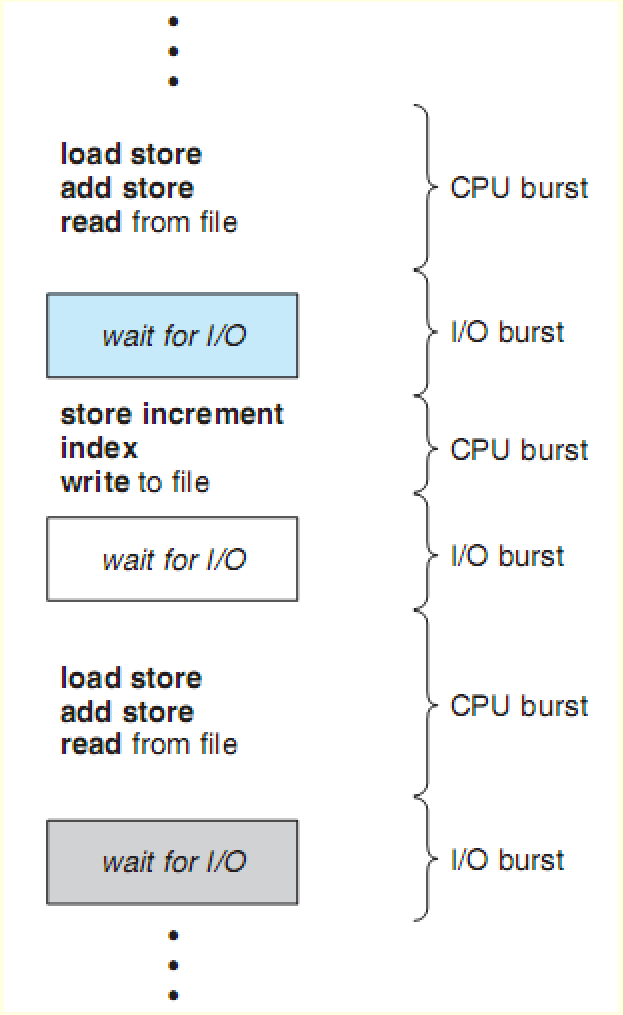
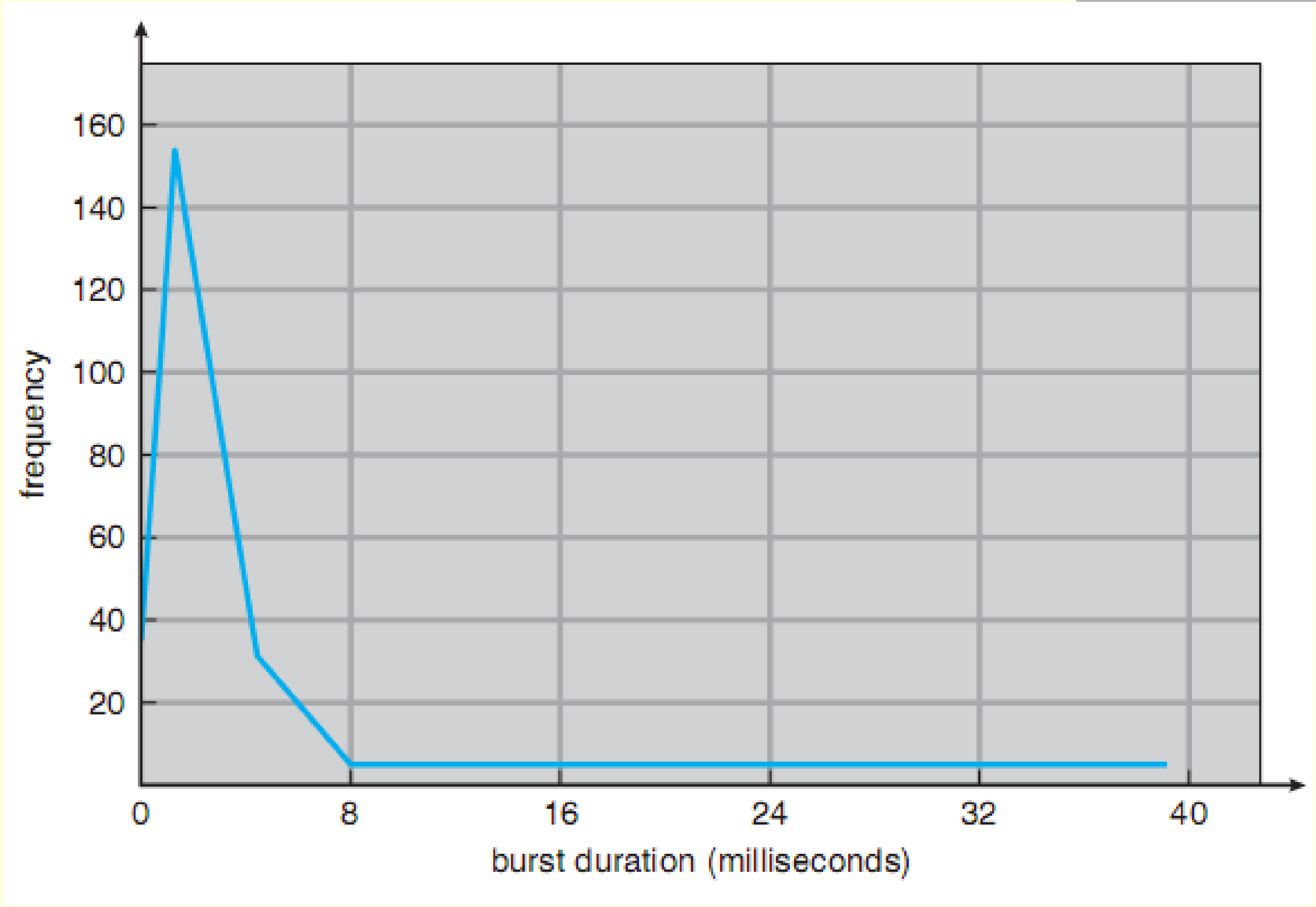
CPU Scheduling

