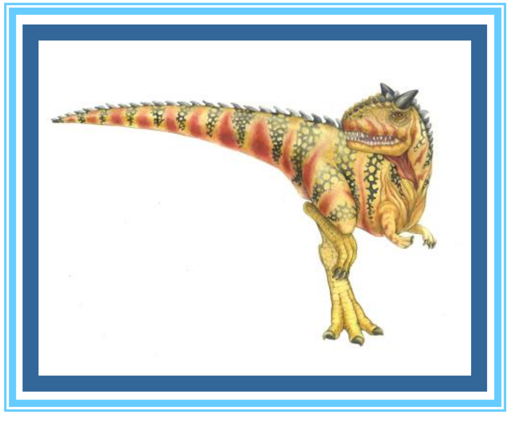
CPU Scheduling



Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples
- Algorithm Evaluation

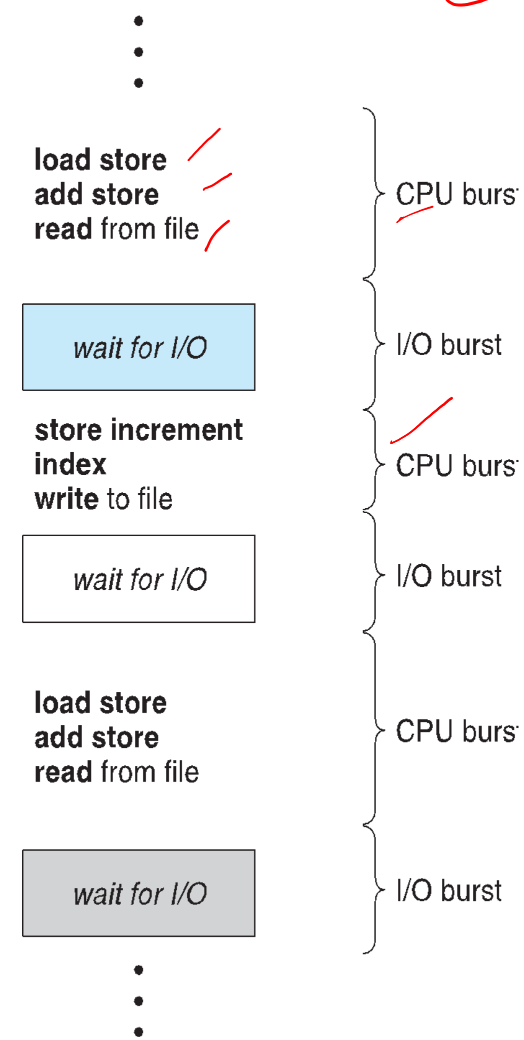
Objectives

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

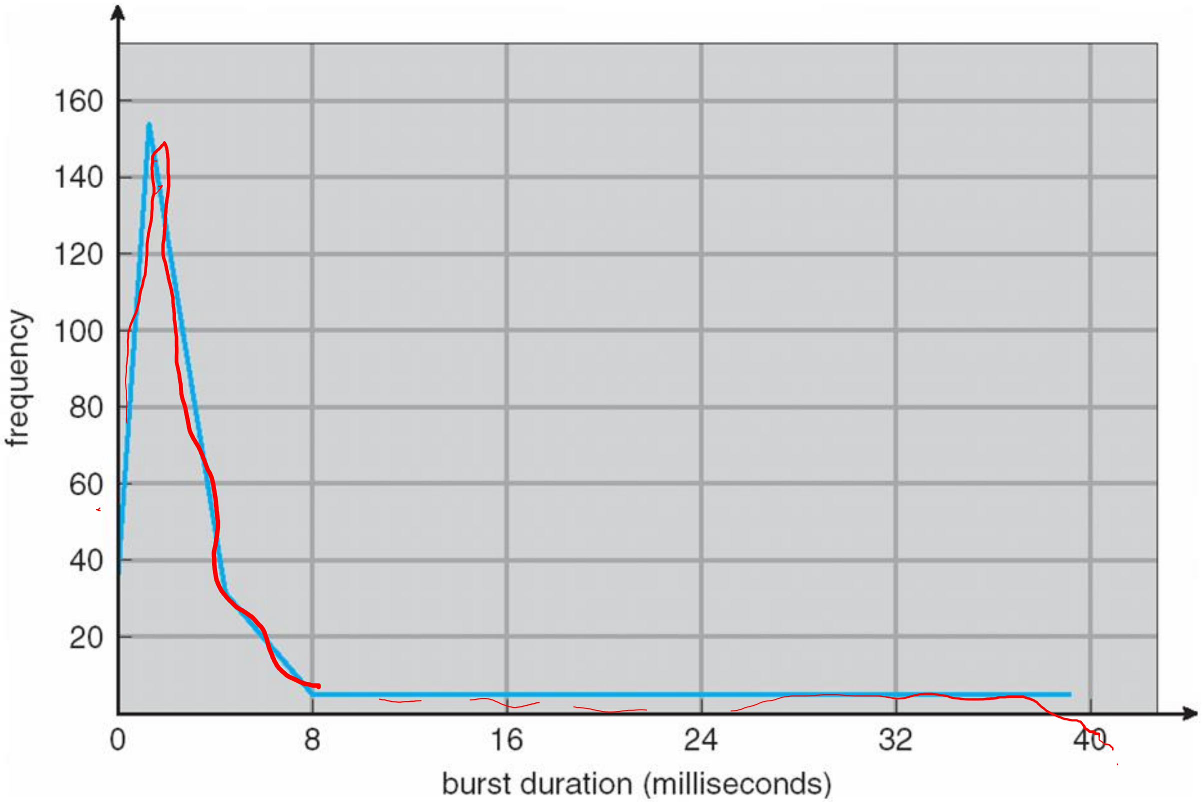
Basic Concepts

cycle
cpu burst
I/O burst

- 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



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
- 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 ready
  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

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

