CPU Scheduling

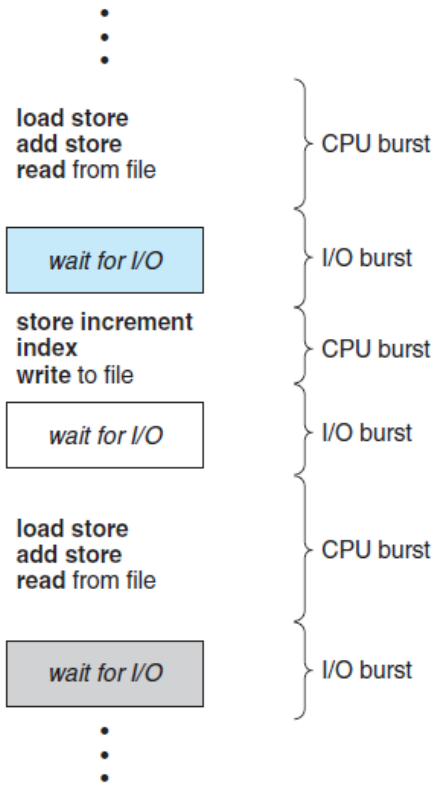
Objectives

- To introduce CPU scheduling, which is the basis for multi-programmed 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.

Why?

- More Productive
- **Basic Concept:**
  - typically for the completion of some I/O request- the CPU then just sits idle.
  - With multiprogramming, we try to use this time productively. Several processes are kept in memory at one time.

CPU–I/O Burst Cycle



- process execution consists of a **cycle** of CPU execution and I/O wait.
- Processes alternate between these two states. Process execution begins with a **CPU burst**.
- That is followed by an **I/O burst**, which is followed by another CPU burst, then

  another I/O burst, and so on.
- An **I/O-bound program** typically has many short CPU bursts. A **CPU-bound**

  program might have a few long CPU bursts.

CPU Scheduler

1. short-term scheduler- operating system must select one of the processes in the ready queue to be executed.

- The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.
- A ready queue can be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list.
- The records in the queues are generally process control blocks (PCBs) of the processes.

Preemptive Scheduling

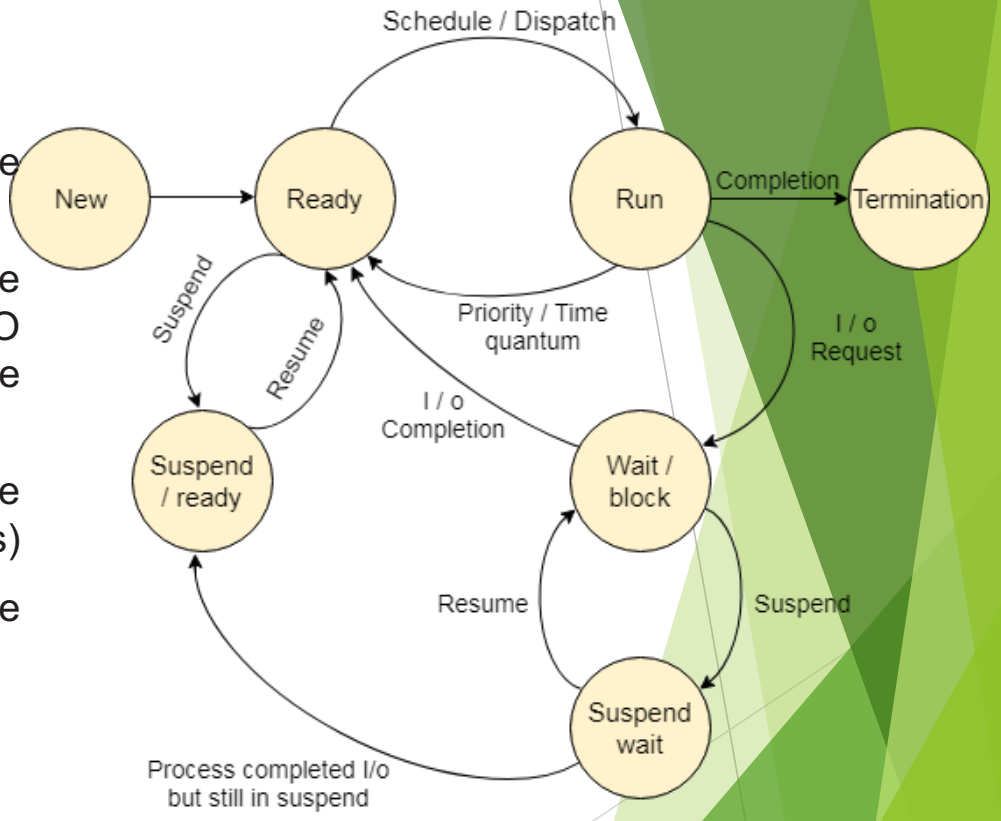
CPU-scheduling decisions may take place under the following four circumstances:

