OPERATING SYSTEMS

Multiprocessor and Real Time Scheduling

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Scheduling Criteria

Topics to be covered	% of Portions covered	
	Reference chapter	Cumulative
Introduction: What Operating Systems Do, Computer-System Organization	1.1, 1.2	21.4
Computer-System Architecture, Operating-System Structure & Operations	1.3,1.4,1.5	
Kernel Data Structures, Computing Environments	1.10, 1.11	
Operating-System Services, Operating-System Design and Implementation	2.1, 2.6	
Process concept: Process in memory, Process State, Process Control Block, Context switch, Process Creation & Termination,	3.1 – 3.3	
CPU Scheduling - Preemptive and Non-Preemptive, Scheduling Criteria, FIFO Algrorithm	5.1-5.2	
Scheduling Algorithms:SJF, Round-Robin and Priority Scheduling	5.3	
Multi-Level Queue, Multi-Level Feedback Queue	5.3	
Multiprocessor and Real Time Scheduling	5.5, 5.6	
Case Study: Linux/ Windows Scheduling Policies.	5.7	
Inter Process Communication – Shared Memory, Messages	3.4	
Named and unnamed pipes (+Review)	3.6.3	

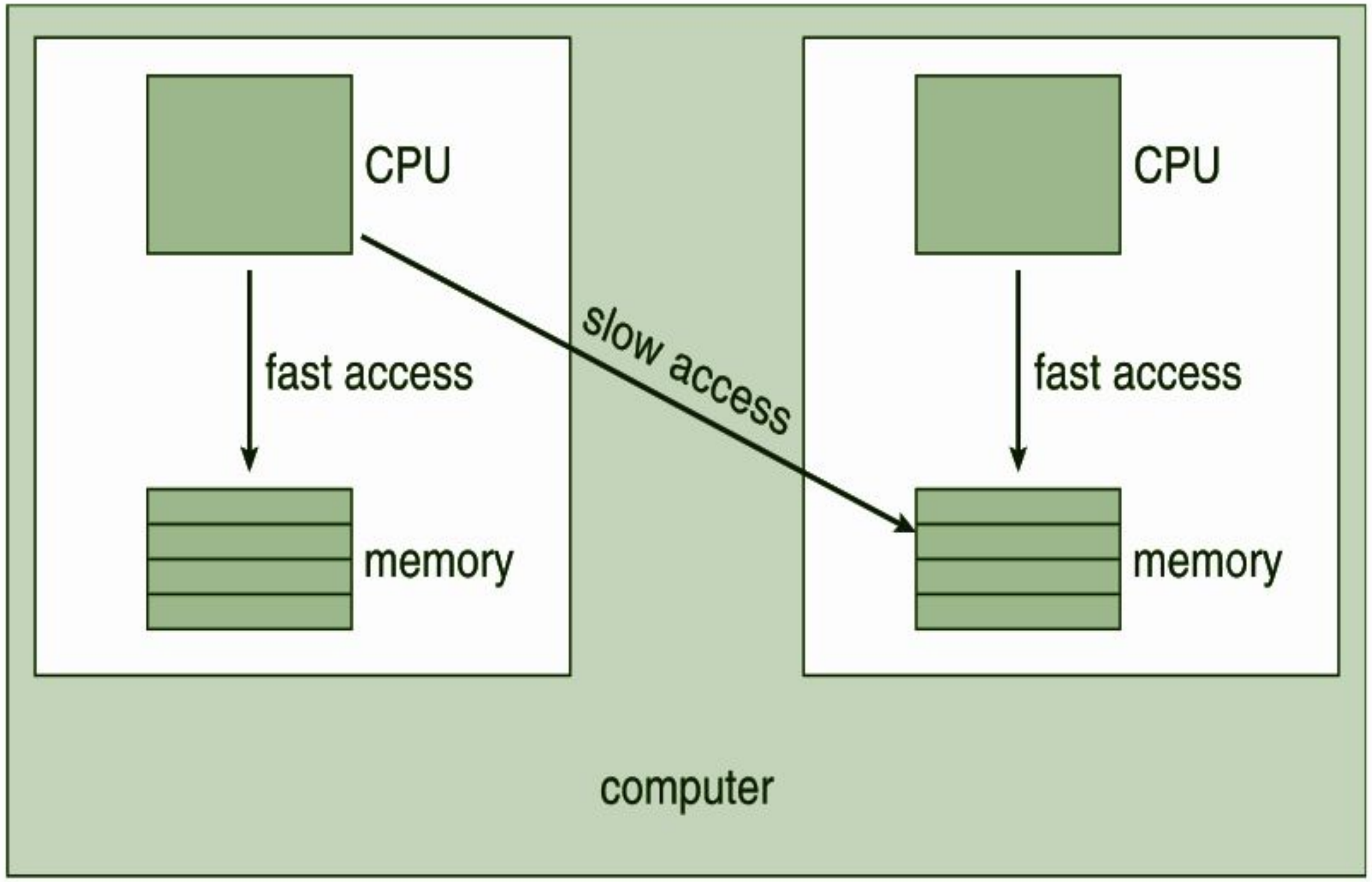
Topic Outline

- Multiprocessor Scheduling
- Real-time Scheduling
  - Priority Based Scheduling
  - Rate Monotonic Scheduling
  - Earliest Deadline First Scheduling
  - Proportional Share Scheduling

MultiProcessor 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 in common ready queue, or each has its own private queue of ready processes
  - Currently, most common
- **Processor affinity** – process has affinity for processor on which it is currently running
  - **soft affinity, hard affinity,** Variations including **processor sets**

Non Uniform Memory access (NUMA and CPU Scheduling)



Note that memory-placement algorithms can also consider affinity

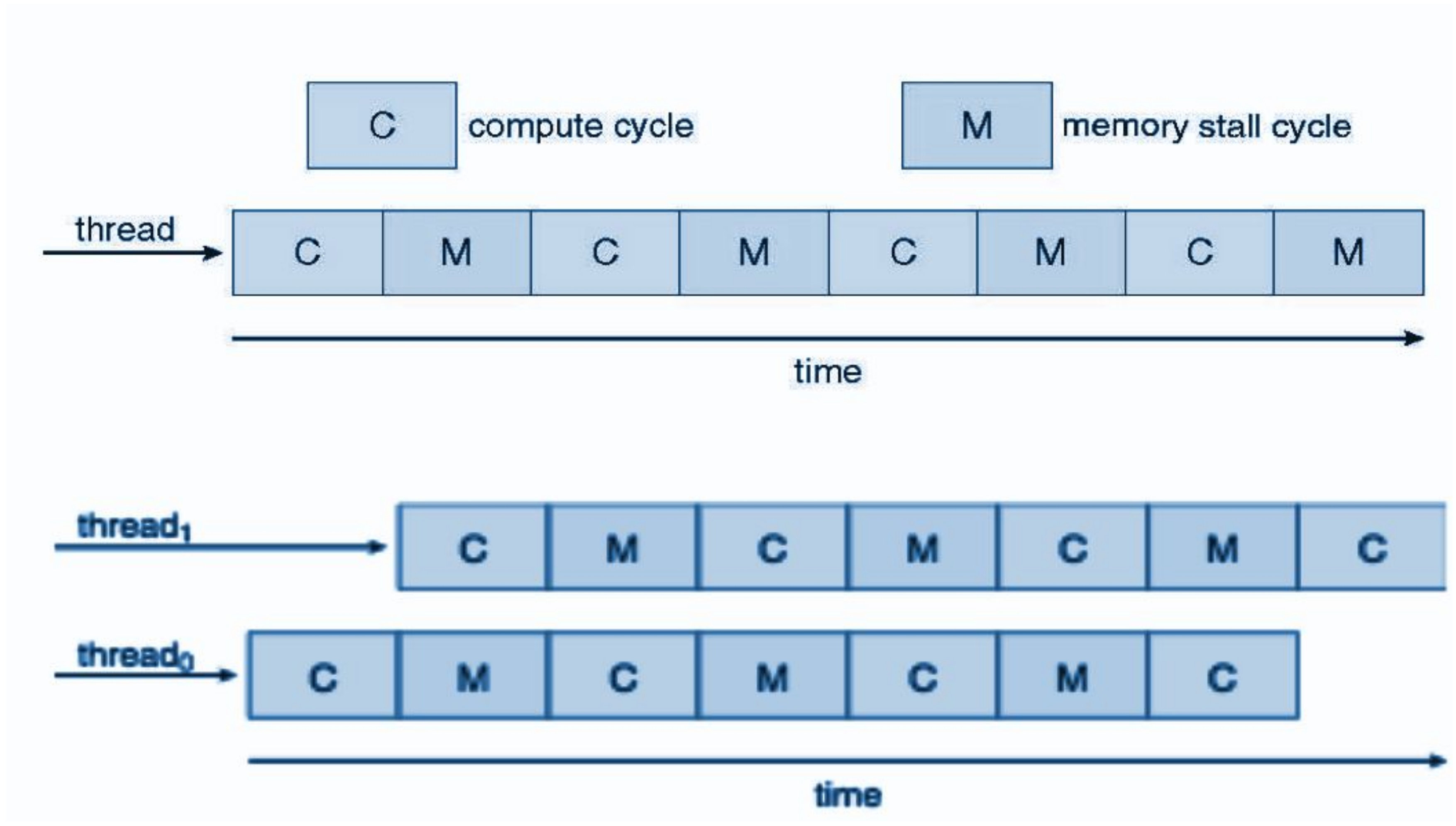
Multiple-Processor Scheduling – Load Balancing

