Software Engineering for Embedded Systems

Real-time systems

Laura Carnevali

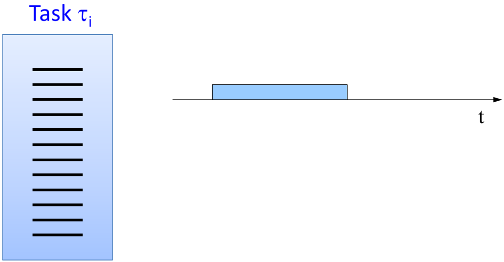
Software Technologies Lab
Department of Information Engineering
University of Florence
`http://stlab.dinfo.unifi.it/carnevali`
laura.carnevali@unifi.it

Outline

Basic concepts

Task (1/2)

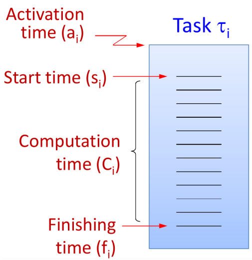
- A **task** is a sequence of instructions that, in the absence of other activities, is continuously executed by the processor until completion
  - Example: a single running task that incurs no preemption

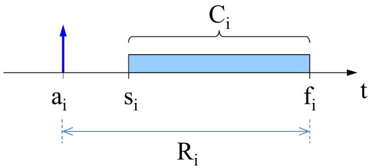


- A **process** is a program in execution, composed by concurrent **tasks** (also termed **threads**) that share a common memory space

Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

- Task characteristics
  - **Activation time** a_i: time at which a task becomes ready for execution
  - **Start time** s_i: time at which a task starts its execution
  - **Finishing time** f_i: time at which a task finishes its execution
  - **Computation time** C_i: task execution time without interruption
  - **Completion time** $K_i := f_i - s_i$ (in the example, $K_i = C_i$)
  - **Response time** $R_i := f_i - a_i$





Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Task states

- A task is termed **active** if it can potentially execute on the processor
  - An active task is termed **ready** if it is waiting for the processor
  - An active task is termed **running** if it is in execution
- A task is termed **blocked** if it is waiting to use some resource

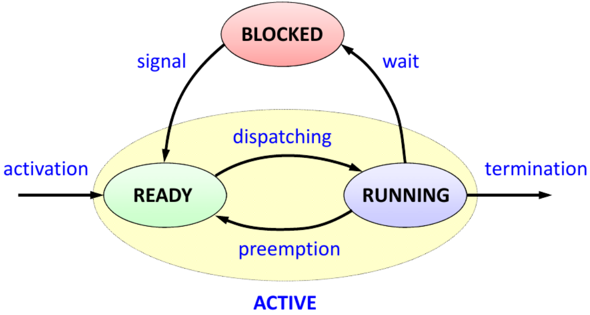


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Dispatching

- Ready tasks are kept in a **ready queue** managed by a **scheduling policy**
- A **dispatching** operation assigns the processor to the first task in the queue
- If there were multiple types of task $\Rightarrow$ there may be multiple ready queues

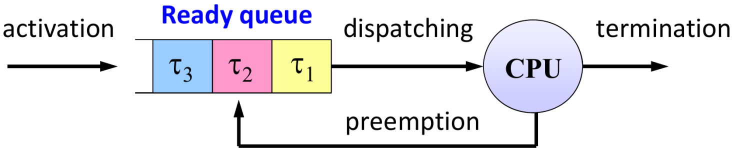


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Preemption

- Kernel mechanism that suspends the execution of the running task (which goes back in the ready queue) in favour of a more important task
- Enhances concurrency among tasks ☺
- Reduces the response time of high priority tasks, destroys program locality, and introduces a runtime overhead on the execution times of tasks ☹
- Can be disabled (temporarily/permanently) to ensure critical task consistency

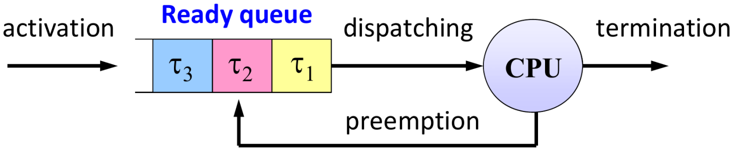


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Schedule

- Informally, a schedule is an assignment of tasks to the processor, which determines the execution sequence of the considered tasks
- Formally, a schedule for a task set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a function $\sigma : \mathbb{R}^+ \rightarrow \mathbb{N}$ such that $\forall\, t \in \mathbb{R}^+ \; \exists\, t_1, t_2 \in \mathbb{R}^+$ such that $t \in [t_1, t_2)$ and $\sigma(t) = \sigma(t') \;\forall\, t' \in [t_1, t_2]$ where:
  - $\sigma(t) = 0$ if the processor is idle at time t
  - $\sigma(t) = k \in \{1, 2, \ldots, n\}$ if the processor is executing task τ_k at time t
- Each time interval $[t_i, t_{i+1})$ is termed **time slice** $\forall\, i \in \mathbb{N}^+$
- At each time instant t_i the processor performs a **context switch** $\forall\, i \in \mathbb{N}^+$