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)

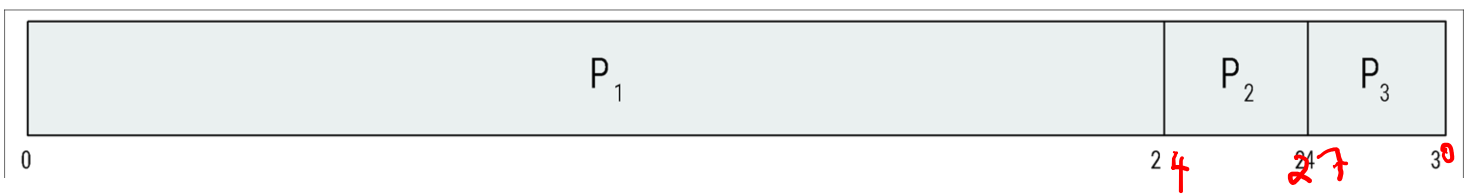
Scheduling Algorithm Optimization Criteria

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

First- Come, First-Served (FCFS) Scheduling

Process	Burst Time	TT
P_1	24	24
P_2	3	27
P_3	3	30

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  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 (Cont.)

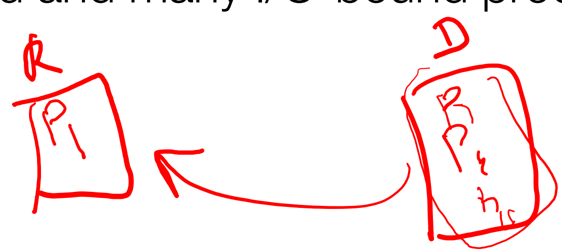
Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:

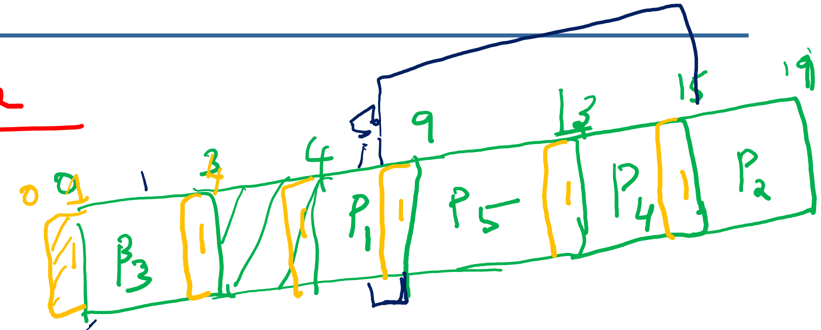
- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- Convoy effect - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