When a process switches from the running state to the waiting state (for example, as the result of an I/O request or an invocation of wait() for the termination of a child process)

When a process switches from the running state to the ready state (for example, when an interrupt occurs)

When a process switches from the waiting state to the ready state (for example, at completion of I/O)

When a process terminates



- When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is **nonpreemptive** or **cooperative**.
- Otherwise in situation 2 or 3, the scheduling is **preemptive**.

2. Dispatcher

- The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves the following:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

- The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**.

Scheduling Criteria

- **Throughput-** measure of work is the number of processes that are completed per time unit.
- **Turnaround time**- The interval from the time of submission of a process to the time of completion is the turnaround time.
- **Waiting time**- Waiting time is the sum of the periods spent waiting in the ready queue.
- **Response time- T**he time from the submission of a request until the first response is produced.
- **CPU utilization**- it should be high.

- It is desirable to maximize CPU utilization and throughput and to minimize turnaround time, waiting time, and response time.

Scheduling Algorithms

1. First-come, first-served (FCFS) scheduling algorithm-

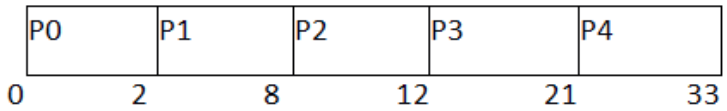
- FIFO queue
- Average waiting time under the FCFS policy is often quite long.

Gantt Chart

Turn Around Time = Completion Time - Arrival Time
Waiting Time = Turnaround time - Burst Time

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
0	0	2	2	2	0
1	1	6	8	7	1
2	2	4	12	10	6
3	3	9	21	18	9
4	6	12	33	27	15 7

Avg Waiting Time=31/5



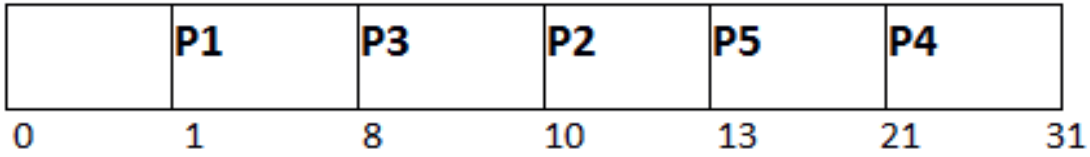
Disadvantage

- The scheduling method is non preemptive, the process will run to the completion.
- Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compared to other scheduling algorithms.
- troublesome for time-sharing systems
- convoy effect

All the other processes wait for the one big process to get off the CPU. This effect results in lower CPU and device utilization than might be possible if the shorter processes were allowed to go first.

Shortest-Job-First Scheduling

PID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	1	7	8	7	0
2	3	3	13	10	7
3	6	2	10	4	2
4	7	10	31	24	14
5	9	8	21	12	4



Prediction of CPU Burst Time

- predict the burst time of a new process according to the burst time of an old process of similar size as of new process
- Exponential Averaging or Aging

Tn be the actual burst time of nth process. τ(n) be the predicted burst time for nth process then the CPU burst time for the next process (n+1) will be calculated as,

$$\tau(n+1) = \alpha . Tn + (1-\alpha) . \tau(n)$$

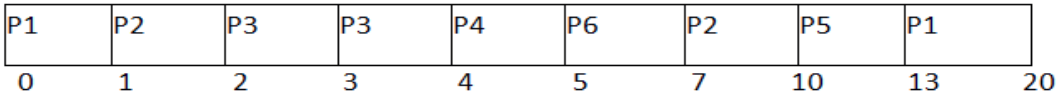
Where, α is the smoothing. Its value lies between 0 and 1.