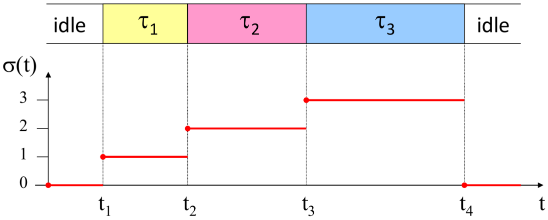


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Preemptive schedule

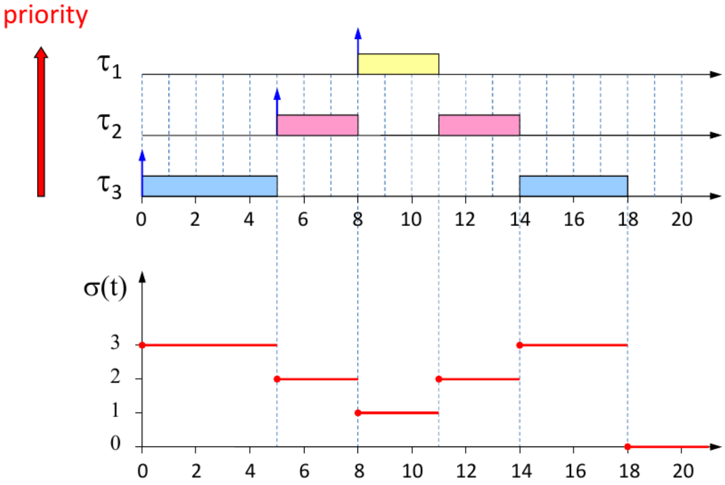


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Concurrent system

- Multiple tasks can be simultaneously active (i.e., ready or running)
- One and only one task is running at any time

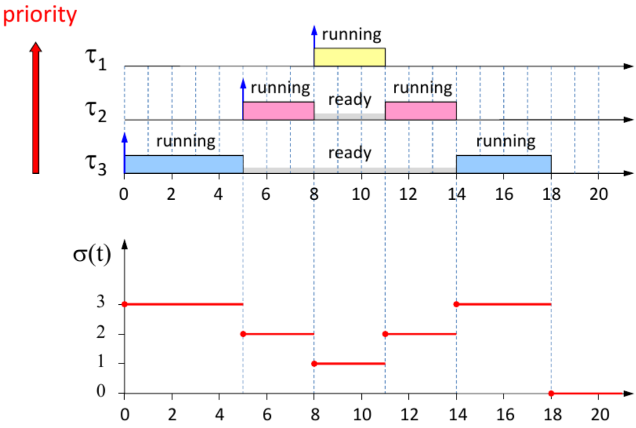
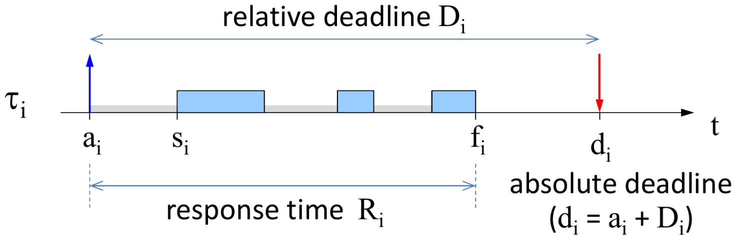


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Real-time task

- A **real-time task** is a task characterized by a timing constraint on its response time, which is termed (absolute/relative) **deadline**
  - **Absolute deadline** d_i: time before which a task should complete execution
  - **Relative deadline** $D_i := d_i - a_i$



- A real-time task τ_i is termed **feasible** if it completes its execution within its absolute deadline d_i, i.e., if $f_i \leq d_i$ or, equivalently, if $R_i \leq D_i$

Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Laxity

- **Laxity** X_i of task τ_i: the maximum delay that τ_i can experience after its activation and still complete within its deadline
  - Measured at the activation time
  - Equal to the relative deadline minus the computation time, i.e., $X_i := D_i - C_i$
  - Also termed **slack time**
- **Residual laxity** Y_i of task τ_i: the laxity measured upon the completion of τ_i
  - Equal to the absolute deadline minus the finishing time, i.e., $Y_i := d_i - f_i$