P1 - 100
P2 - 1
P3 - 1
P4 - 1



Non - Preemptive

Process id	Arrival time	Burst Time
P_1	4	5
P_2	6	4
P_3	0	3
P_4	6	2
P_5	5	4



Turnaround = Completion time − Arrival.Time

Waiting = Turn.Time − CPU.Burst

Turnaround Time

P_1 - 9 - 4 = 5

P_2 - 19 - 6 = 13

P_3 = 3

P_4 = 9

P_5 = 8

7.6 ≈ 8

Waiting time

P_1 = 0

P_2 = 15 − 6

P_3 = 0

P_4 = 13 − 6

P_5 = 9 − 5

4.4

If FCFS scheduling is followed and there is 1 unit of overhead scheduling the process find the efficiency of the algorithm

Shortest-Job-First (SJF) Scheduling

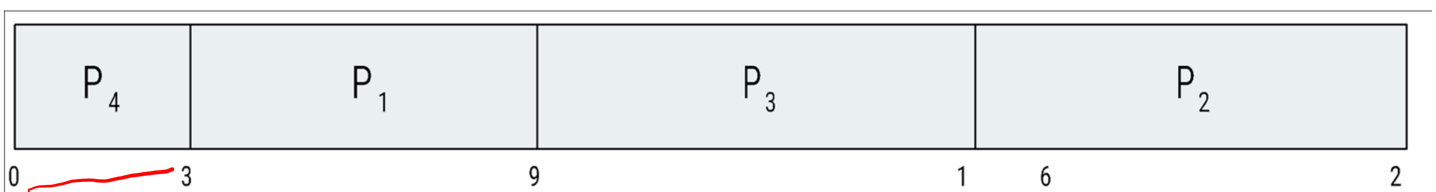
- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

  Shortest next CPU burst

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

Example of Shortest-remaining-time-first

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

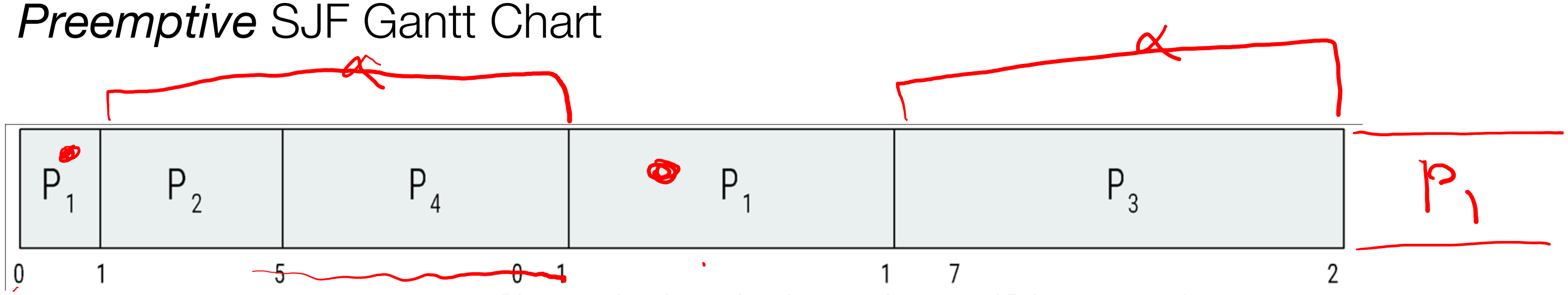
Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