Shortest Remaining Time First (SRTF) Scheduling Algorithm-preemptive version of SJF scheduling

- preemptive version of SJF scheduling

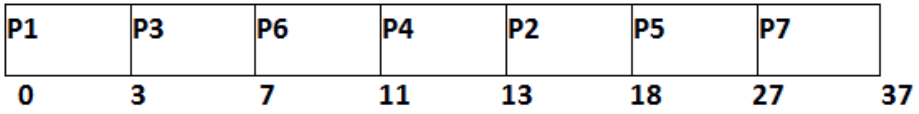
Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	8	20	20	12
2	1	4	10	9	5
3	2	2	4	2	0
4	3	1	5	2	1
5	4	3	13	9	6
6	5	2	7	2	0

Avg Waiting Time = 24/6



Priority Scheduling

Process Id	Priority	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
1	2	0	3	3	3	0
2	6	2	5	18	16	11
3	3	1	4	7	6	2
4	5	4	2	13	9	7
5	7	6	9	27	21	12
6	4	5	4	11	6	2
7	10	7	10	37	30	18



Avg Waiting Time = 52/7 units

Disadvantage

- A major problem with priority scheduling algorithms is **indefinite blocking**, or **starvation**.
- A solution to the problem of indefinite blockage of low-priority processes is **Aging**. Aging involves gradually increasing the priority of processes that wait in the system for a long time.

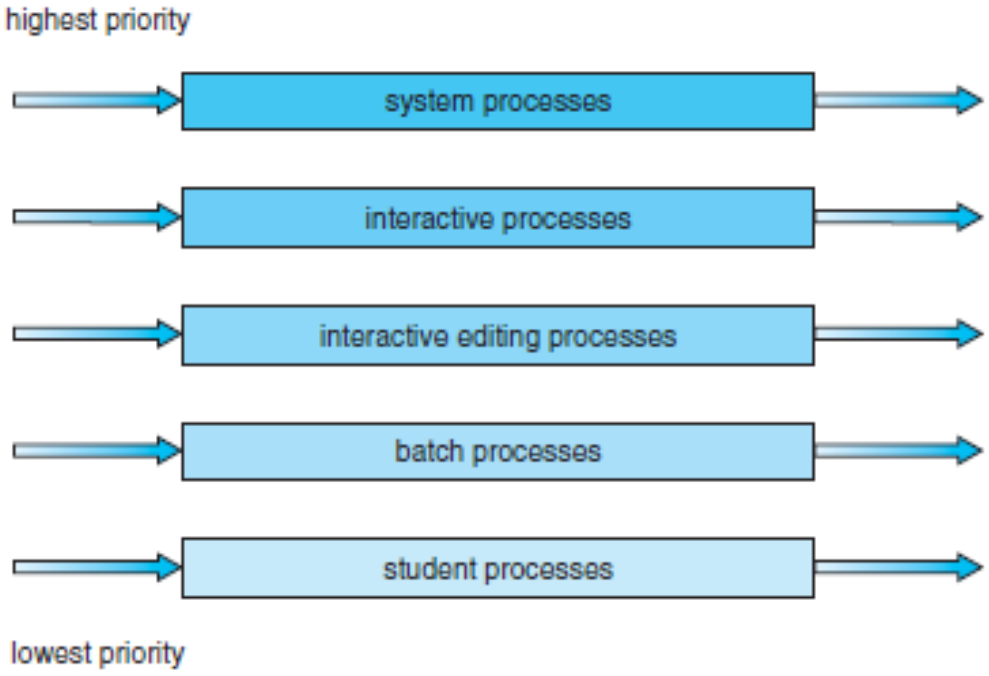
Round-Robin Scheduling

- The time quantum is generally from10 to 100 milliseconds in length. The ready queue is treated as a circular queue.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	5	17	17	12
2	1	6	23	22	16
3	2	3	11	9	6
4	3	1	12	9	8
5	4	5	24	20	15
6	6	4	21	15	11

