



RTOS

Characteristics
Time constraints
Scheduling



Real-Time Systems

- Definition of real-time system
 - ▶ “A real-time system is one in which the correctness of the computation not only depends on the logical correctness of the computation, but also on the time at which the result is computed. If the timing constraints of the system are not met, system failure is said to have occurred”
- Tasks or processes attempt to control or react to external events occurring in “real time”
- RT processes must be able to keep up events
 - ▶ Hard Real Time: timing violation produces a disaster
 - ▶ Soft Real Time: timing violations are undesirable but not catastrophic



RTOS

- A Real-Time Operating System (RTOS)
 - ▶ Is just one element of a complete real-time system
 - ▶ Must provide sufficient functionality to enable the overall real-time system to meet its requirements
 - ▶ Should ensure that all time critical events are handled as quickly and efficiently as possible
- RTOS are characterized by these main features
 - ▶ Deterministic behaviour
 - ▶ Responsiveness
 - ▶ User control
 - ▶ Reliability and fail-soft operation



RTOS: Determinism

- Operations are performed at fixed, predetermined times or within predetermined time intervals
- Concerned with
 - ▶ How long the operating system delays before acknowledging an interrupt
 - ▶ Whether the system has sufficient capacity to handle all requests within required time
- RTOS should show minimum interrupt latency
 - ▶ Typical measure:
 - Max delay to start ISR with highest priority
 - RT: 1 μ s - 1ms
 - Non-RT: 10ms - 100 ms



RTOS: Responsiveness

- How long, after acknowledgment, the operating system takes to **service** the interrupt
- Includes the time to begin execution of the ISR
 - ▶ If a context switch is necessary, the delay is longer than an ISR executed within the context of the current process
- Includes the amount of time to perform the interrupt, depending also on the specific hardware
- Depends on the possible ISR nesting and if ISRs can be interrupted



RTOS: User control

- Much broader in RTOS than in ordinary O.S.
- User should be able to
 - ▶ Specify paging or process swapping
 - ▶ Decide which processes must reside in main memory
 - ▶ Establish the rights of processes
 - ▶ Fine-grained control over task priorities
 - ▶ Select algorithms for disks scheduling
 - ▶ ...



RTOS: Reliability

- In the application domain of RTOS, degradation of performances may have catastrophic consequences
 - ▶ Attempt either to fix the problem or minimize its effects while continuing to run
 - ▶ Graceful degradation can be not enough
- Fail-soft
 - ▶ Ability to fail in such a way to preserve as much data and capability as possible
 - E.g., multiple attempts to improve data consistency
- Stability
 - ▶ The system adopts policies such that the most critical high priority tasks execute first, even if some needs of less important tasks can be not always met



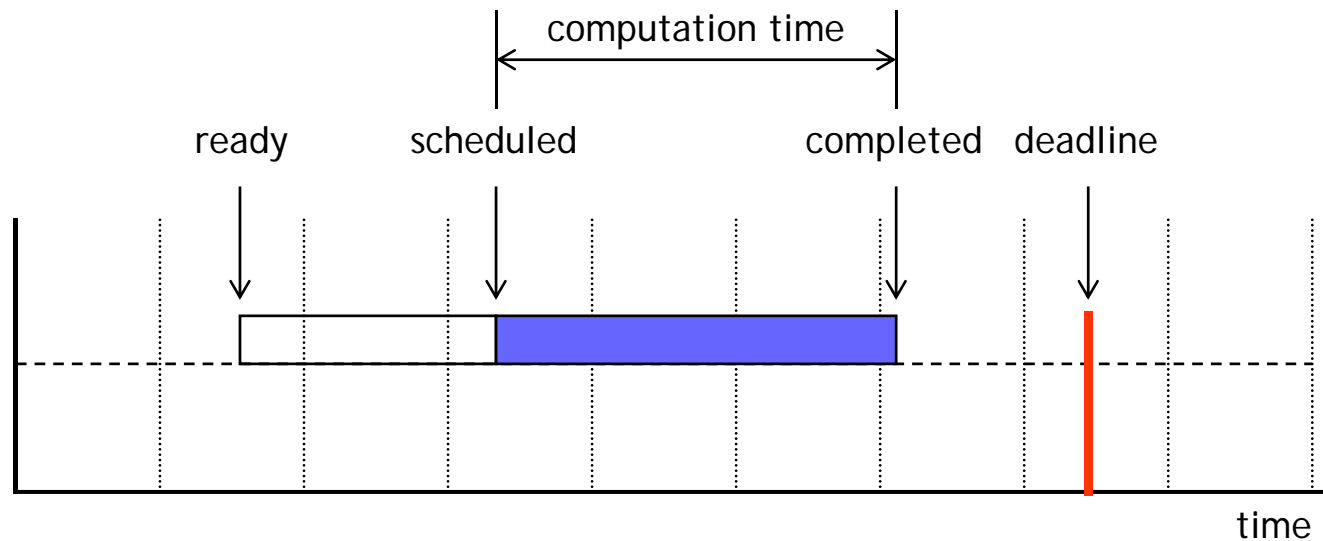
RTOS: Typical features

- Low cost, small size
- Management of at least 20 tasks
- Have minimum interrupt latency
- Deterministic execution time for all kernel services
- Provide at least the following services
 - ▶ Allow tasks to be dynamically created and deleted
 - ▶ Provide semaphore management services and IPC
 - ▶ Allow time delays and timeouts on kernel services
 - ▶ Management of special alarms
 - ▶ Use of special file to accumulate data at a fast rate
 - ▶ Preemptive priority-based scheduling with fast context switch



Time constraints

- Computational activities in a real-time system generally have timing constraints





Time constraints

- The "ready" event may be
 - ▶ Periodic
 - Tasks: Overall activity
 - Jobs: Instantiations or individually scheduled computations of the task. A task is a stream of jobs
 - ▶ Aperiodic but predictable
 - ▶ Unpredictable