- 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 and Multicore System



OPERATING SYSTEMS

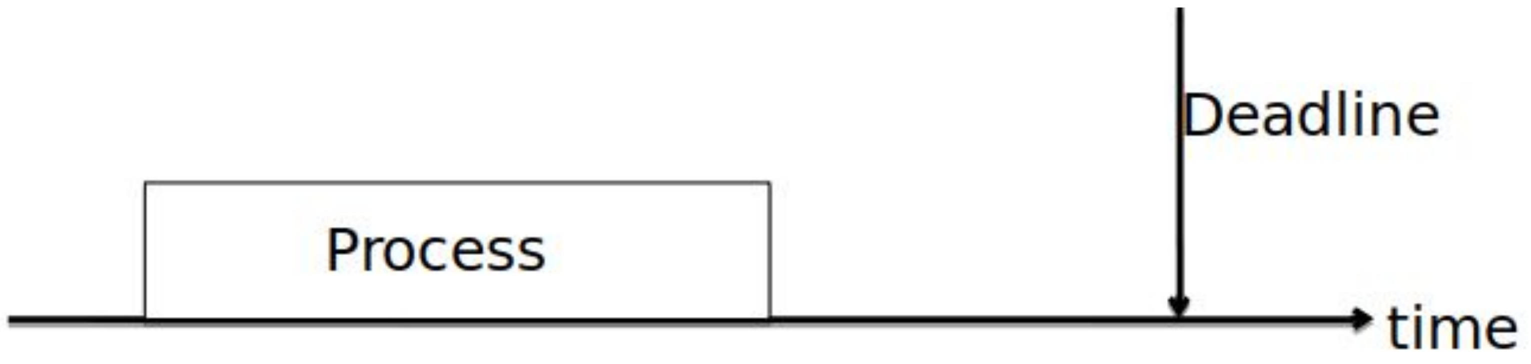
Real time Systems

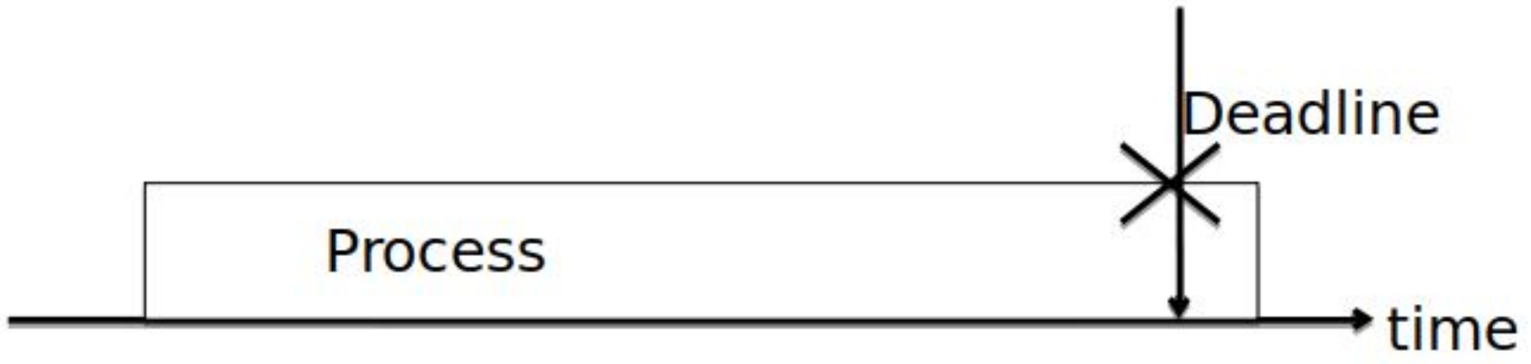
- A **real-time system** is one whose correctness depends on timing as well as functionality.

- When we discussed more traditional scheduling algorithms, the metrics we looked at were turnaround time (or throughput), fairness, and mean response time. But real-time systems have very different requirements, characterized by different metrics
  - **Timeliness**: How closely does it meet its timing requirements (e.g. ms/day of accumulated tardiness) ?
  - **Predictability**: How much deviation is there in delivered timeliness ?

- A few new concepts
  - **Feasibility:** Whether or not it is possible to meet the requirements or a particular task set ?
  - **Hard real-time :** There are strong requirements that specified tasks be run a specified intervals (or within a specified response time). Failure to meet this requirement (perhaps by as little as microsecond) may result in system failure.
  - **Soft real-time :** We may want to provide very good (e.g. microseconds) response time, the only consequences of missing a deadline are degraded performance or recoverable failures.

Real Time System Requirements





OPERATING SYSTEMS

Real time Systems

Real-time scheduling is more critical and difficult than traditional time-sharing, and in many ways it is. But real-time systems may have a few characteristics that make scheduling easier:

- We may actually know how long each task will take to run. This enables much more intelligent scheduling.