7

3 2 X

9

- *Preemptive* SJF Gantt Chart



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

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

$\alpha = 1$

Priority Scheduling

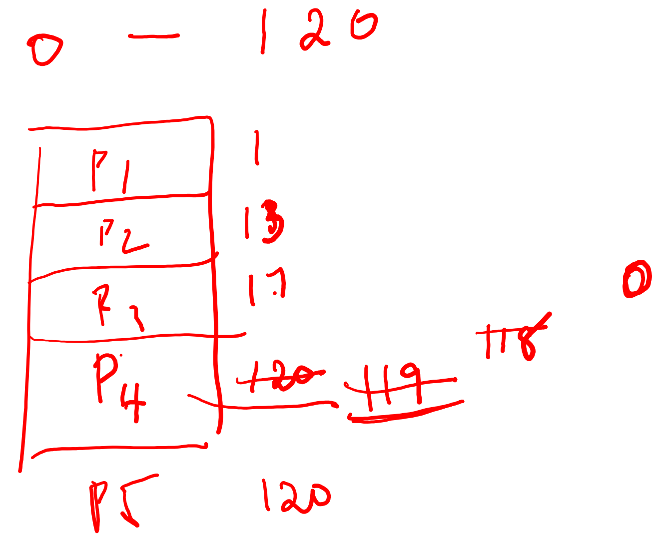
internal

external

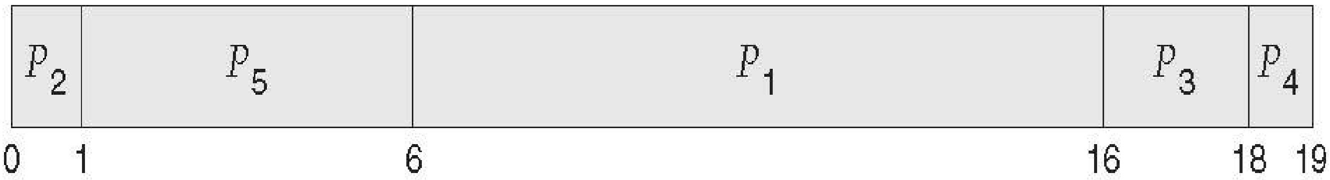
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- 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



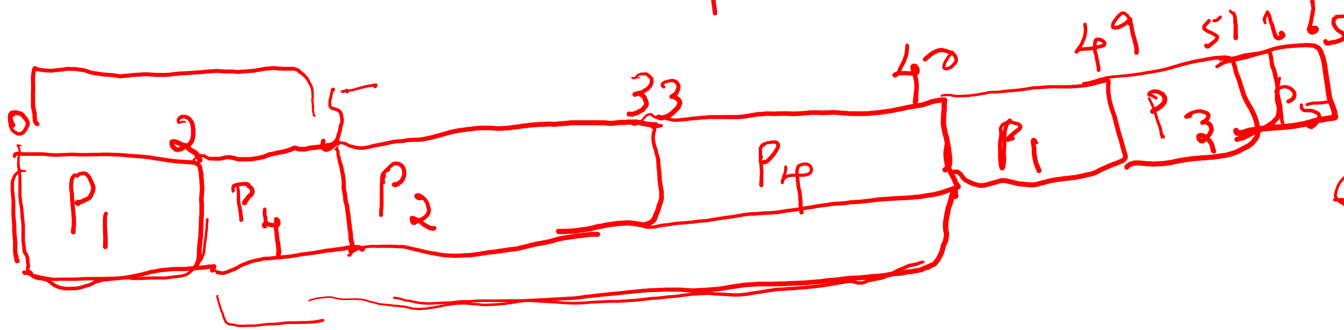
- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

Preemptive – Priority Scheduling

Process	Arrival time	CPU	Priority	Remaining
P_1	0	11	2 ✓	9
P_2	5	28	0 ✗	
P_3	12	2	3	
P_4	2	10	1	7
P_5	9	16	4	



~~P_1 – 2~~

~~P_4 – 1~~

~~P_2 – 0~~

P_5 – 4

~~P_3 – 3~~

Round Robin (RR)

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

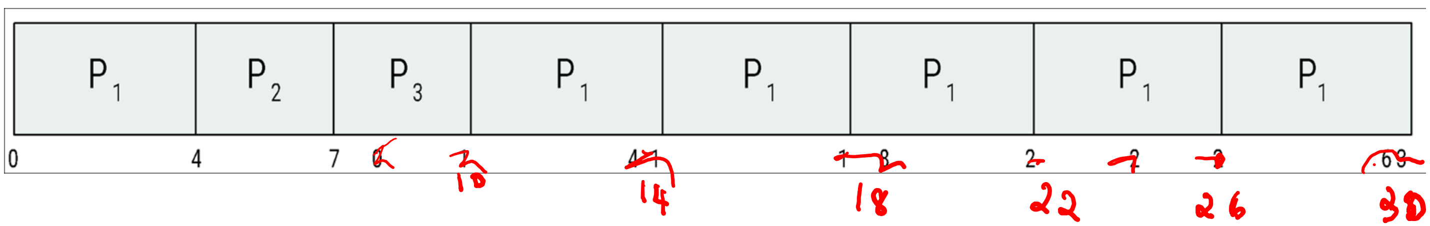
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