- The "computation time" may be
 - ▶ Fixed in duration
 - ▶ Variable
 - ▶ Unpredictable



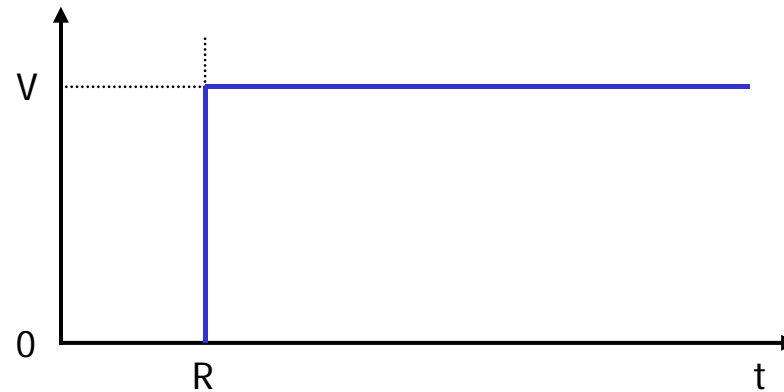
Deadlines

- Deadlines
 - ▶ Hard
 - The computation must be completed by the deadline time or a fatal error will occur
 - ▶ Soft
 - The deadline may just be a recommendation or preference for completion of the computation
- A "Value Function"
 - ▶ Indicates the criticality of a deadline
 - ▶ Indicates the value that the completion of a task contributes to the overall value of the system



Deadlines

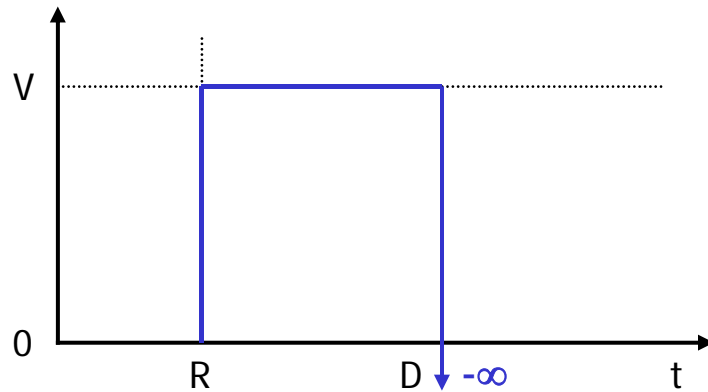
- In a non real-time system the time by which a task is completed does not influence the overall value - or quality - of the system
- The value function is the following