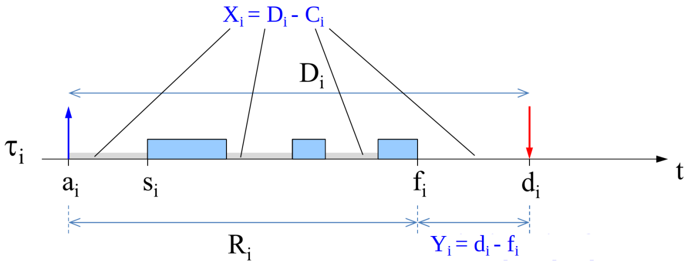
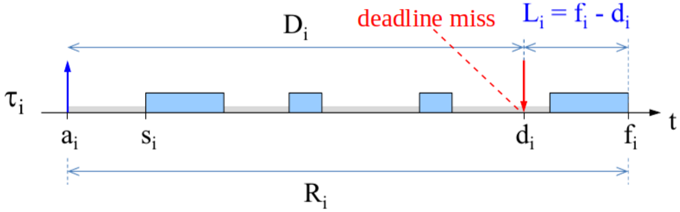


Image adapted from Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Lateness and tardiness

- **Lateness** L_i of task τ_i: delay of τ_i completion with respect to its deadline
  - Equal to the finishing time minus the absolute deadline, i.e., $L_i := f_i - d_i$
  - Negative if the task completes before the deadline



- **Tardiness** E_i of a task τ_i: time τ_i stays active after its deadline
  - Defined as $E_i := \max\{0, L_i\}$
  - Also termed **exceeding time**

Image adapted from Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Tasks and jobs

- A task running several times on different input data generates a sequence of identical activities termed **jobs** or **task instances** (same code, different data)

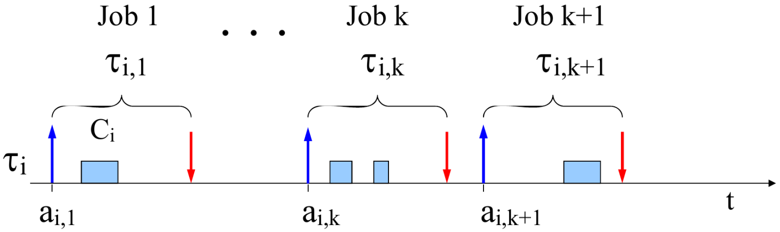
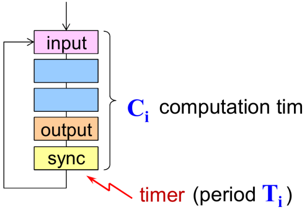


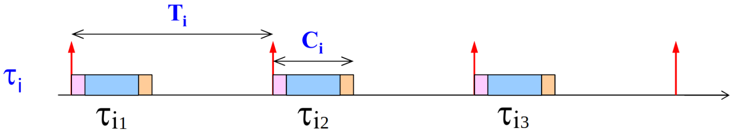
Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Activation mode of a task

- **Time-driven** activation mode
  - Task automatically activated by the operating system at predefined times
  - Activation mode of **periodic** tasks
- **Event-driven** activation mode
  - Task activated at the arrival of an event, by an interrupt or by another task through an explicit system call
  - Activation mode of **aperiodic** tasks

- A **periodic task** τ_i consists of an infinite sequence of **jobs** $\tau_{i1}, \tau_{i2}, \ldots, \tau_{ik}$ that are regularly activated at a constant rate, i.e., with period T_i
  - If $T_i = D_i$ then the task is termed a **pure periodic task**
  - The **task utilization factor** is $U_i := C_i/T_i$





Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Periodic task (2/3)

- Parameters of a periodic task τ_i
  - **Period** T_i, **computation time** C_i, **relative deadline** D_i
  - Activation time of the first job: $\Phi_i := a_{i,1}$ (termed **phase** of the task)
  - Activation time of the k-th job with $k > 1$: $a_{i,k} := \Phi_i + (k-1)T_i$
  - Absolute deadline of the k-th job with $k > 1$: $d_{i,k} := a_{i,k} + D_i$

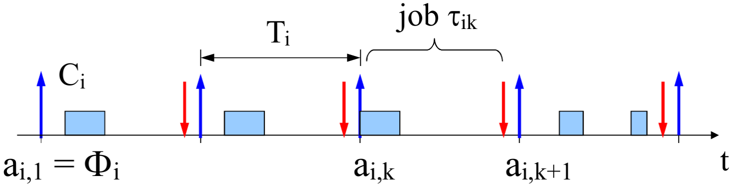


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Periodic task (3/3)

- Support for periodic tasks
  - Pseudo-code illustrating a fragment of a typical implementation

```
wait_for_activation();
while(condition) {
    ...
    wait_for_next_period();
}
```

  - In the time interval from the invocation of `wait_for_next_period()` until the beginning of the next period the task is neither active nor blocked... It is **idle**!