4

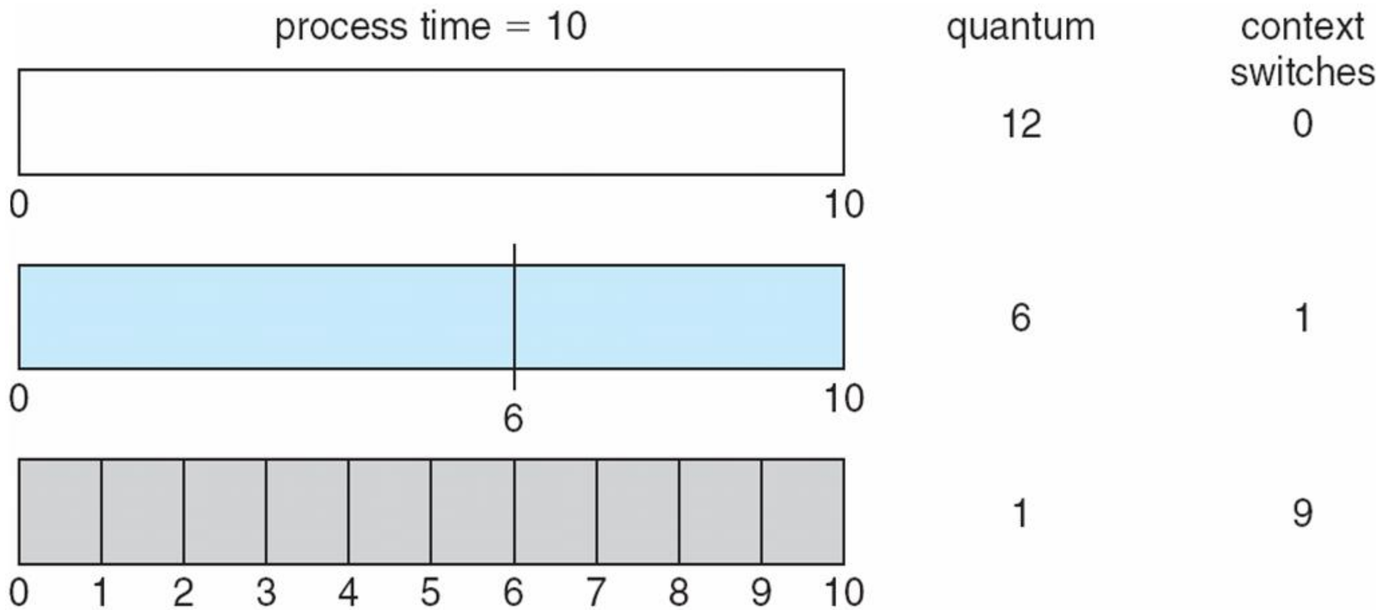
4

- The Gantt chart is:



- 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 < 10 usec

Time Quantum and Context Switch Time



RR

Time quantum = 2 units

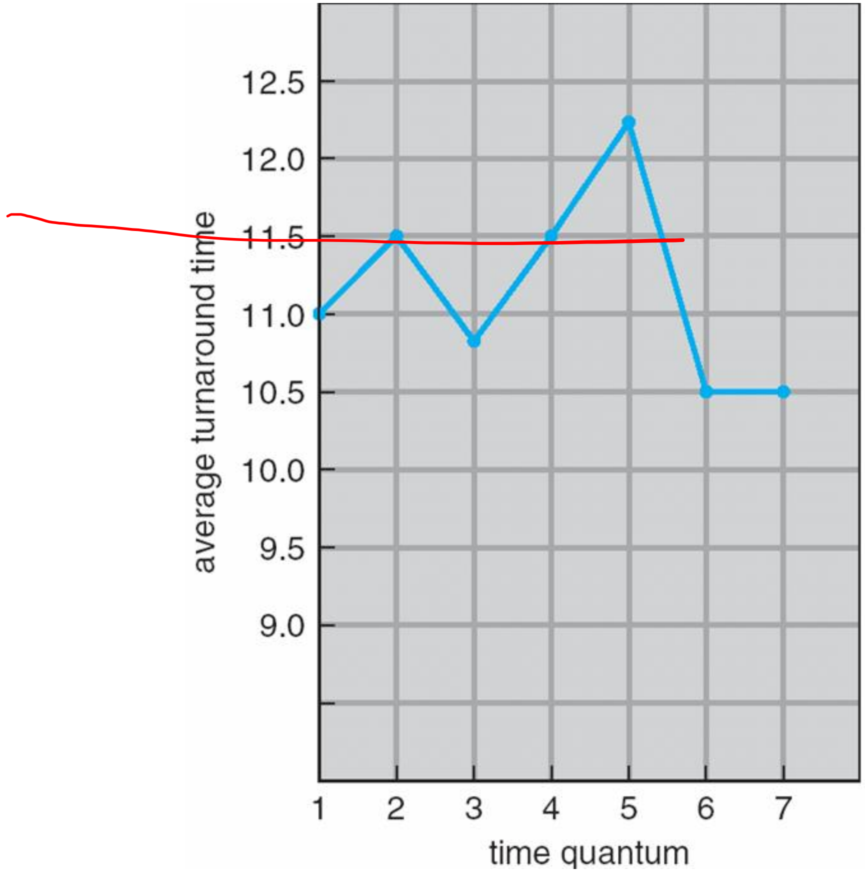
Process	Arrival	Burst time	Remaining
P_1	0	5	3 – 1
P_2	1	3	1 – 0
P_3	2	1	0
P_4	3	2	0
P_5	4	3	– 1