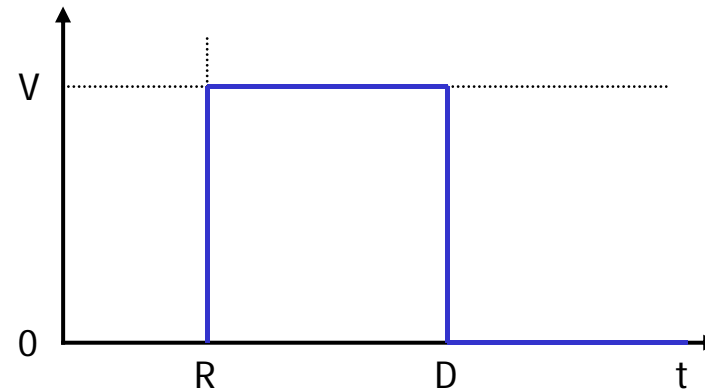


Deadlines

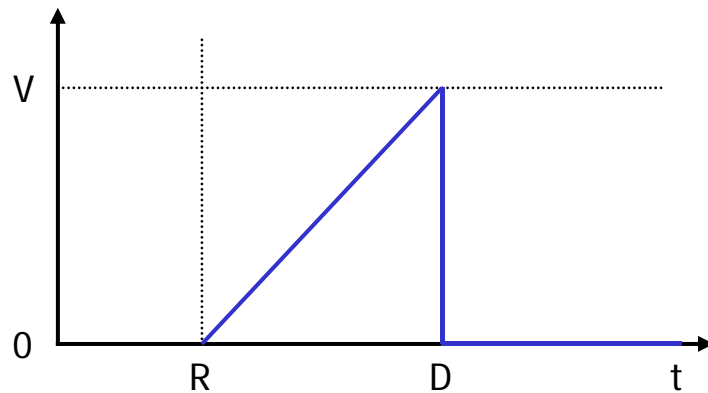
Catastrophic effect



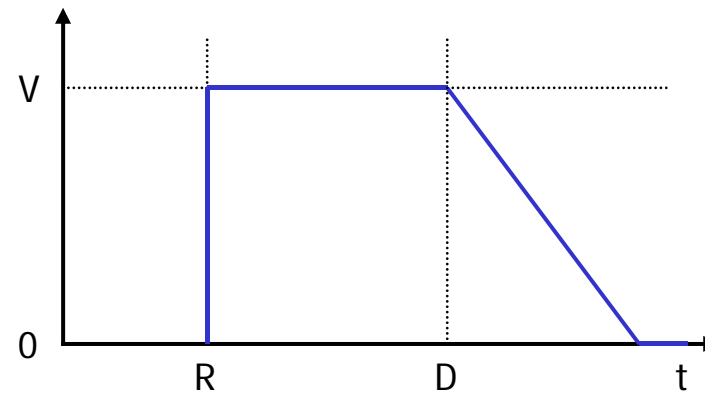
Hard Deadline



Ramped Hard Deadline



Soft Deadline



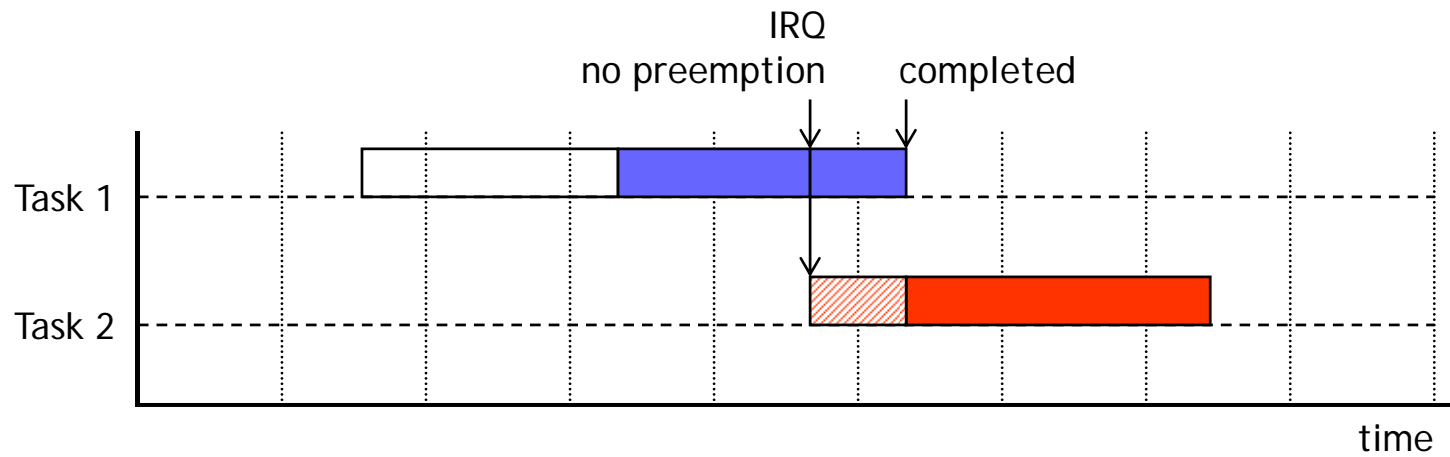
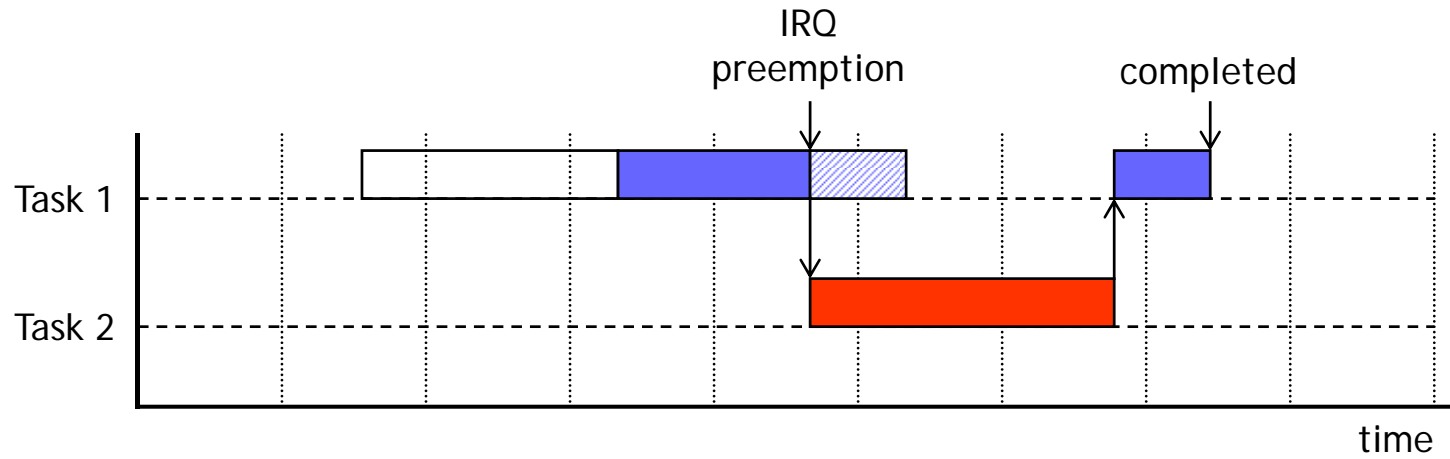


Preemption

- A computation or task is preemptable if it can be interrupted (suspended) when another more critical task needs to be executed
- A computation can be
 - ▶ Preemptable
 - ▶ Preemptable with one or more non-preemptible critical regions
 - ▶ Non-preemptable



Preemption





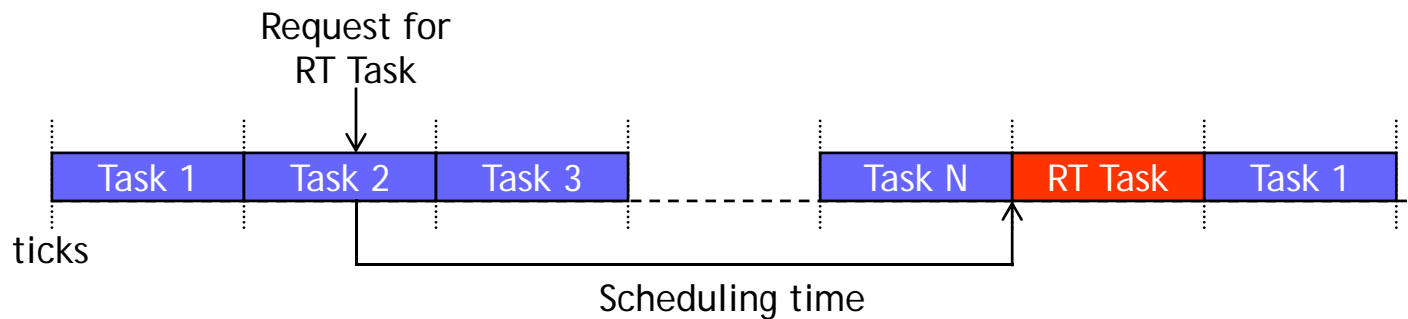
Scheduling

- The heart of a RTOS is the short-term scheduler
 - ▶ Not crucial
 - Fairness, Minimum average response time, ...
 - ▶ Crucial
 - All hard-RT tasks must complete by their deadline
 - As many soft-RT tasks as possible should also complete meeting their deadlines
- Most RTOSs are unable to deal with deadlines
 - ▶ Designed to be as responsive as possible to RT tasks
 - When deadline approaches, RT tasks can be quickly scheduled
 - ▶ This approach requires deterministic response time sometimes below milliseconds