Mehdi Kargahi
School of ECE
University of Tehran
Summer 2016

CPU and I/O Bursts



Histogram of CPU-Burst Durations



When CPU Scheduling Occurs

1. Running process switches to wait state
   - I/O request
   - Waiting for termination of one of the child processes
2. Running process switches to ready state (timer interrupt)
3. Waiting process switches to ready state (I/O completion)
4. Running process terminates

Under <u>non-preemptive</u> scheduling: 1 and 4 are legal scheduling points

Under <u>preemptive</u> scheduling: 1 through 4 are legal Scheduling points

Preemptive Scheduling

- Inconsistency of shared data should be controlled
- During the processing of a system call or doing I/O, preemption may result inconsistency
  - Most kernel codes are non-preemptive
  - These portions of code with disabled interrupts do not occur very often and typically contain few instructions
  - Problematic for real-time systems
  - Linux RedHat 2.6 kernel is preemptable

Dispatcher

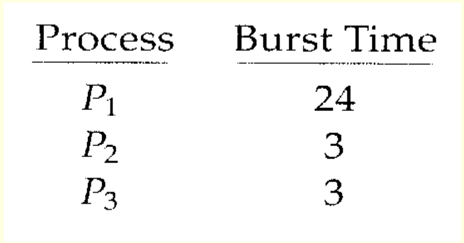
- The dispatcher function includes
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program
- *Dispatch latency* must be short

Scheduling Criteria

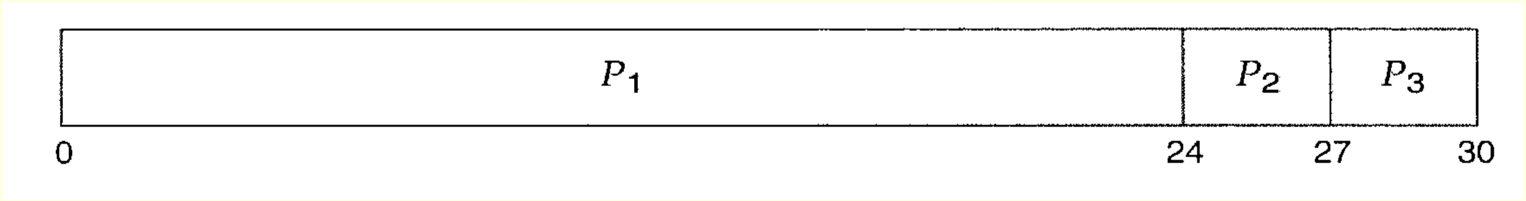
- Some criteria for selecting an scheduling algorithm are as follows:
  1. CPU utilization
     - Is 100% good for utilization?
     - Normally between 40% and 90%
  2. Throughput
     - Long processes have small throughput
  3. Turnaround time
     - Includes loading, waiting in ready queue, execution, I/O, …
  4. Waiting time
  5. Response time
- Maximizing 1 and 2
- Minimizing 3,4, and 5 (e.g., minimizing the maximum response time)

Scheduling Algorithms

- FCFS: First-Come First-Served



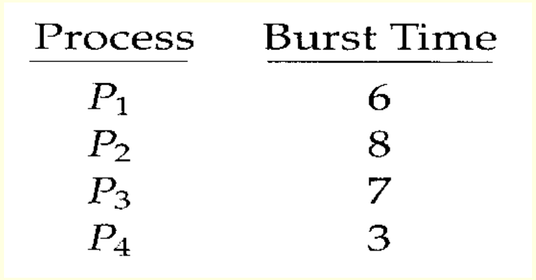
Process	Burst Time
P_1	24
P_2	3
P_3	3



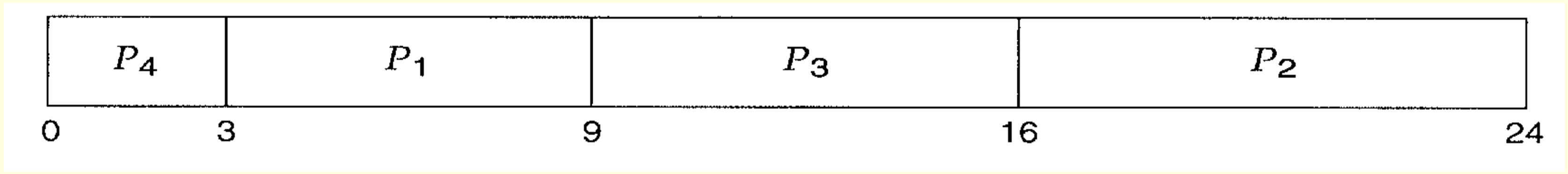
- AWT: 17
- Generally, the AWT is not minimal. LCFS?
- Convoy effect: All the other processes wait for the one big process to get off the CPU
- FCFS is not proper for time-sharing

Scheduling Algorithms

- SJF: Shortest-Job-First



Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3



- AWT: 7 (minimal)
- *Proof* ?
- How can we find the length of CPU-burst of jobs?