Ready Q.

P_5 P_1 P_2 P_5 P_4 P_1 P_3 P_2 P_1

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_1	P_5
0–2	2–4	4–5	5–7	7–9	9–11	11–12	12–13	13–14

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

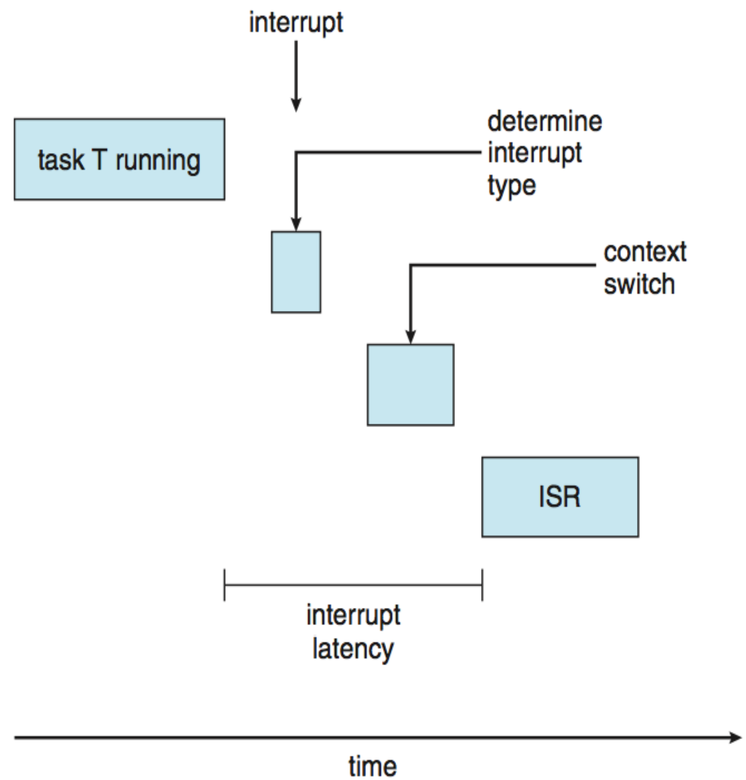
Thread Scheduling

- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- 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