Scheduling

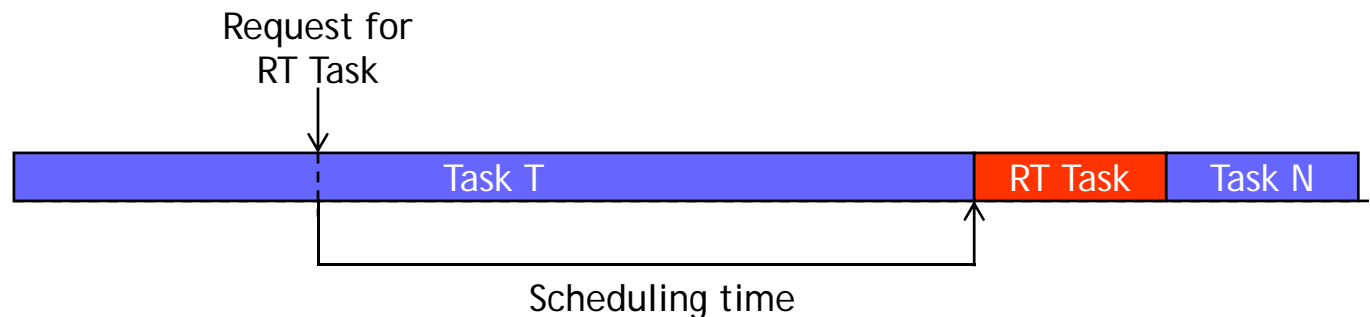
- In a Round Robin non-preemptive scheduler
 - ▶ The RT task is appended at the end of the ready queue
 - Wait for its next timeslice
 - ▶ The delay can be unacceptable for RT applications





Scheduling

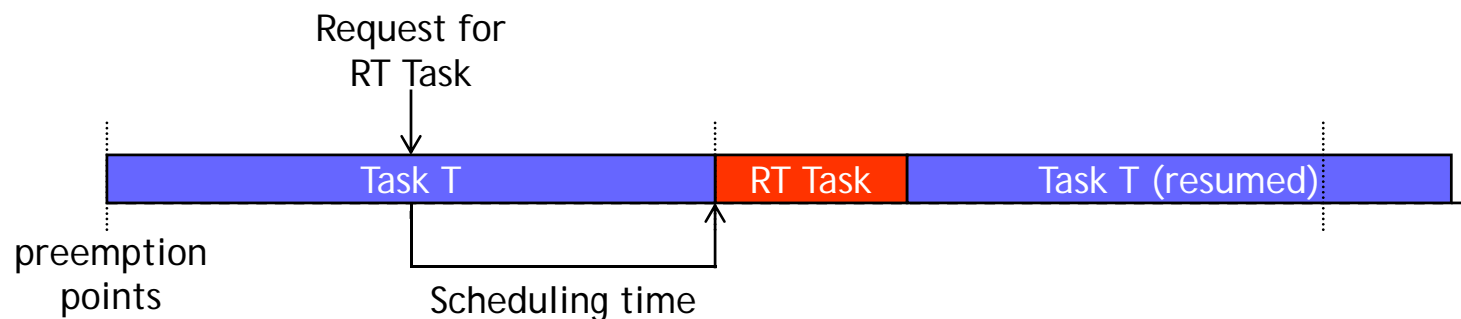
- Priority driven non-preemptive scheduler
 - ▶ RT tasks have higher priority
 - ▶ A RT task is scheduled when the current task T
 - Is blocked, or
 - Runs to completion (even if with low priority)
 - ▶ The RT is added at the head of the ready queue
 - ▶ Possible delay of seconds, unacceptable for real-time





Scheduling

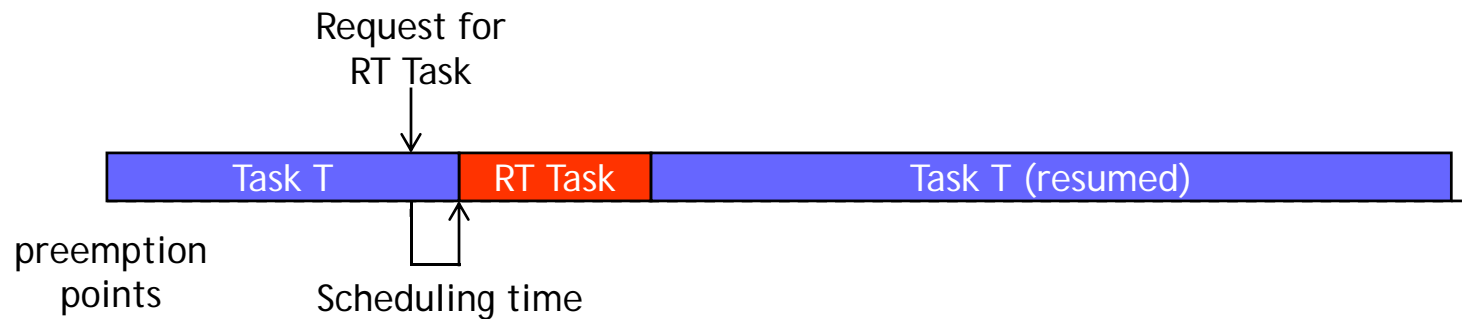
- Priority driven preemptive on preemption points
 - ▶ Preemption can only take place at the end of some regular intervals
 - ▶ In those points
 - The highest priority task is scheduled
 - Including kernel tasks
 - ▶ Delays in the order of some ms, adequate for some applications not for more demanding ones