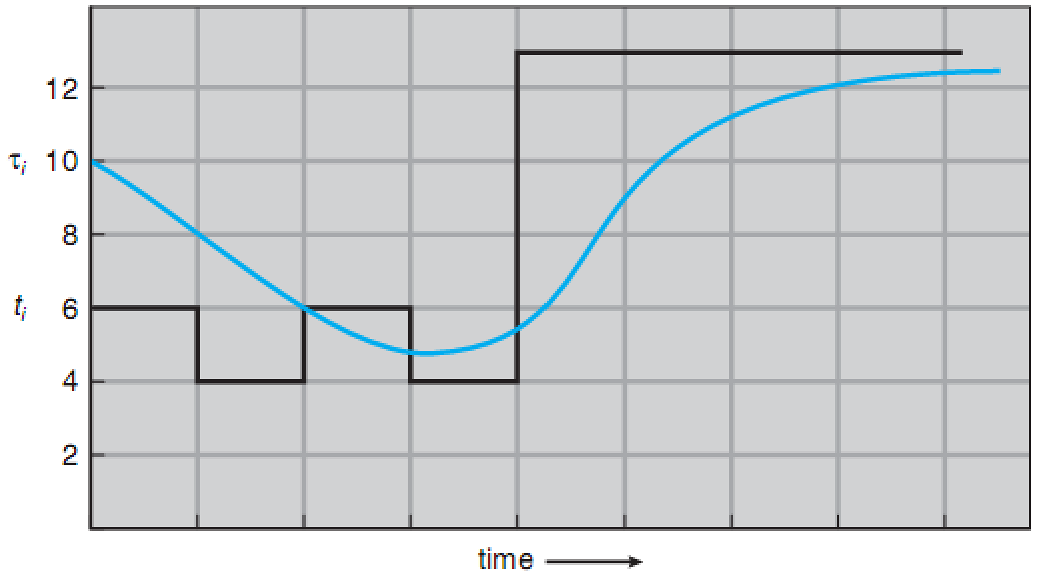
Exponential Average for CPU-Burst Prediction

$0 \leq \alpha \leq 1$

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

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \cdots + (1 - \alpha)^j \alpha t_{n-j} + \cdots + (1 - \alpha)^{n+1}\tau_0.$$

$\alpha = 0.5 \rightarrow$

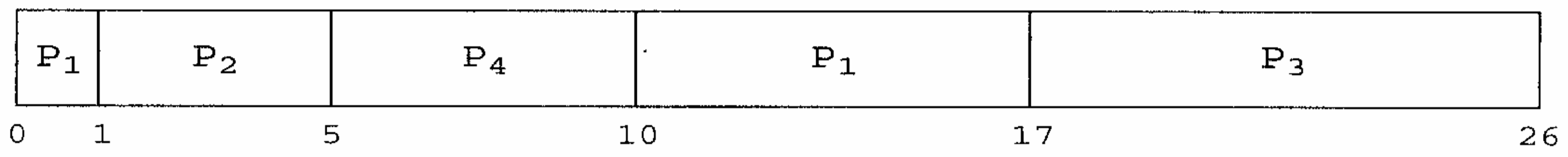


CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

Scheduling Algorithms

- SRTF: Shortest-Remaining-Time-First

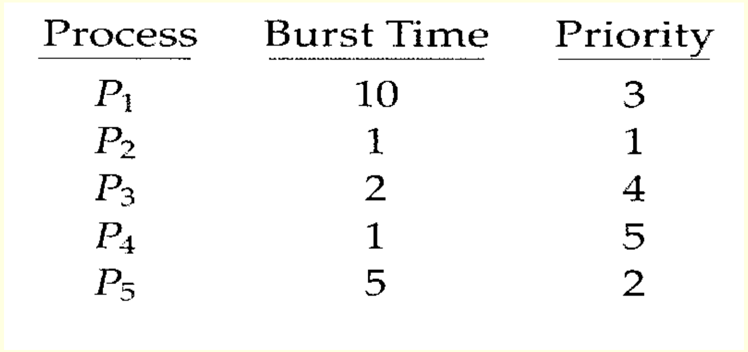
Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