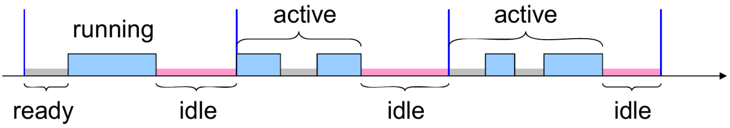
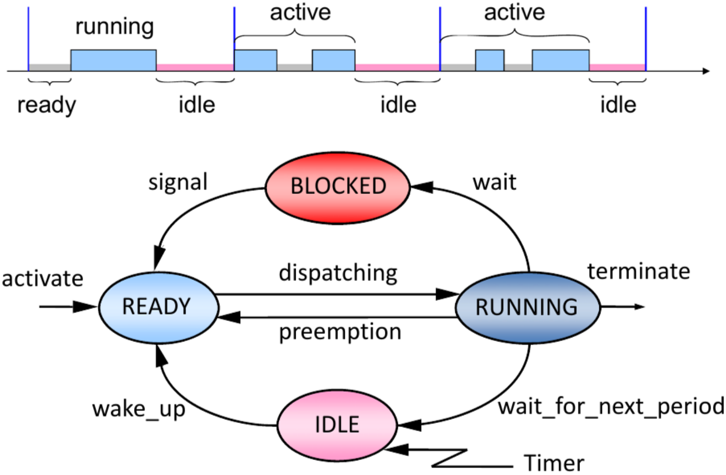


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Task states



Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Aperiodic task

- An **aperiodic task** τ_i consists of an infinite sequence of **jobs** $\tau_{i1}, \tau_{i2}, \ldots, \tau_{ik}$ that are not regularly activated
- A **sporadic task** τ_i is an aperiodic task whose consecutive jobs are separated by a **minimum inter-arrival time** T_i, i.e., $a_{i,k+1} \geq a_{i,k} + T_i$

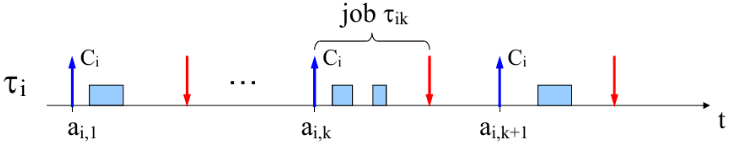


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Jitter

- **Jitter** is a measure of variation of a periodic event
  - **Absolute jitter**: $\max_k \{t_k - a_k\} - \min_k \{t_k - a_k\}$
  - **Relative jitter**: $\max_k \{|(t_k - a_k) - (t_{k-1} - a_{k-1})|\}$

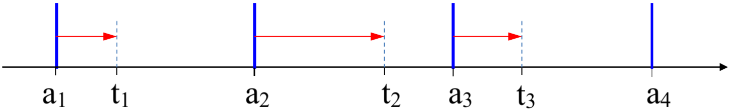


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Start time jitter

- Absolute start time jitter
  - Maximum deviation of the start time among all jobs
  - $\max_k\{s_{i,k} - a_{i,k}\} - \min_k\{s_{i,k} - a_{i,k}\}$
- Relative start time jitter
  - Maximum deviation of the start time among two consecutive jobs
  - $\max_k\{|(s_{i,k} - a_{i,k}) - (s_{i,k-1} - a_{i,k-1})|\}$

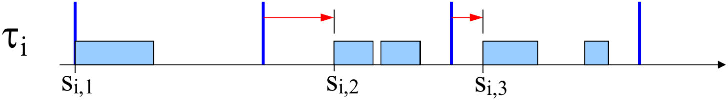


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Finishing time jitter

- Absolute finishing time jitter
  - Maximum deviation of the finishing time among all jobs
  - $\max_k\{f_{i,k} - a_{i,k}\} - \min_k\{f_{i,k} - a_{i,k}\}$
- Relative finishing time jitter
  - Maximum deviation of the finishing time among two consecutive jobs
  - $\max_k\{|(f_{i,k} - a_{i,k}) - (f_{i,k-1} - a_{i,k-1})|\}$

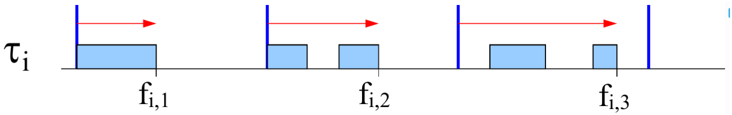


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Completion time jitter

- Absolute completion time jitter
  - Maximum deviation of the completion time among all jobs
  - $\max_k \{f_{i,k} - s_{i,k}\} - \min_k \{f_{i,k} - s_{i,k}\}$
- Relative completion time jitter
  - Maximum deviation of the completion time among two consecutive jobs
  - $\max_k \{|(f_{i,k} - s_{i,k}) - (f_{i,k-1} - s_{i,k-1})|\}$

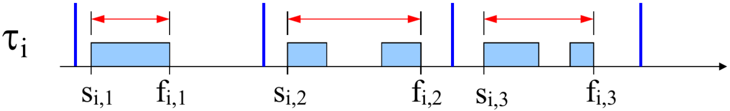


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Summary of task parameters

- Parameters **known offline** (specified by the programmer)
  - Period T_i
  - Computation time C_i
  - Relative deadline D_i
- Parameters **known online** (dependent on scheduling and actual execution)
  - Arrival time a_i, start time s_i, finishing time f_i, and response time R_i
  - Laxity X_i and residual laxity Y_i
  - Lateness L_i and tardiness E_i
  - Start time jitter, finishing time jitter, and completion time jitter