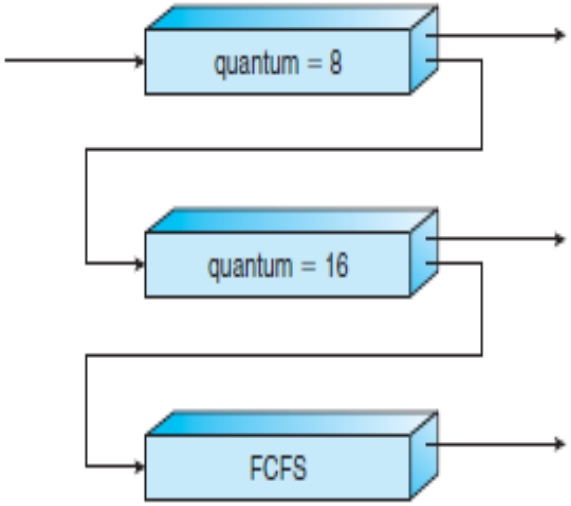
Multilevel Queue Scheduling

- **foreground** (interactive) processes and **background** (batch) processes
- These two types of processes have different response-time requirements and so may have different scheduling needs.
- This algorithm partitions the ready queue into several separate queues.
- The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.
- There must be scheduling among the queues-fixed-priority pre-emptive scheduling.


highest priority
system processes
interactive processes
interactive editing processes
batch processes
student processes
lowest priority

Multilevel Feedback Queue Scheduling

- Allows a process to move between queues.
- to separate processes according to the characteristics of their CPU bursts.
- a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.



Thread Scheduling

- ***user-level*** and ***kernel-level*** threads
- To run on a CPU, user-level threads must ultimately be mapped to an associated kernel-level thread, and may use a lightweight process (LWP).
- **Contention Scope**
- **Process contention scope (PCS)**- On systems implementing the many-to-one and many-to-many models, the thread library schedules user-level threads to run on an available LWP.
- PCS is done according to priority.
- User-level thread priorities are set by the programmer
- **system-contention scope (SCS)**- Competition for the CPU with SCS scheduling takes place among all threads in the system.
- One-to-one model

Multiple-Processor Scheduling

- If multiple CPUs are available, load sharing becomes possible—but scheduling problems become correspondingly more complex.
- limitations - system with an I/O device attached to a private bus of one processor. Processes that wish to use that device must be scheduled to run on that processor.

Approaches to Multiple-Processor Scheduling

asymmetric multiprocessing-

- all scheduling decisions, I/O processing, and other system activities handled by a single processor —**the master server.** The **other processors** execute only user code.

symmetric multiprocessing (SMP)-

- Each processor is self-scheduling
- All processes may be in a common ready queue, or each processor may have its own private queue of ready processes.
- Virtually all modern operating systems support SMP, including Windows, Linux, and Mac OS X.

1. Processor Affinity

- a process has an affinity for the processor on which it is currently running.
- E.g.-cache memory of earlier processor-migration to another processor
- **soft affinity-** the operating system will attempt to keep a process on a single processor.
- **hard affinity-** allowing a process to specify a subset of processors on which it may run.
- Linux implements soft affinity, but it also provides the sched setaffinity() system call, which supports hard affinity.

2. Load Balancing

- The workload evenly distributed across all processors in an SMP system.
- load balancing is typically necessary only on systems where each processor has its own private queue
- Common queue- it become unnecessary
- But in OS- each processor has its own private queue
- There are two general approaches to load balancing:

push migration

- a specific task periodically checks the load on each processor- evenly distributes the load by moving (or pushing) processes from overloaded to idle or less-busy processors.

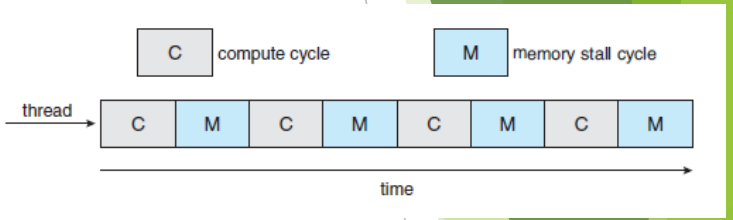
pull migration

- when an idle processor pulls a waiting task from a busy processor.