- **Starvation** (of low priority tasks) may be acceptable. In aeronautics, a spoiler (sometimes called a lift spoiler or lift dumper) is a device which intentionally reduces the lift component of an airfoil in a controlled way can miss deadline,

  The space shuttle absolutely must sense attitude and acceleration and adjust spoiler positions once per millisecond. But it probably doesn't matter if we update the navigation display once per millisecond or once every ten seconds. Telemetry transmission is probably somewhere in-between. Understanding the relative criticality of each task gives us the freedom to intelligently shed less critical work in times of high demand.

- The workload may be relatively fixed. Normally high utilization implies long queuing delays, as bursty traffic creates long lines. But if the incoming traffic rate is relatively constant, it is possible to simultaneously achieve high utilization and good response time.

Real time Systems Algorithms

- In the simplest real-time systems, where the tasks and their execution times are all known, there might not even be a scheduler. One task might simply call (or yield to) the next. This model makes a great deal of sense in a system where the tasks form a producer/consumer pipeline (e.g. MPEG frame receipt, protocol decoding, image decompression, display).

- In more complex real-time system, with a larger (but still fixed) number of tasks that do not function in a strictly pipeline fashion, it may be possible to do **static** scheduling. Based on the list of tasks to be run, and the expected completion time for each, we can define (at design or build time) a fixed schedule that will ensure timely execution of all tasks.