Scheduling

- Immediate preemptive scheduling
 - ▶ The service to the interrupt is almost immediate
 - Apart from the case OS is executing a critical region
 - ▶ Scheduling delays fall down to 100 μ s, or less
 - A minimum amount of time is nevertheless required to save the context and to switch to the new context
 - ▶ Good for critical systems





Factors influencing scheduling

- Schedulability analysis
 - ▶ Yes/No
 - ▶ Static/dynamic
- The result of the analysis can be
 - ▶ A clear scheduling
 - ▶ A "plan" to follow at run-time for task dispatching
- Classes of algorithms
 - ▶ Static table-driven
 - ▶ Static priority-driven preemptive
 - ▶ Dynamic planning-based
 - ▶ Dynamic best-effort



Static table-driven

- Through a static feasibility analysis of schedule, determines, at run-time, when a task must begin execution
 - ▶ Applicable to periodic tasks
- Analysis inputs
 - ▶ Periodic arrival time
 - ▶ Execution time
 - ▶ Periodic ending deadline
 - ▶ Relative priority of each task
- Predictable but inflexible approach
 - ▶ Any change in the requirements of tasks imply the computation of a new schedule
- Example: Earliest-deadline first



Static priority-driven preemptive

- Uses standard priority-driven preemptive scheduler
- Static analysis is performed
 - ▶ No schedule is drawn-up
 - ▶ Assigns priorities to tasks
 - ▶ In RT systems depends on time constraints associated to tasks
- Example: Rate Monotonic



Dynamic planning-based

- Feasibility is determined at run-time
 - ▶ Rather than offline
- A task is accepted for execution
 - ▶ If and only if it is feasible to meet its time constraints
- Before its execution
 - ▶ An attempt is made to create a schedule including previous tasks and the new one
 - ▶ The deadline of the extended task set must be met



Dynamic best-effort

- Used in many commercial RT systems
 - ▶ Easy to implement
 - ▶ No feasibility analysis is performed
 - Typically, the tasks are aperiodic and no static scheduling analysis is possible
- When a task arrives
 - ▶ The system assign a priority based on its characteristics
 - E.g., based on earliest deadlines
 - ▶ The system tries to meet deadlines and aborts any started process whose deadline is missed
 - ▶ Until the task is completed (or deadline arrives), it is unknown if time constraints will be met



Deadline scheduling

- Real-time applications are not concerned with speed but with completing tasks
- Priorities
 - ▶ Provide a crude tool
 - ▶ Do not capture the requirement of completion or initiation at the most valuable time
- Other deadline related information should be taken into account



Deadline scheduling: Other information

- Ready time
 - ▶ Time at which task is ready for execution
 - ▶ For periodic task it is a sequence of times
 - Known in advance
- Starting/completion deadline
 - ▶ Typical RT application have either deadlines, but not both
- Processing time
 - ▶ In some cases it is supplied
 - ▶ In others cases the operating system measures an exponential average
- Resource requirements
 - ▶ In addition to microprocessor



Deadline scheduling: Other information

- Priority
 - ▶ Measures relative importance of tasks
 - ▶ Hard-RT tasks have “absolute” priority
- Subtask structure
 - ▶ Task possibly decomposed in
 - Mandatory subtasks: the only with hard RT deadlines
 - Optional subtasks: all other subtasks
- ...



Design issues

- Which task to schedule next?
 - ▶ For a given preemption strategy, scheduling tasks with the earliest deadline minimizes the fraction of tasks that miss their deadlines
 - True for single and multiprocessor configurations
- What sort of preemption is allowed?
 - ▶ When starting deadlines are given, non-preemptive scheduler makes sense
 - ▶ The RT task has the responsibility to block itself after executing the critical portion, allowing starting other RT deadlines