- load balancing often counteracts the benefits of processor affinity.
- Either pulling or pushing a process from one processor to another removes the benefit of using the its data being in previous processor's cache memory.
- Thus, in some systems, an idle processor always pulls a process from a non-idle processor. In other systems, processes are moved only if the imbalance exceeds a certain threshold.

Multicore Processors

- Researchers have discovered that when a processor accesses memory, it spends a significant amount of time waiting for the data to become available-**Memory Stall**
- remedy - multithreaded processor cores in which two (or more) hardware threads are assigned to each core.
- multithread a processing core:
- **coarse grained –**
- a thread executes on a processor until a long-latency event such as a memory stall occurs. 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 –**

switches between threads at a much finer level of granularity—typically at the boundary of an instruction cycle.

- **one level are the scheduling decisions –**

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.

Real-Time CPU Scheduling

- Two types:
- Soft real time system :

They guarantee only that the process will be given preference over noncritical processes.

- Hard real time system :

-stricter requirements.

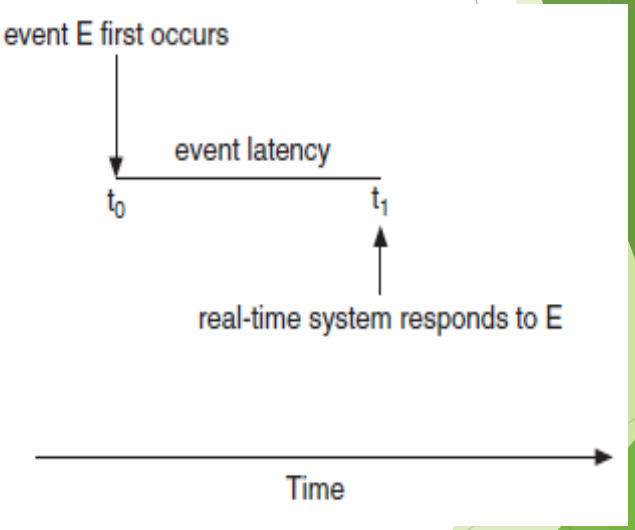
- A task must be serviced by its deadline; service after the deadline has expired is the same as no service at all.

Minimizing Latency

- Events may arise either in software —as when a timer expires—or in hardware—as when a remote-controlled vehicle detects that it is approaching an obstruction

Event Latency

the amount of time that elapses from when an event occurs to when it is serviced.

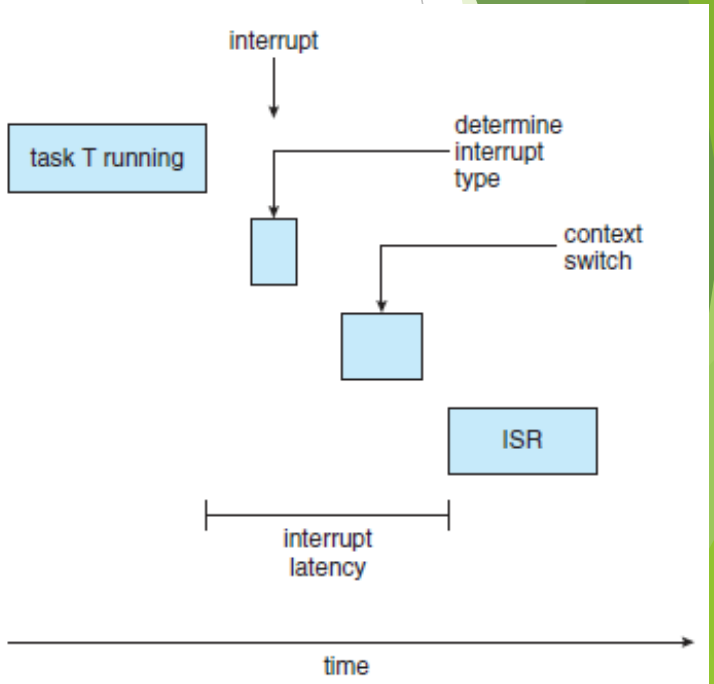


Types of latencies

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.