- But for many real-time systems, the workload changes from moment to moment, based on external events. These require **dynamic** scheduling. For **dynamic** scheduling algorithms, there are two key questions:

  1. how they choose the next (ready) task to run
     - shortest job first
     - static priority ... highest priority ready task
     - soonest start-time deadline first (ASAP)
     - soonest completion-time deadline first (slack time)
  2. how they handle overload (infeasible requirements)
     - best effort
     - periodicity adjustments ... run lower priority tasks less often.
     - work shedding ... stop running lower priority tasks entirely.

Real time Systems Algorithms

- Preemption may also be a different issue in real-time systems
    - In ordinary time-sharing, preemption is a means of improving mean response time by breaking up the execution of long-running, compute-intensive tasks.
    - A second advantage of preemptive scheduling, particularly important in a general purpose timesharing system, is that it prevents a buggy (infinite loop) program from taking over the CPU. The trade-off, between improved response time and increased overhead (for the added context switches), almost always favors preemptive scheduling. This may not be true for real-time systems

Real time Systems Algorithms

- Preemption may also be a different issue in real-time systems
  - Preempting a running task will almost surely cause it to miss its completion deadline.
  - Since we so often know what the expected execution time for a task will be, we can schedule accordingly and should have little need for preemption.
  - Embedded and real-time systems run fewer and simpler tasks than general purpose time systems, and the code is often much better tested , so infinite loop bugs are extremely rare

Real time Systems Algorithms

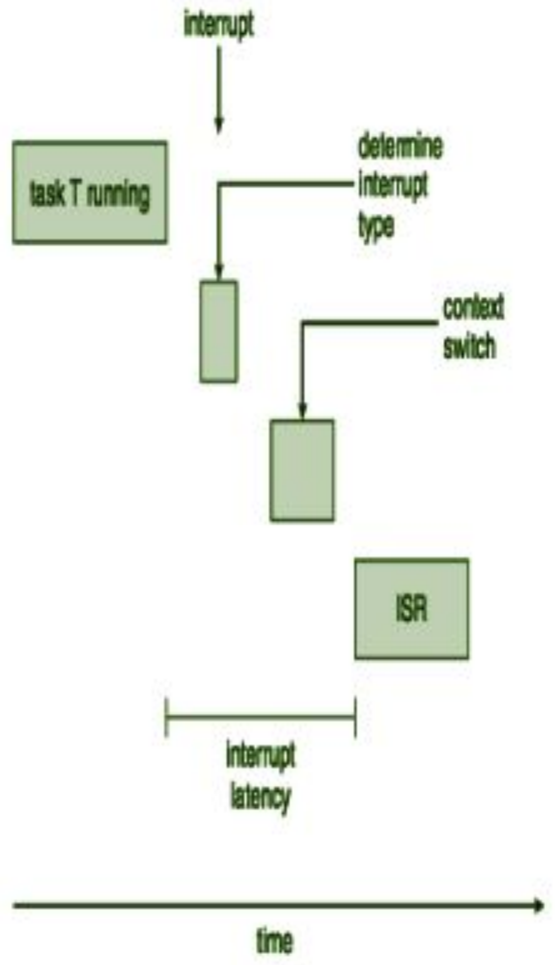
- For the least demanding real time tasks, a sufficiently lightly loaded system might be reasonably successful in meeting its deadlines.

- However, this is achieved simply because the frequency at which the task is run happens to be high enough to meet its real time requirements, not because the scheduler is aware of such requirements.

- A lightly loaded machine running a traditional scheduler can often display a video to a user's satisfaction, not because the scheduler "knows" that a frame must be rendered by a certain deadline, but simply because the machine has enough cycles and a low enough workload to render the frame before the deadline has arrived.