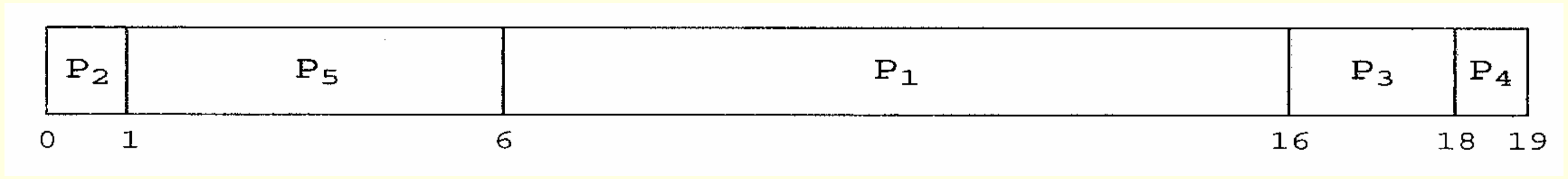
- AWT: 6.5
- AWT (SJF): 7.75

Scheduling Algorithms

- Priority Scheduling
  - Each process has a priority (e.g., 0..127)
  - Equal-priority processes are scheduled according to another algorithm, e.g., FCFS



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

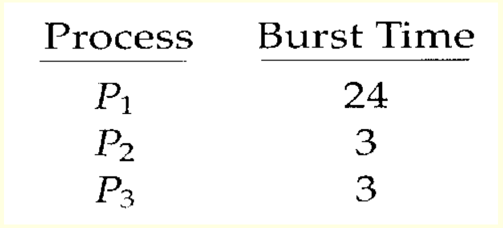


Priority Scheduling

- Starvation (Indefinite blocking)
  - E.g., SJF
- IBM 7094 at MIT
  - Duration of running a low-priority process:1967-1973
- Solution
  - Aging: gradually increasing the priority of processes (according to timer interrupts) that wait in the system for a long time

Scheduling Algorithms

- RR: Round-Robin
  - Preemptive
- Time-slice, time-quantum, time-slot (according to timer interrupts)
- Ex.: q=4



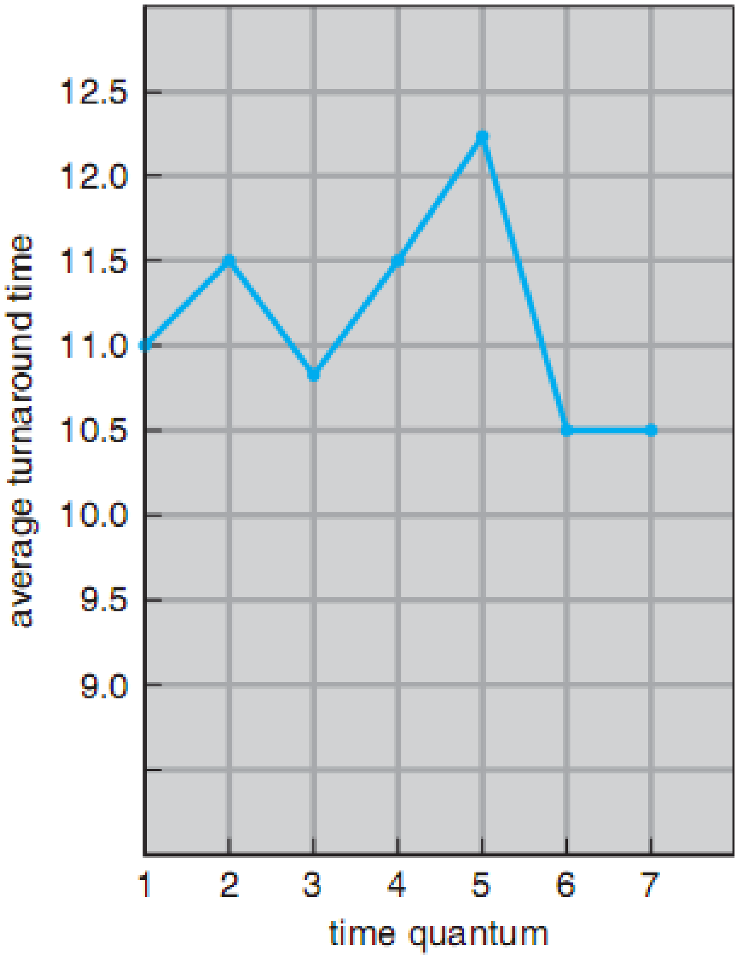
Process	Burst Time
P_1	24
P_2	3
P_3	3

- AWT: 5.66

Round-Robin

- When time quantum becomes very large, RR degenerates to FCFS
- When time quantum becomes very small, RR is known as *processor sharing* (CS overhead becomes more important)
  - CS overhead should normally be 0.01% to 0.1%
- A rule of thumb
  - 80% of CPU bursts < time quantum