1. The operating system must first complete the instruction
2. determine the type of interrupt that occurred.
3. It must the save the state of the current process before servicing the interrupt using the specific interrupt service routine (ISR)
4. Soft real time system and hard real time system

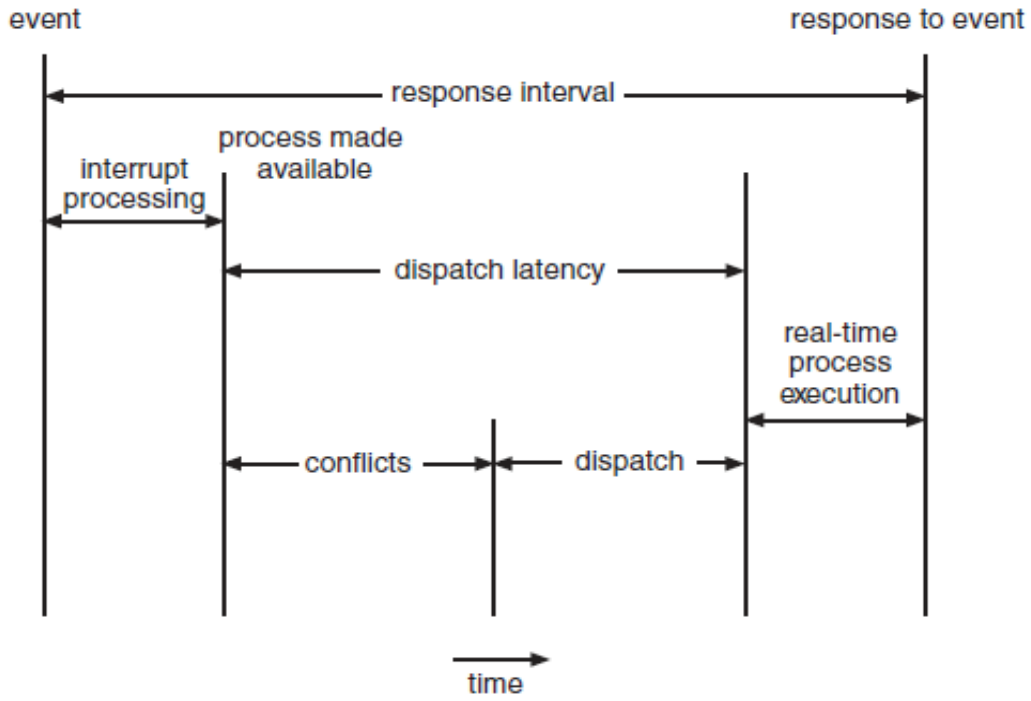


Dispatch Latency

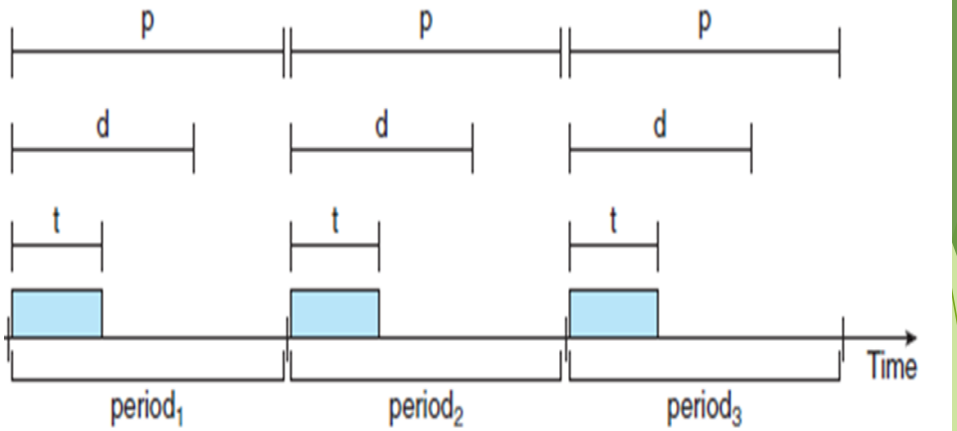
- The amount of time required for the scheduling dispatcher to stop one process and start another.
- The conflict phase of dispatch latency has two components:

1. Preemption of any process running in the kernel

2. Release by low-priority processes of resources needed by a high-priority process



Priority-Based Scheduling



- System should respond immediately to a real-time process as soon as that process requires the CPU.
- Thus, the scheduler for a real-time operating system must support a priority-based algorithm with preemption.
- only guarantees soft real-time functionality.
- Windows has 32 different priority levels. The highest levels—priority values 16 to 31—are reserved for real-time processes. Solaris and Linux have similar prioritization schemes.