Real Time CPU Scheduling

- 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

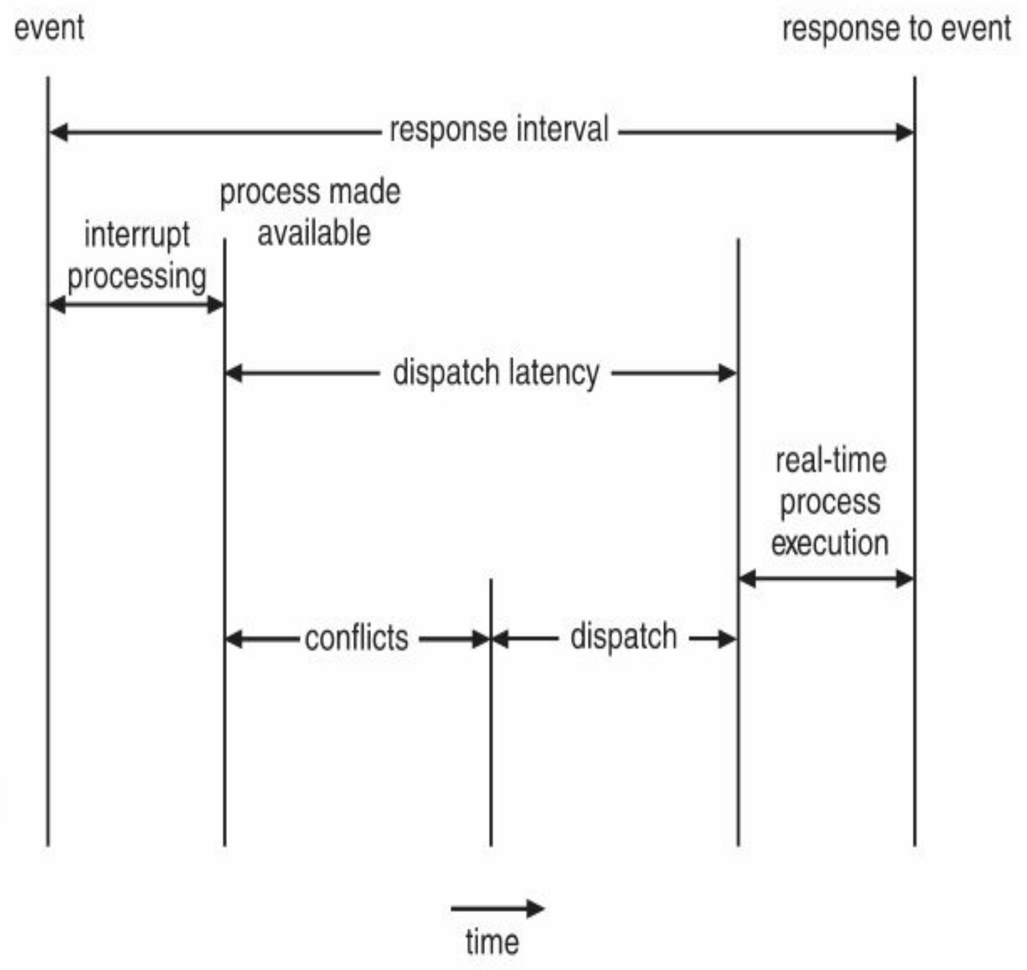


Real Time CPU Scheduling

- Conflict phase of dispatch latency:

  1. Preemption of any process running in kernel mode

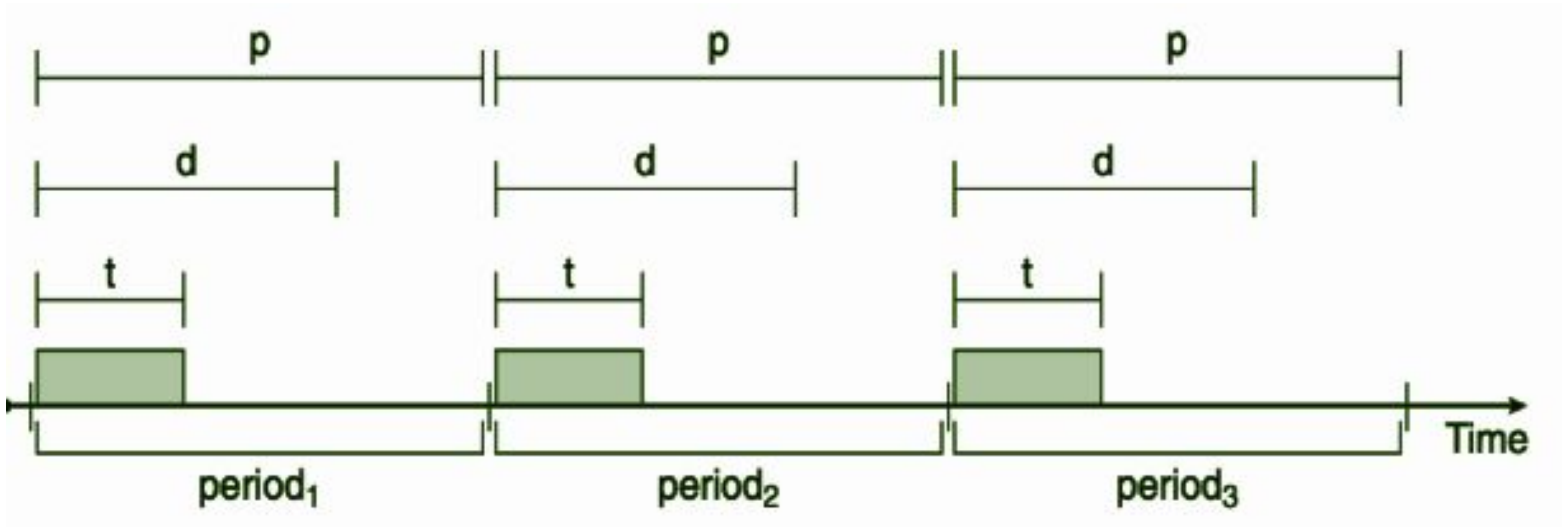
  2. Release by low-priority process of resources needed by high-priority processes



Real Time Priority Based Scheduling

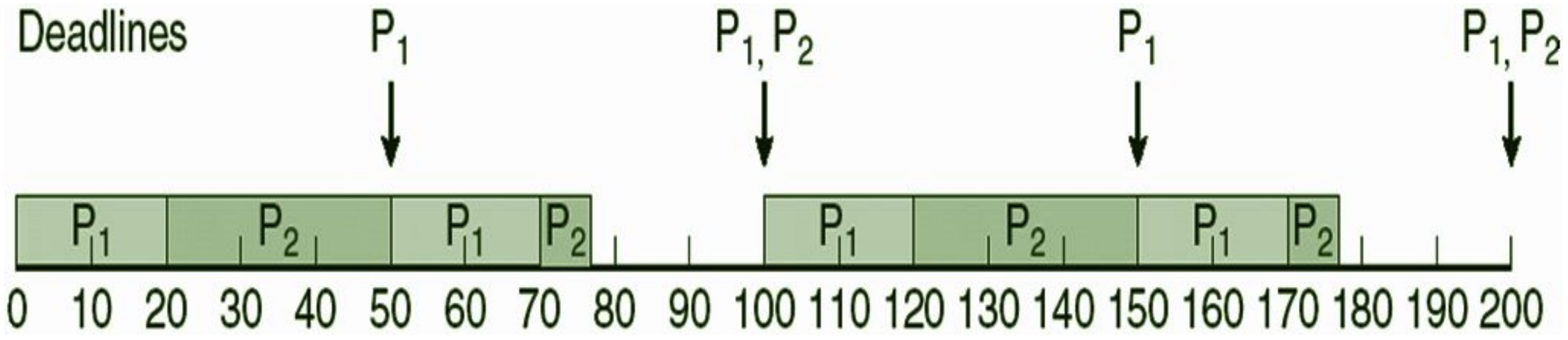
- Many real-time processes are periodic, i.e., they require CPU at constant intervals