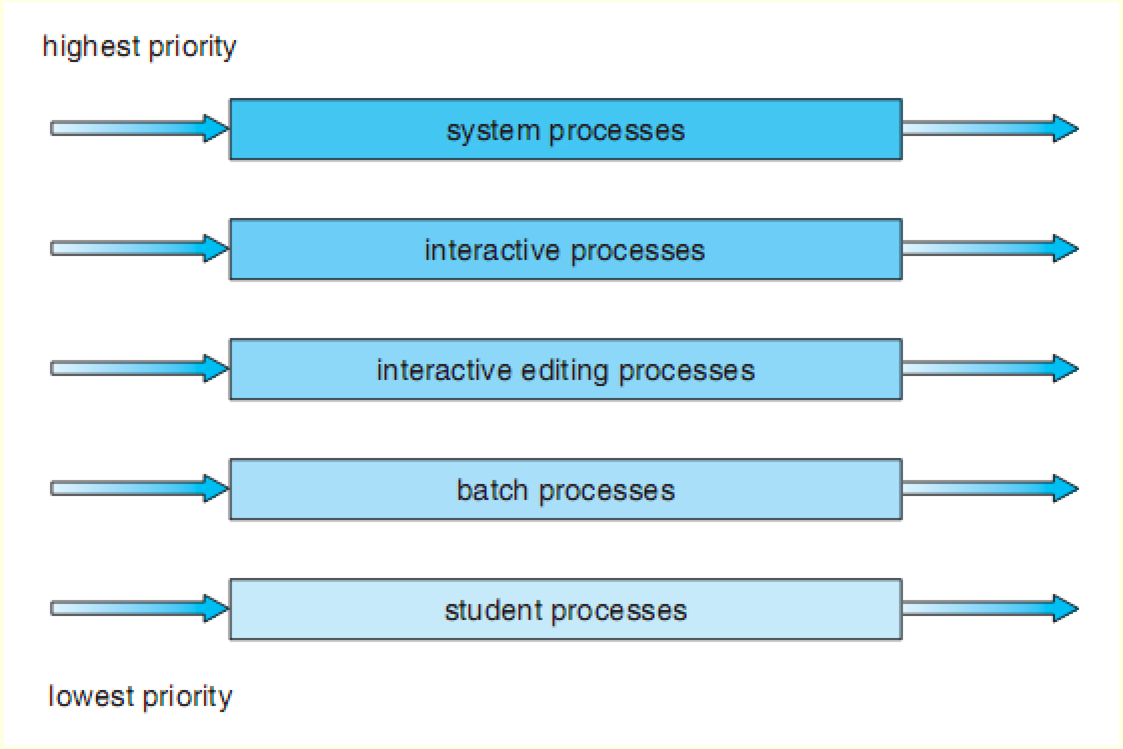
How turnaround time varies with the time quantum?



process	time
P_1	6
P_2	3
P_3	1
P_4	7

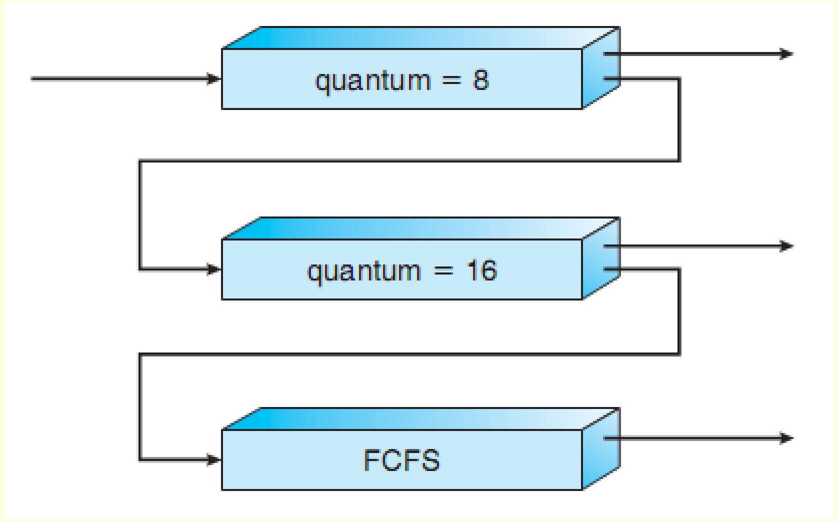
Scheduling Algorithms

- Multilevel Queue Scheduling
  - Absolute priority (starvation)
  - Time-slicing among queues (e.g., 80% for RR among foreground processes and 20% for FCFS serving background processes)



Scheduling Algorithms

- Multilevel Feedback Queue Scheduling



  - Aging can also be applied

Scheduling Algorithms

- Multilevel Feedback Queue Scheduling
  - Parameters
    - Number of queues
    - The scheduling algorithm for each queue
    - How to upgrade the priority of a process
    - How to downgrade the priority of a process
    - Specifying the queue to which a process should be added to get service

Multiple-Processor Scheduling

- Asymmetric multiprocessing
  - Master server does scheduling decisions, I/O processing, …
- Symmetric multiprocessing (SMP)
  - Each processor is self-scheduling
  - Each processor may have its own ready queue or they may share a common ready queue

Symmetric multiprocessing (SMP)

- Processor Affinity
  - Migration
    - → cache invalidating of the source processor
    - → cache re-populating of the target processor
  - Soft affinity
    - OS tries to avoid migration but cannot guarantee
  - Hard affinity
    - OS guarantees that a process will not migrate

Symmetric multiprocessing (SMP)