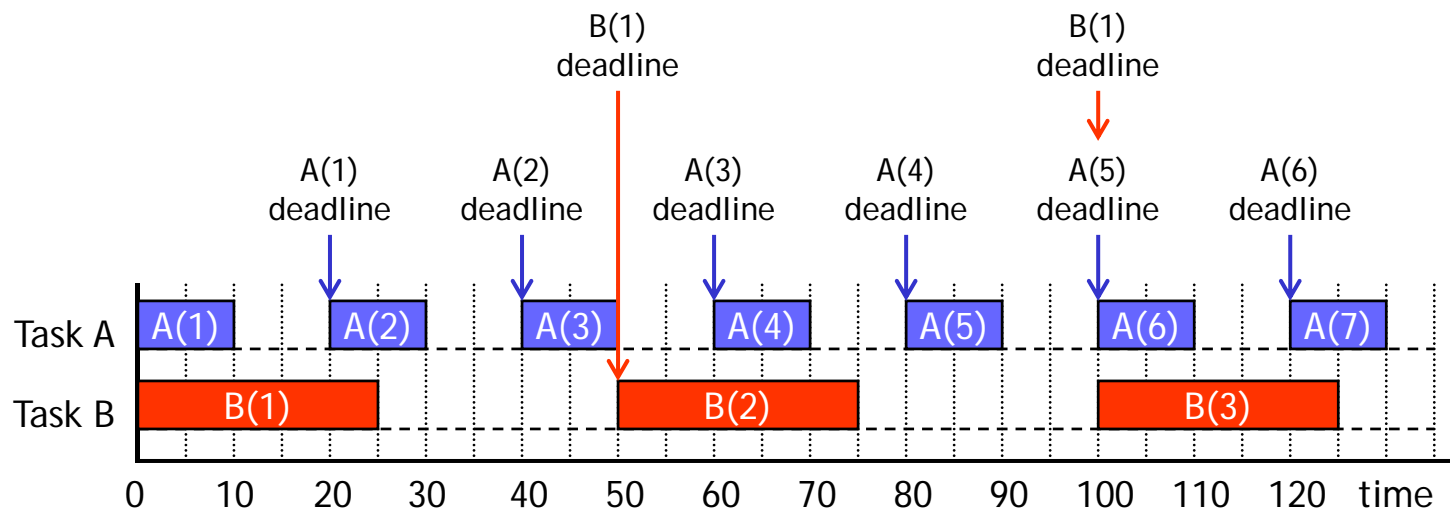
Example: Periodic tasks

- A system collects and processes data from two sensors, A and B
- Deadlines:
 - ▶ A must be read every 20 ms
 - ▶ B must be read every 50 ms
- Processing time (including OS overhead):
 - ▶ Processing data from A requires 10 ms
 - ▶ Processing data from B requires 25 ms
- The system makes scheduling decision every 10 ms



Example: Periodic tasks

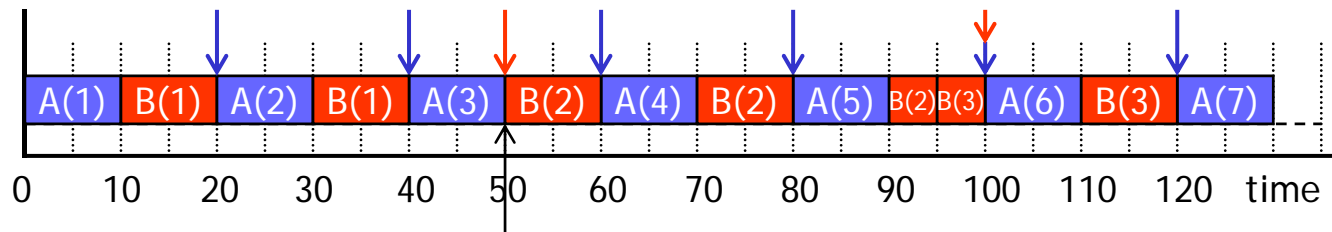
Process	Arrival time	Execution time	Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
...
B(1)	0	25	50
B(2)	50	25	100
...





Example: Periodic tasks

- Fixed priority scheduling
 - ▶ Task A has priority
 - ▶ Task B may miss some deadlines

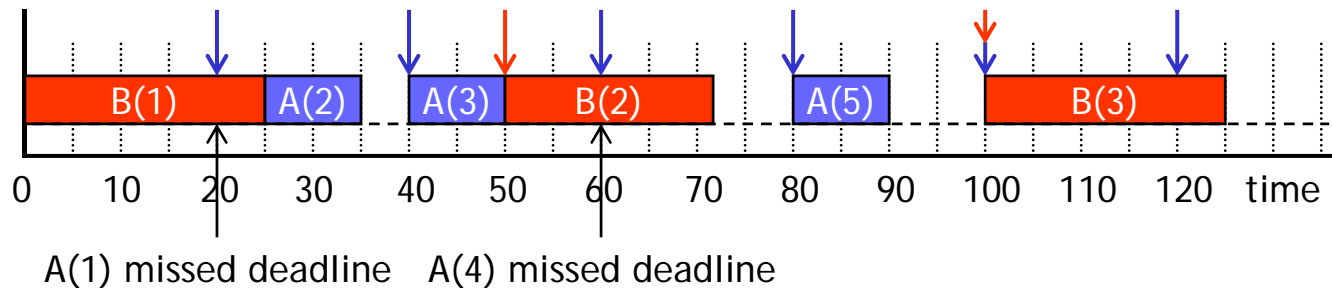


B(1) is not rescheduled since its deadline has been reached and will thus be violated



Example: Periodic tasks

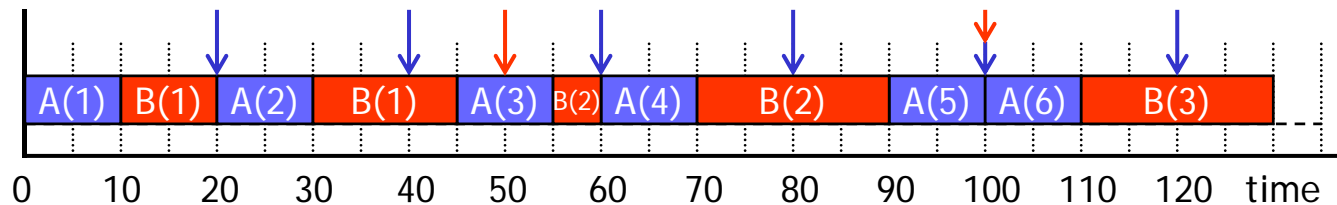
- Fixed priority scheduling
 - ▶ Task B has priority
 - ▶ Task A may miss some deadlines





Example: Periodic tasks

- Earliest deadline first with preemption
 - ▶ Completion deadlines are considered
 - ▶ At any preemption point the scheduler gives priority to the task with the earliest deadline
 - ▶ All task can meet their deadlines

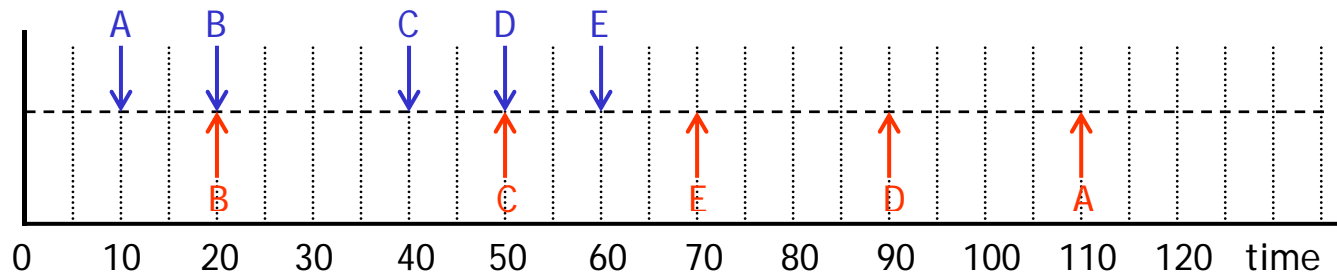


All task can be scheduled within their completion deadlines



Example: Aperiodic tasks

- Five tasks
 - ▶ All tasks have an execution time of 20 ms
 - ▶ Arrival times are unknown
 - ▶ Starting deadlines are considered