Real-Time CPU Scheduling

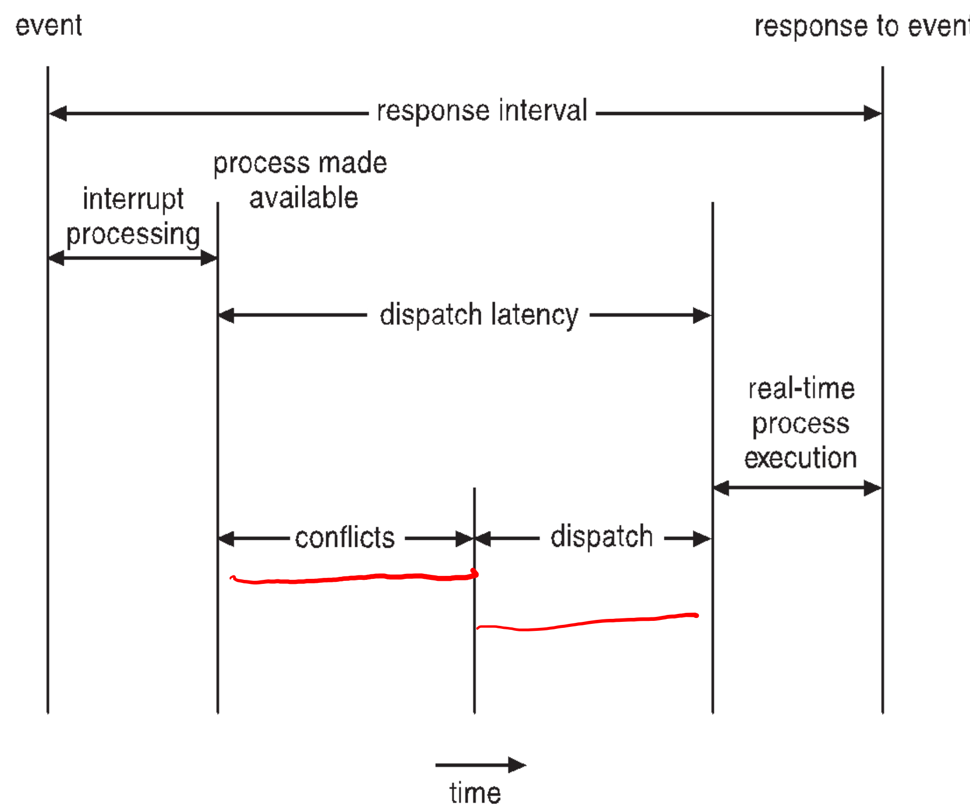
Event latency

- Can present obvious challenges
- **Soft real-time systems** – no guarantee as to when critical real-time process will be scheduled
- **Hard real-time systems** – task must be serviced by its deadline
- Two types of latencies affect performance
  1. Interrupt latency – time from arrival of interrupt to start of routine that services interrupt
  2. Dispatch latency – time for schedule to take current process off CPU and switch to another



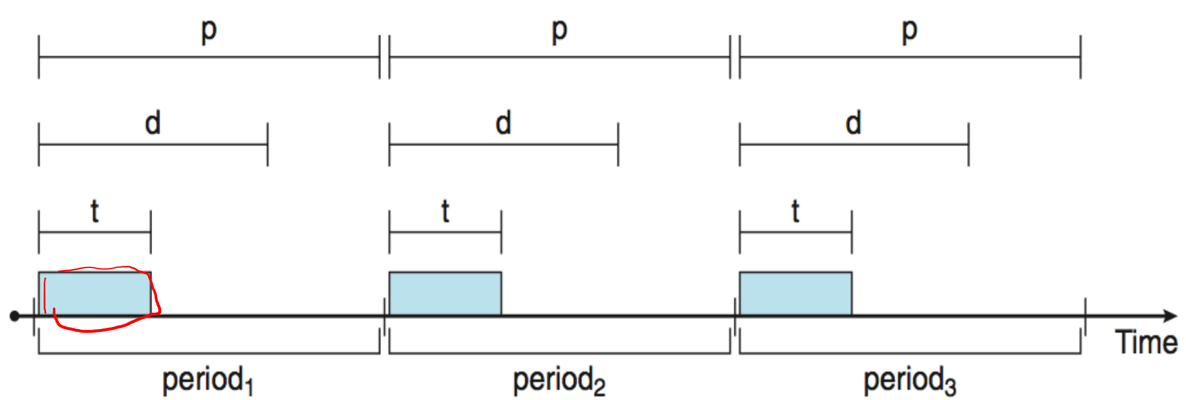
Real-Time CPU Scheduling (Cont.)