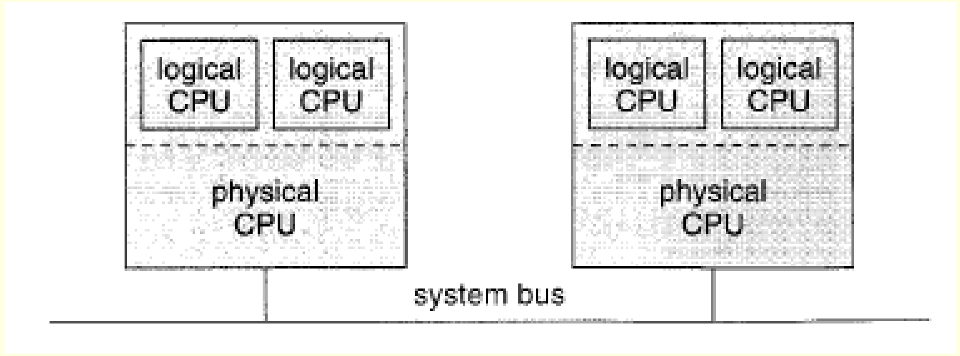
- Load balancing
  - Push migration
    - Checking periodically with a specific task
    - When imbalance is detected, the load of overloaded processors are pushed to less-busy processors
  - Pull migration
    - An idle processor pulls a waiting task from a busy processor

Symmetric Multithreading (SMT)

- SMP runs several threads concurrently on multiple *physical* processors
- SMT runs several threads concurrently on multiple *logical* processors (hyper-threading technology in Intel processors)
  - HW simulates one or more physical processors as a number of logical processors for the OS

Symmetric Multithreading (SMT)

- OS does not need to know if it is running on a SMT
- But, if it is aware of SMT, it may make better decisions
  - E.g., running two threads on two logical CPUs on different physical CPUs rather than one such CPU

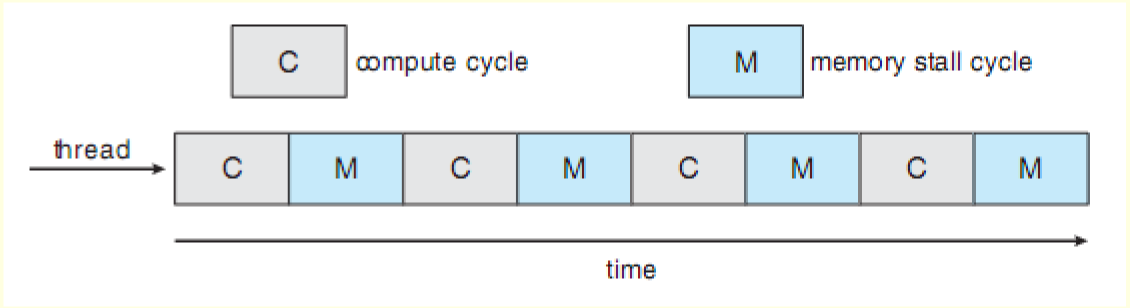


Thread Scheduling

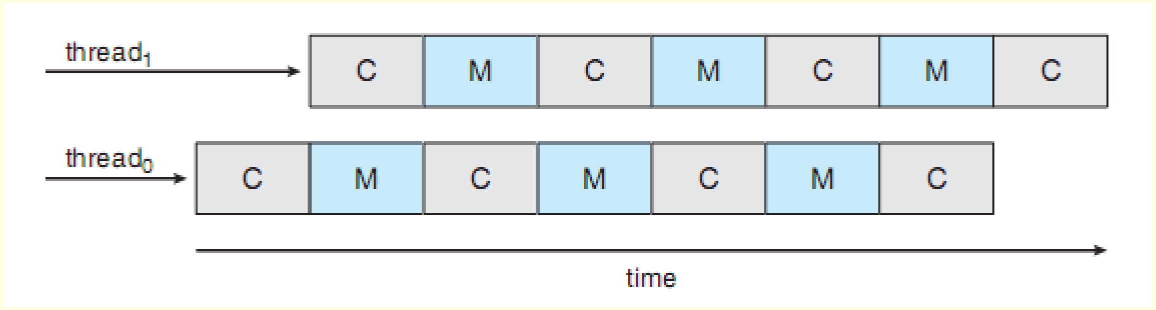
- Process-Contention Scope (PCS):
  - Competition for the CPU among the threads belonging to the same process (thread library)
- System-Contention Scope (SCS):
  - Competition for the CPU among all threads in the system (OS CPU scheduler)

Multicore Processors

- Memory Stall (50% in this example)



- Many recent hardware designs have implemented multithreaded processor cores in which two (or more) hardware threads are assigned to each core
- A dual-threaded processor core on which the execution of thread 0 and the execution of thread 1 are interleaved