- Has processing time t, deadline d, period p $0 \le t \le d \le p$; Rate of periodic task is 1/p

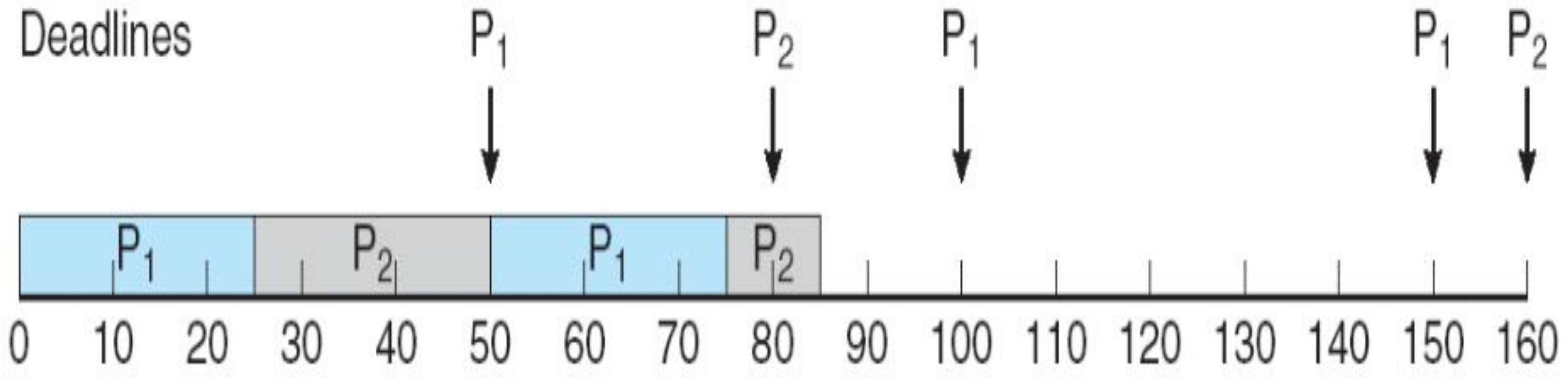


Rate Monotonic Scheduling

- A priority is assigned based on the inverse of its period
  - Shorter periods = higher priority
  - Longer periods = lower priority
  - P1=>20 is assigned a higher priority than P2=>30

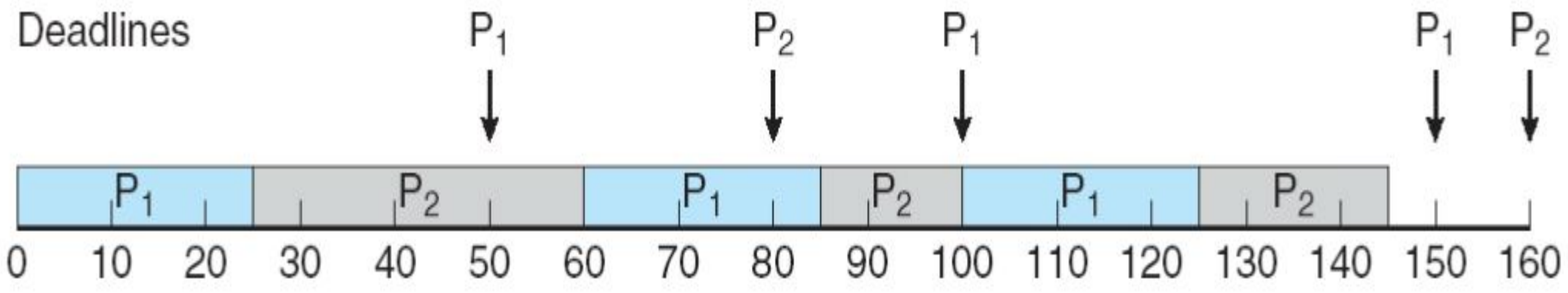


Missed deadlines with Rate Monotonic Scheduling



Earliest deadline First (EDF) Scheduling

- 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

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

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com