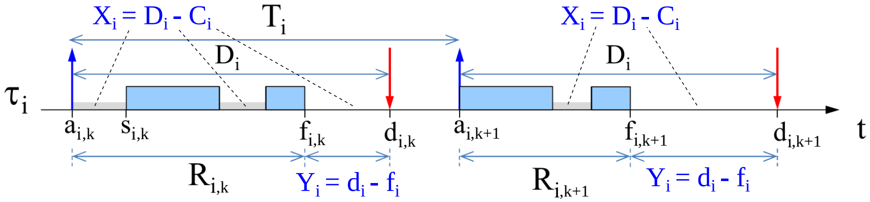


Image adapted from Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Feasibility vs schedulability (1/2)

- A **schedule** is termed **feasible** if all **task constraints** are satisfied
  - **Timing constraints**: activation, period, deadline, jitter
    - **Explicit constraints**: directly included in the system specification
    - **Implicit constraints**: not included in the system specification but needed to be met to satisfy performance requirements
  - **Precedence constraints**: impose an order in the execution of tasks, typically expressed in the form of a Directed Acyclic Graph (DAG) termed **task graph**
  - **Resource access constraints**: enforce synchronization in accessing mutually exclusive resources (to solve conflicts generated by concurrent access)

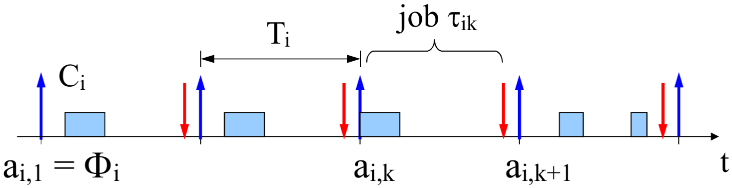
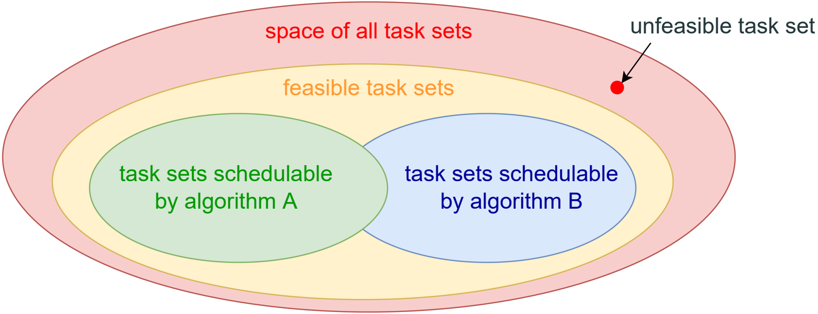


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Feasibility vs schedulability (2/2)

- A **task set** Γ is termed **feasible** if there exists an algorithm that generates a feasible schedule for Γ
- A **task set** Γ is termed **schedulable** by an algorithm A if A generates a feasible schedule for Γ



The scheduling problem

- Given a set of tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$, a set of processors $P = \{P_1, \ldots, P_m\}$, a set of resources $R = \{R_1, \ldots, R_s\}$, a set of constraints $C = \{H_1, \ldots, H_k\}$, the **scheduling problem** consists in finding an assignment of P and R to Γ that produces a feasible schedule (i.e., satisfying the constraints in C)
  - The scheduling problem is **NP-complete** (Garey&Johnson, 1979): in practice, scheduling algorithms have exponential execution time in the number of tasks
  - Polynomial time algorithms can be found under **simplifying assumptions**
    - Single processor, homogeneous task set (e.g., only periodic/aperiodic tasks), fully preemptable tasks, simultaneous activations, no precedence constraints, no resource constraints

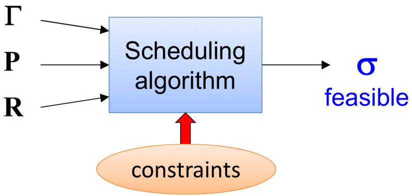


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Recap on complexity

- A decision problem is **NP** if it can be solved in polynomial time by a nondeterministic Turing machine
  - It may not be solved in polynomial time by a deterministic Turing machine (i.e., it may not be solved in polynomial time by a computer)
  - It can be solved in over-polynomial time by a deterministic Turing machine (i.e., it can be solved in over-polynomial time by a computer)
  - Over-polynomial complexity is typically exponential complexity
- A decision problem H is **NP-hard** if every decision problem in NP can be reduced in polynomial time to H
- A decision problem is **NP-complete** if it is NP and NP-hard
- Example: algorithm with the elementary step requiring 1 μs and $n = 20$ tasks
  - Linear complexity $O(n) \Rightarrow$ 20 μs
  - Polynomial complexity $O(n^{10}) \Rightarrow$ 2844.44 h
  - Exponential complexity $O(10^n) \Rightarrow \sim 3$ million years