Multicore Processors

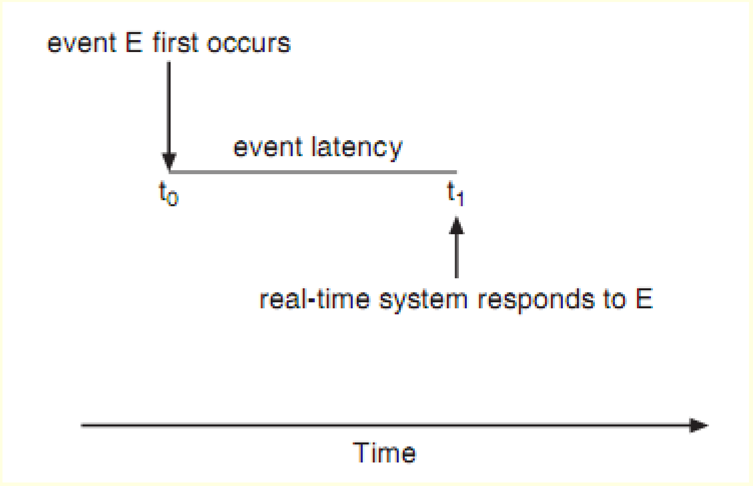
- Ways to multithread a processing core:
  - Coarse-grained:
    - A thread executes on a processor until a long-latency event such as a memory stall occurs.
    - Because of the delay caused by the long-latency event, the processor must switch to another thread to begin execution.
    - However, the cost of switching between threads is high, since the instruction pipeline must be flushed before the other thread can begin execution on the processor core.
  - Fine-grained (or Interleaved)
    - Switching occurs typically at the boundary of an instruction cycle.
    - However, the architectural design of fine-grained systems includes logic for thread switching. As a result, the cost of switching between threads is small.

Multithreaded Multicore Processors

- Two-level scheduling:
  - On one level are the scheduling decisions that must be made by the operating system as it chooses which software thread to run on each hardware thread (logical processor). For this level of scheduling, the operating system may choose any scheduling algorithm.
  - A second level of scheduling specifies how each core decides which hardware thread to run. There are several strategies to adopt in this situation,
    - e.g., in UltraSPARC T3: round-robin algorithm to schedule eight hardware threads to each core.
    - In Intel Itanium (a dual-core processor with two hardware managed threads per core): Assigned to each hardware thread is a dynamic urgency value ranging from 0 to 7. The thread-switching logic compares the urgency of the two threads and selects the thread with the highest urgency value to execute on the processor core.

Real-Time CPU Scheduling

- HRT
- SRT
- Event Latency

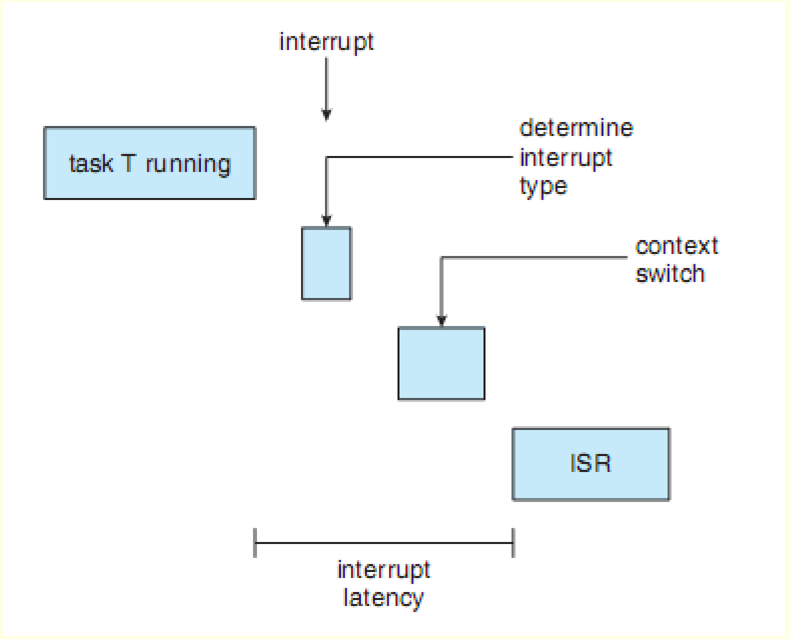


Event Latency

- Interrupt Latency: the period of time from the arrival of an interrupt at the CPU to the start of the routine that services the interrupt.
- When an interrupt occurs, the operating system must:
  - First complete the instruction it is executing and determine the type of interrupt that occurred
  - Second it must then save the state of the current process before servicing the interrupt using the specific interrupt service routine (ISR).

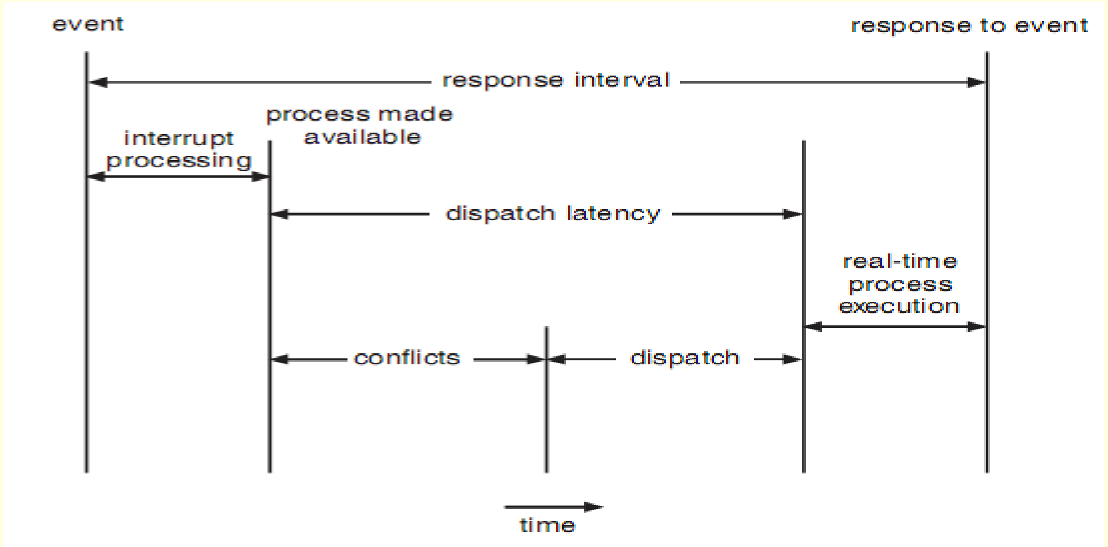
- One important factor contributing to interrupt latency is the amount of time interrupts may be disabled while kernel data structures are being updated.