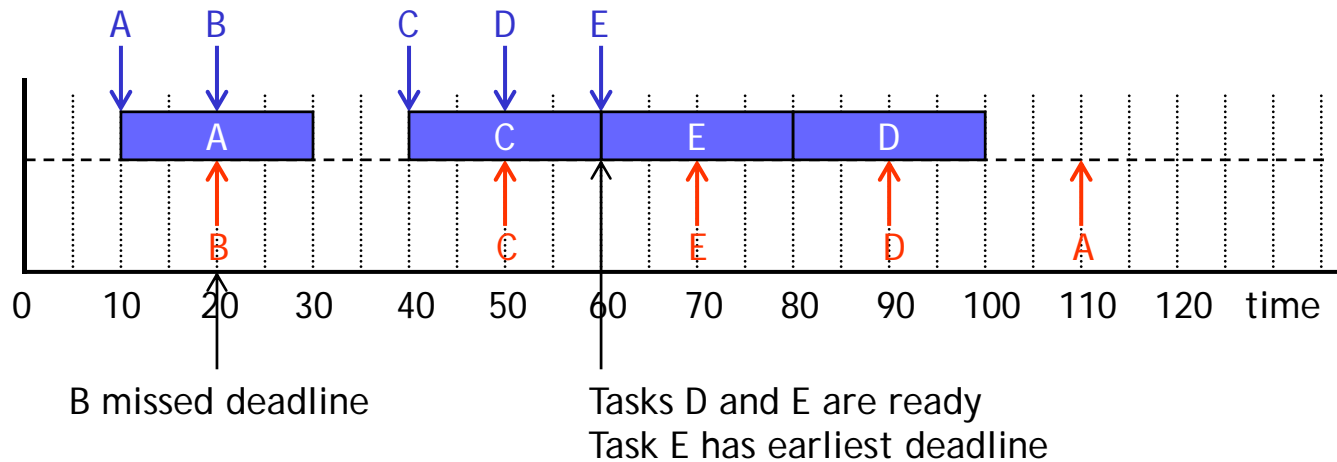


- Arrival times
- Starting deadlines



Example: Aperiodic tasks

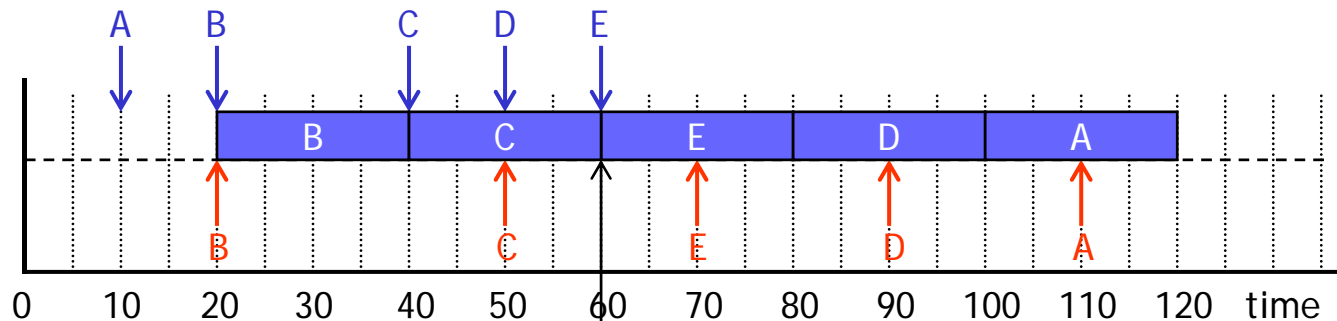
- Earliest deadline first
 - ▶ Starting deadlines are unknown before the task becomes ready for scheduling
 - ▶ Schedule the ready task with the earliest deadline and let the task run to completion





Example: Aperiodic tasks

- Earliest deadline with unforced idle times
 - ▶ Possible if deadlines are known before the task is ready
 - Schedule the eligible task with the earliest deadline
 - Let the task run to completion
 - ▶ Eligible tasks may not be ready
 - Processor remains idle even though ready tasks exist