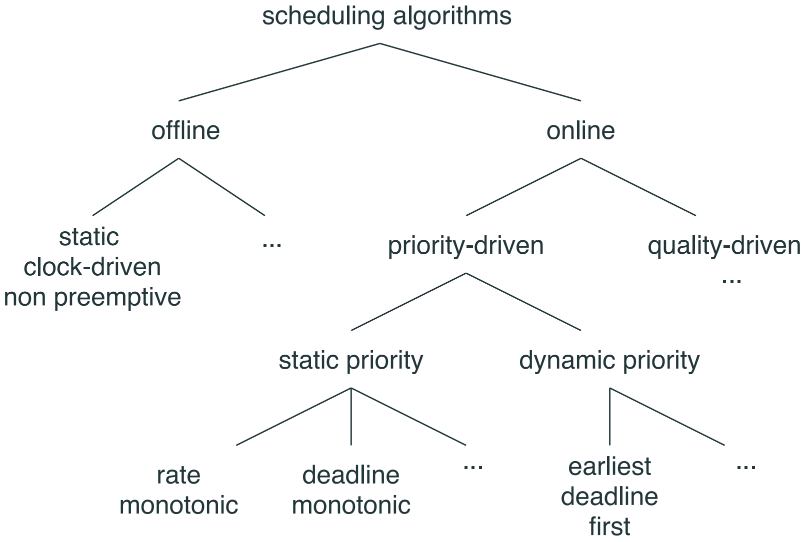
Taxonomy of scheduling algorithms (1/2)

- Preemptive vs non preemptive
  - Preemptive: a task can be interrupted by a higher priority task
  - Non preemptive: a task can never be interrupted by another task
- Static vs dynamic
  - Static: decisions taken based on fixed parameters which are statically assigned to tasks before activation
  - Dynamic: decisions taken based on parameters that can change with time
- Offline vs online
  - Offline: decisions taken before task activation (table driven scheduling)
  - Online: decisions taken at runtime based on the active tasks

Taxonomy of scheduling algorithms (2/2)

- Optimal vs heuristic
  - Optimal: generates a schedule that minimizes a cost function defined by an optimality criterion
  - Heuristic: generates a schedule according to a heuristic function that tries to satisfy an optimality criterion with no guarantee of success
- Guaranteed-based vs best-effort
  - Guaranteed-based: generates a feasible schedule if it exists; needed for hard real-time tasks
  - Best-effort: no guarantee of a feasible schedule; useful for soft real-time tasks; optimizes average performance
- Clairvoyant algorithm
  - It knows all future task activations
  - It can be used to compare performance

Classification of scheduling algorithms



Scheduling anomalies

- **Theorem (Graham, 1976)**: If a task set is optimally scheduled on a multiprocessor with some priority assignment, fixed number of processors, fixed execution times, and precedence constraints $\Rightarrow$
  $\Rightarrow$ increasing the number of processors, reducing execution times, or weakening precedence constraints can increase the schedule length
  - Small changes in parameters may have big unexpected consequences!
  - Example: faster processor, i.e., double speed (yellow indicates critical sections)

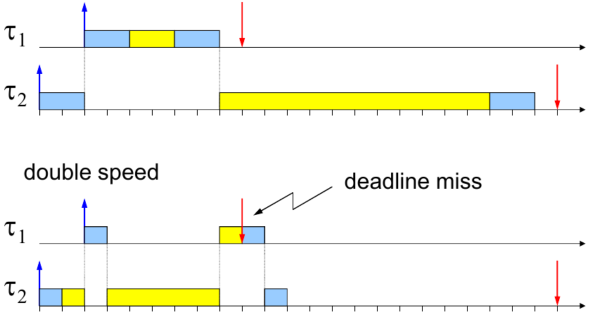
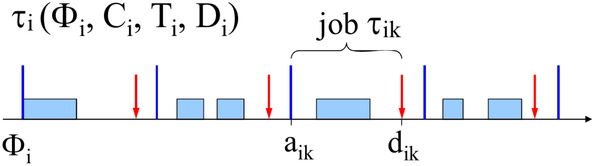


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Periodic task scheduling

Periodic task scheduling: problem formulation (1/2)