Admission-control algorithm

- It either admits the process, guaranteeing that the process will complete on time, or rejects the request as impossible if it cannot guarantee that the task will be serviced by its deadline.
- Characteristics of process:

- periodic in nature

Once a periodic process has acquired the CPU, it has a

fixed processing time **t**,

a deadline **d** by which it must be serviced by the CPU, and

a period **p**.

$$0 \leq t \leq d \leq p$$

Rate-Monotonic Scheduling

Case 1

- Process P1

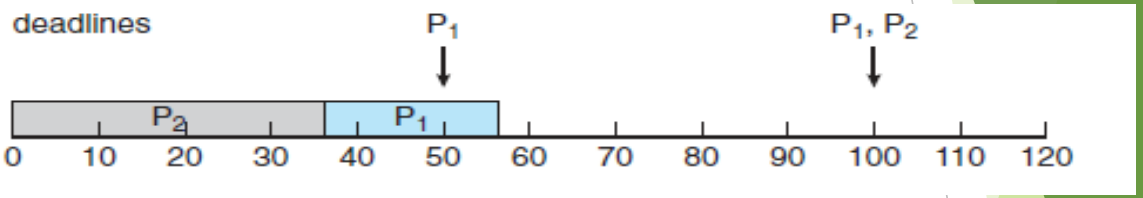
p1 = 50

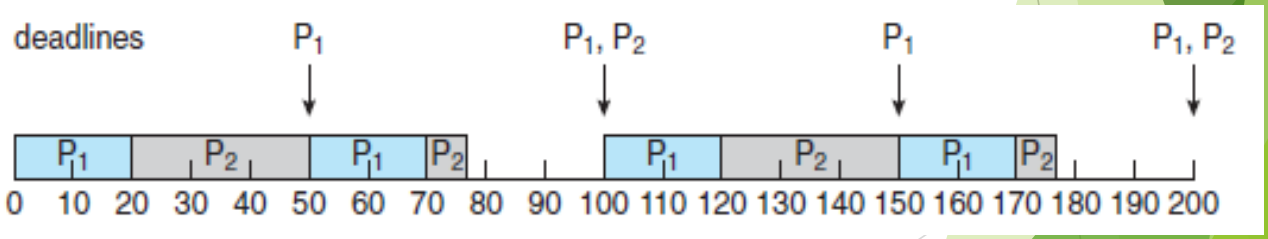
t1 = 20

- Process P2

p2 = 100

t2 = 35





CPU Utilization:

(20/50)+(35/100)=0.75

Case 2

- Process P1

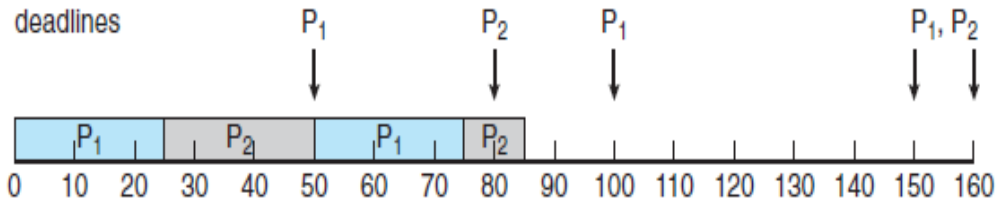
p1 = 50

t1 = 25

- Process P2

p2 = 80

t2 = 35



CPU Utilization:

(25/50)+(35/80)=0.94

- Despite being optimal, then, rate-monotonic scheduling has a limitation.
- CPU utilization is bounded, and it is not always possible fully to maximize CPU resources.
- The worst-case CPU utilization for scheduling N processes is:

$$N(2^{1/N} - 1)$$

Earliest-Deadline-First Scheduling

- Earliest-deadline-first (EDF) scheduling dynamically assigns priorities according to deadline.
- When a process becomes runnable, it must announce its deadline requirements to the system.

- Process P1
  - p1 = 50
  - t1 = 25
- Process P2
  - p2 = 80
  - t2 = 35

CPU Utilization:

(25/50)+(35/80)=0.94