- Conflict phase of dispatch latency:
  1. Preemption of any process running in kernel mode
  2. Release by low-priority process of resources needed by high-priority processes



Priority-based Scheduling

- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For hard real-time must also provide ability to meet deadlines
- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time *t*, deadline *d,* period *p*
  - $0 \le t \le d \le p$
  - **Rate** of periodic task is $1/p$

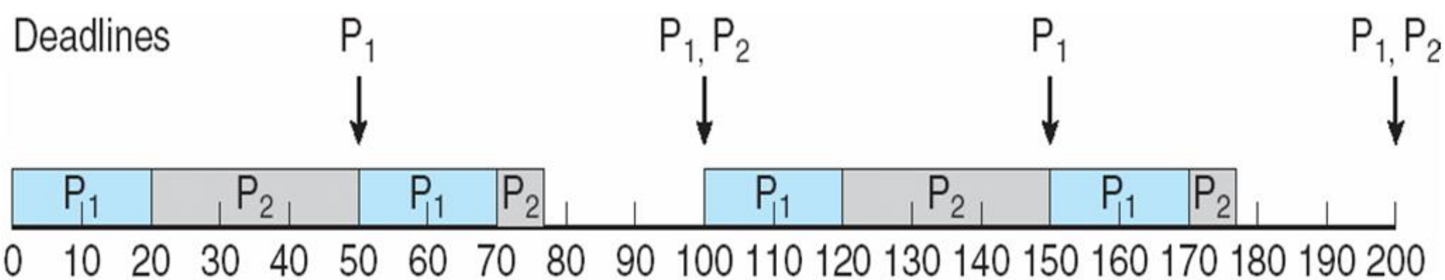


Virtualization and Scheduling

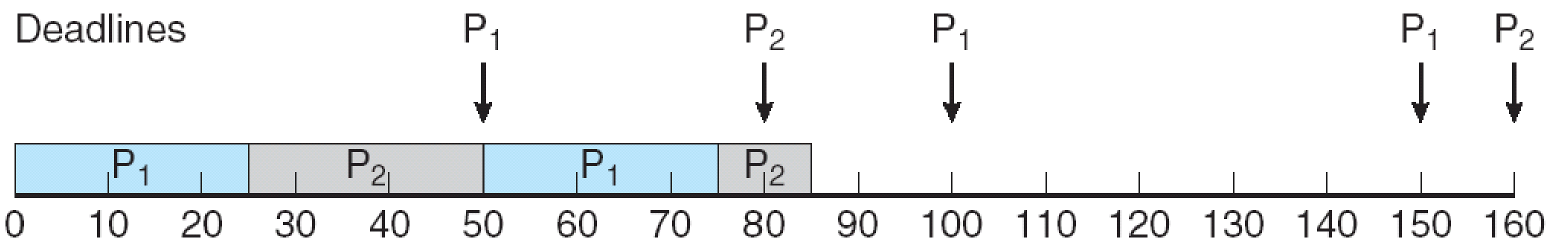
- Virtualization software schedules multiple guests onto CPU(s)
- Each guest doing its own scheduling
  - Not knowing it doesn't own the CPUs
  - Can result in poor response time
  - Can effect time-of-day clocks in guests
- Can undo good scheduling algorithm efforts of guests

Rate Montonic Scheduling

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2.



Missed Deadlines with Rate Monotonic Scheduling

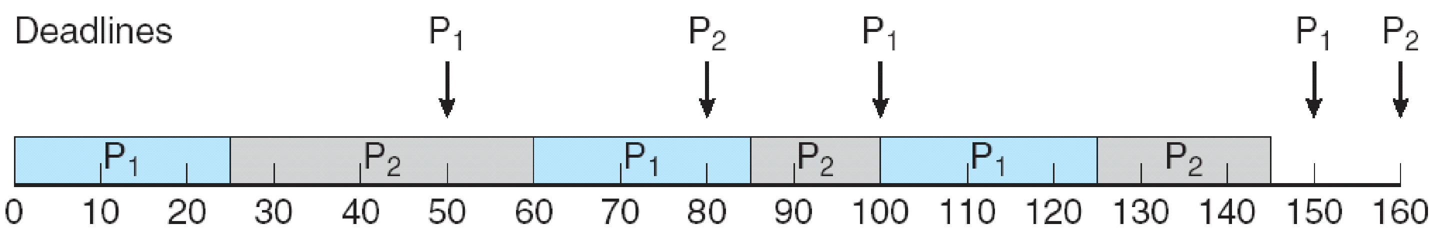


Earliest Deadline First Scheduling (EDF)

- Priorities are assigned according to deadlines:

  the earlier the deadline, the higher the priority;

  the later the deadline, the lower the priority



Proportional Share Scheduling

- *T* shares are allocated among all processes in the system
- An application receives *N* shares where *N* < *T*
- This ensures each application will receive ***N*** / *T* of the total processor time