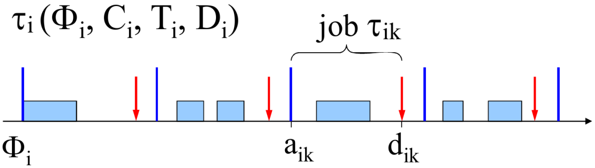
- Set of n **periodic tasks** $\Gamma = \{\tau_1, \ldots, \tau_n\}$, each task τ_i characterized by:
  - Initial arrival time (phase) $\Phi_i := a_{i,1}$
  - Worst Case Execution Time (WCET) C_i
  - Activation period T_i
  - Relative deadline $D_i \leq T_i$
- All tasks independent of each other
  - No precedence constraints
  - No resource constraints (except for the processor)
  - No task self-suspension (except for the suspension until the next period)
  - Task release upon task arrival
  - Zero or negligible kernel overheads



Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Periodic task scheduling: problem formulation (2/2)

- Goal: guarantee that each job $\tau_{i,k}$ of each periodic task τ_i
  - is activated at time $a_{i,k} := \phi_i + (k-1)T_i$
  - completes within time $d_{i,k} := a_{i,k} + D_i$



Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Periodic task scheduling: other parameters of interest

- **Hyper-period** $H := \text{lcm}\{T_1, \ldots, T_n\}$ (lcm = least common multiple)
  - Minimun time interval after which the schedule repeats itself
- **Job response time** $R_{i,k} := f_{i,k} - a_{i,k}$
- **Task response time** $R_i := \max_k\{R_{i,k}\}$
  - Maximum response time among all the jobs of the task

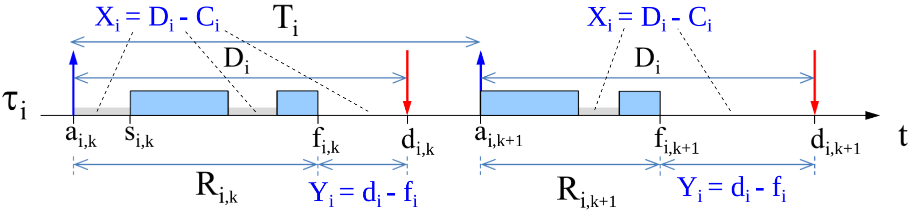
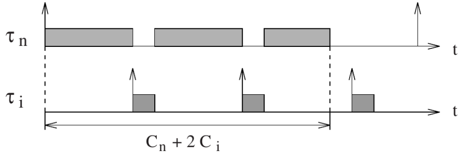


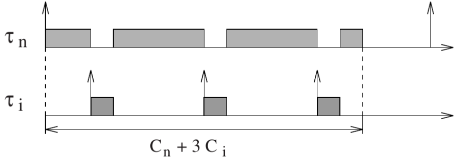
Image adapted from Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Periodic task scheduling: critical instant of a task

- The arrival time that yields the largest task response time
- Occurs when the task arrives together with higher priority tasks
  - Consider the interference of a high priority task τ_i with a low priority task τ_n



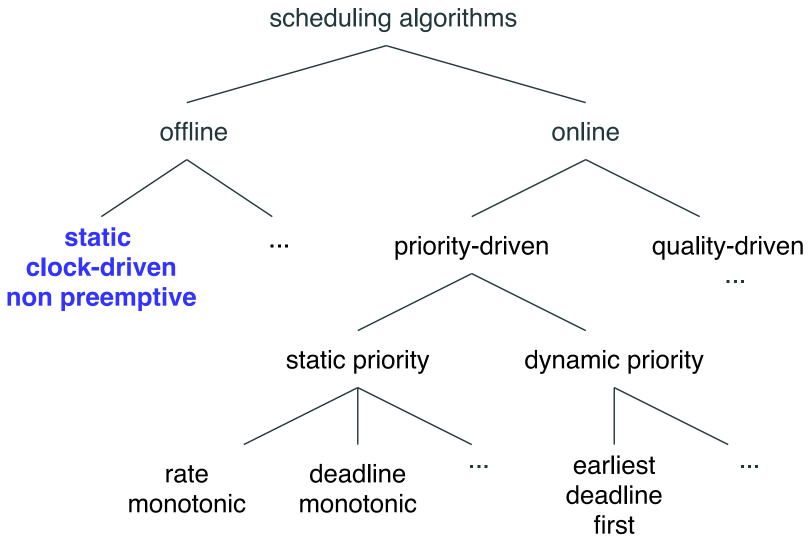
  - Reducing the phase of τ_i increases the response time of τ_n



  - Reducing the phase of any higher-prio task increases the response time of τ_n

Images from "Hard real-time computing systems" by Prof. G. Buttazzo

Static clock-driven non-preemptive scheduling (1/2)



Static clock-driven non-preemptive scheduling (2/2)

- For periodic tasks with **relative deadlines equal to periods**
- Decisions only made **offline** at **priory chosen time instants**
  (i.e., static schedule computed offline and stored in a table
  for use at runtime by a dispatcher activated by a timer)
  - Regular time instants: cyclic executive scheduling
  - Irregular time instants: timer-driven scheduling (timer needs reprogramming)
- Advantages
  - Simple implementation (no real-time operating system needed)
  - Low runtime overhead
  - Very low jitter
- Disadvantages
  - Not robust during overloads
  - Schedule difficult to expand
  - Aperiodic tasks not easy to handle