- Real-time operating systems require that interrupts be disabled for only very short periods of time.



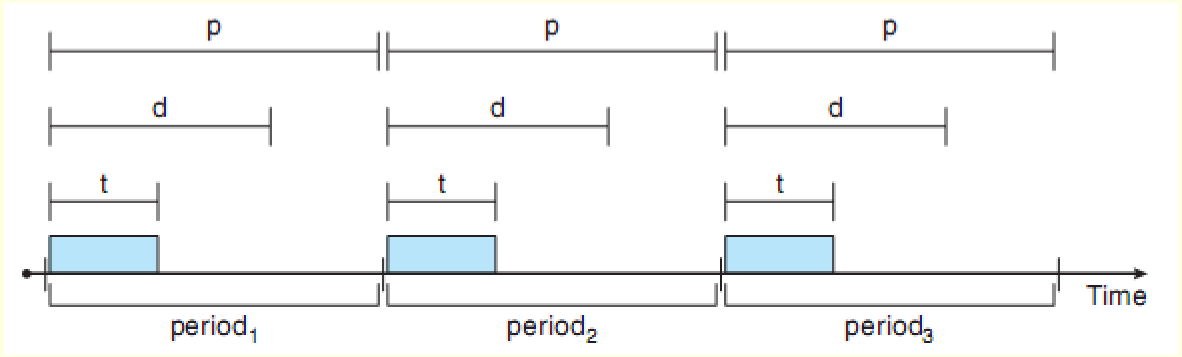
Event Latency

- Dispatch Latency:
  - The amount of time required for the scheduling dispatcher to stop one process and start another
  - The most effective technique for keeping dispatch latency low is to provide preemptive kernels
  - The conflict phase of dispatch latency has two components:
    - Preemption of any process running in the kernel
    - Release by low-priority processes of resources needed by a high-priority process



Priority-Based Scheduling

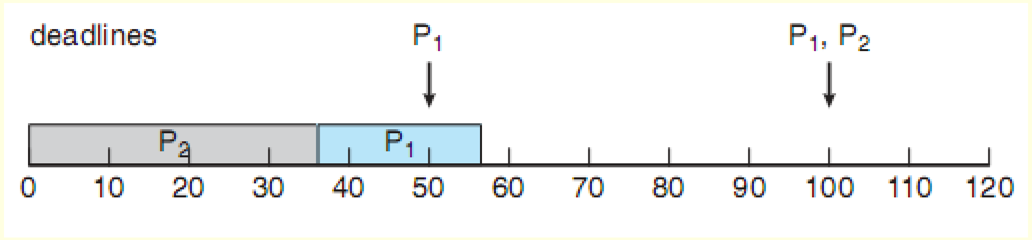
- Periodic Task T:(t,p,d)
- Rate: 1/p



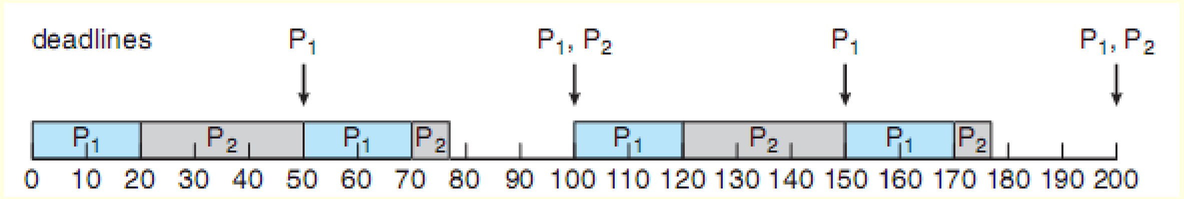
- Admission Control

Rate-Monotonic Scheduling

- RM: Static-Priority Algorithm with Preemption
- Ex: T1: (20,50), T2:(35,100)
  - T2: high priority (deadline miss)



  - RM (U_{RM}=n(2^(1/n)-1)



Earliest-Deadline-First Scheduling

- Dynamic-Priority
- Unlike the rate-monotonic algorithm, EDF scheduling does not require that processes be periodic, nor must a process require a constant amount of CPU time per burst.
- $U_{EDF}=1$
- In practice, however, it is impossible to achieve this level of CPU utilization due to the cost of context switching between processes and interrupt handling.
- RM vs. EDF$\rightarrow$T1:(25,50), T2:(35,80)

Proportional Share Scheduling

- Reservation
- Need to Admission-Control