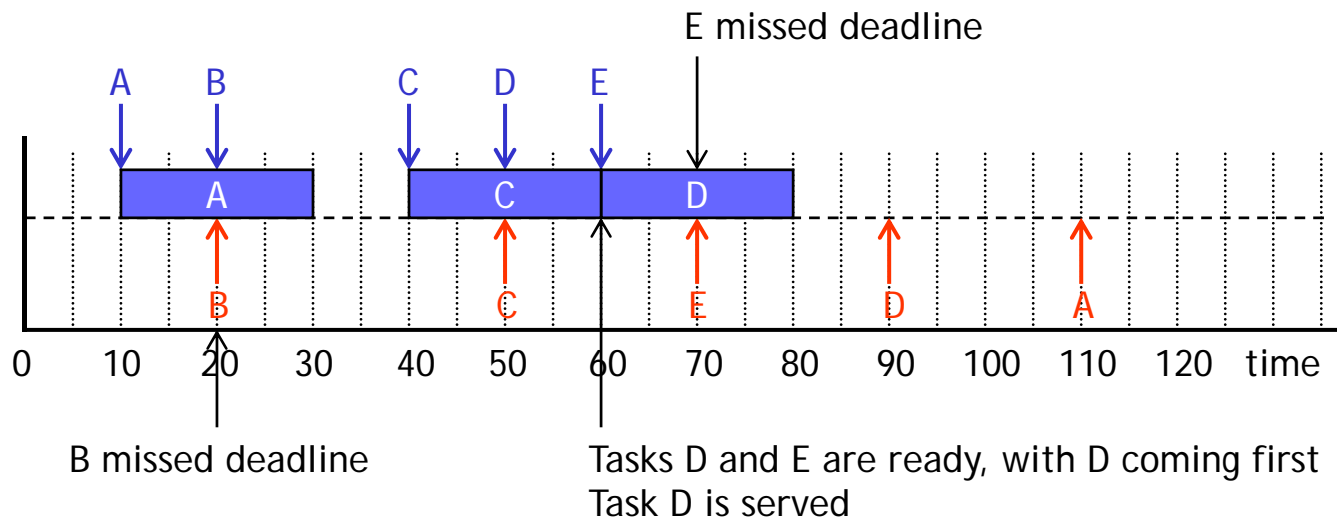


Tasks A, D and E are ready
Task E has earliest deadline



Example: Aperiodic tasks

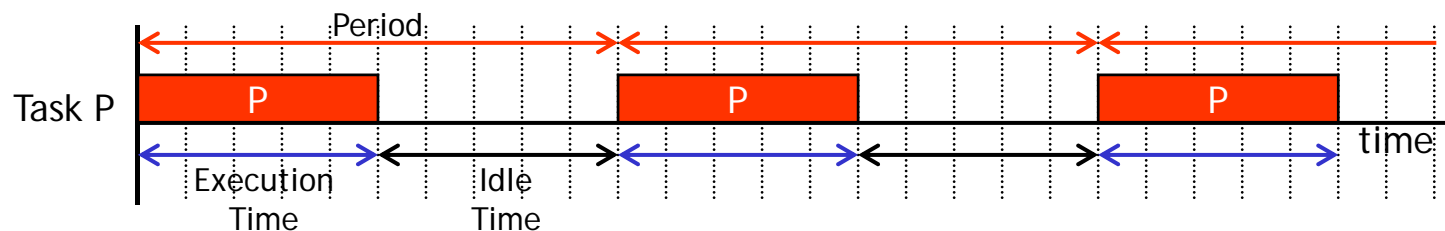
- First-Come, First-Serve (FCFS)
 - ▶ Tasks are ordered based on their arrival times
 - ▶ Tasks are served if their starting deadline is met
 - ▶ Tasks B and E do not meet their deadlines





Rate Monotonic Scheduling

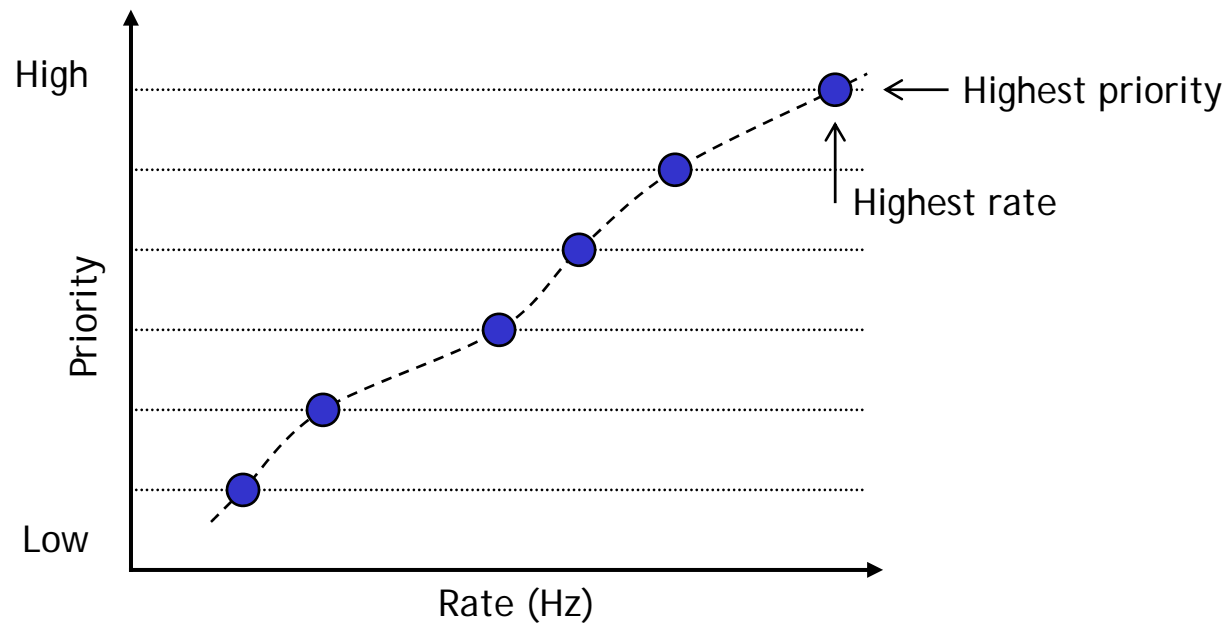
- Each task has
 - ▶ A period T
 - ▶ An execution time C for each occurrence
- In uniprocessor systems
 - ▶ $C < T$
 - ▶ Processor Utilization is $U = C/T (<100\%)$





Rate Monotonic Scheduling

- Assigns priorities to tasks
 - ▶ On the basis of their periods
 - ▶ Shortest period task has highest priority





Rate Monotonic Scheduling

- A measure of the effectiveness of a periodic scheduling algorithm is its capability to meet deadlines
- Consider N tasks with
 - ▶ A fixed period T_i
 - ▶ A fixed execution time C_i
- To meet all deadlines, the processor utilizations of individual tasks must not exceed the available computational power

$$\sum_{i=1}^N \frac{C_i}{T_i} \leq 1$$