Cyclic executive scheduling (timeline scheduling)

- One of the most used scheduling algorithms in defense military systems and traffic control systems (e.g., Boeing 777, Space Shuttle)
- How it works
  - Time divided into intervals (**time slots**) of equal duration Δ (**minor cycle**)
  - One or more tasks **statically allocated** to each time slot (by hand) so that the sum of task WCETs in each time slot is not larger than Δ
  - Execution in each time slot activated by a timer
  - Schedule repeated after a time interval of duration T (**major cycle**)
- Typical values of parameters
  - $\Delta = \mathrm{gcd}\{T_1, \ldots, T_n\}$ (great common divisor of the task periods)
  - $T = \mathrm{lcm}\{T_1, \ldots, T_n\}$ (least common multiple of the task periods)

Cyclic executive scheduling: example

- A set of three tasks:

task τ_i	WCET C_i	period T_i
τ_A	10	25
τ_B	10	50
τ_C	10	100

- Parameter values selected to guarantee that $C_A + C_B \leq \Delta$ and $C_A + C_C \leq \Delta$:
  - Major cycle T =100
  - Minor cycle $\Delta = 25$

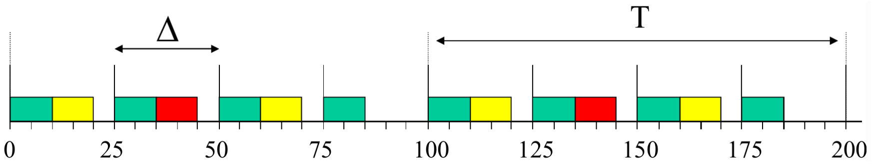
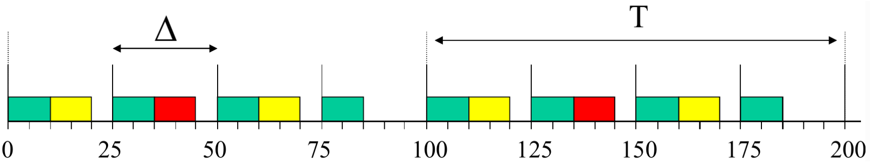
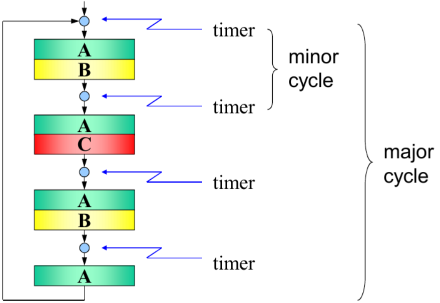


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Cyclic executive scheduling: implementation and coding





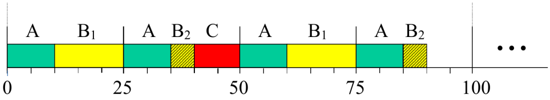

```
#define    MINOR      25          // minor cycle = 25 ms
      initialize_timer(MINOR);    // interrupt every 25 ms
      while (1) {
            sync();               // block until interrupt
            function_A();
            function_B();
            sync();               // block until interrupt
            function_A();
            function_C();
            sync();               // block until interrupt
            function_A();
            function_B();
            sync();               // block until interrupt
            function_A();
      }
```

Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Cyclic executive scheduling: disadvantages

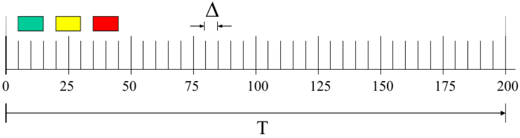
- Problems during **overloads** (task overruns)
  - Let the task continue $\Rightarrow$ possible domino effect on the other tasks
  - Abort the task $\Rightarrow$ possible inconsistent system state
- Difficulty in expanding the schedule in case of **task parameter changes**
  - WCET change: $C_B = 20 \Rightarrow C_A + C_B > \Delta \Rightarrow$ Split τ_B in 2 subtasks τ_{B1} and τ_{B2} with WCET equal to 15 and 5, respectively, and redesign the schedule!

task τ_i	WCET C_i	period T_i
τ_A	10	25
τ_B	**20**	50
τ_C	10	100



  - Period change: $T_B = 40 \Rightarrow \Delta = 5$, $T = 200 \Rightarrow 40$ synchronizations per major cycle! $\Rightarrow$ Very difficult to redesign the schedule by hand!

task τ_i	WCET C_i	period T_i
τ_A	10	25
τ_B	10	**40**
τ_C	10	100



Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Processor utilization factor U

- Fraction of the processor time spent in the task set execution:

$$U := \sum_{i=1}^{n} \frac{C_i}{T_i}$$

- Example: task set with $U = \frac{10}{25} + \frac{10}{40} + \frac{20}{100} = \frac{34}{40} = 0.85$

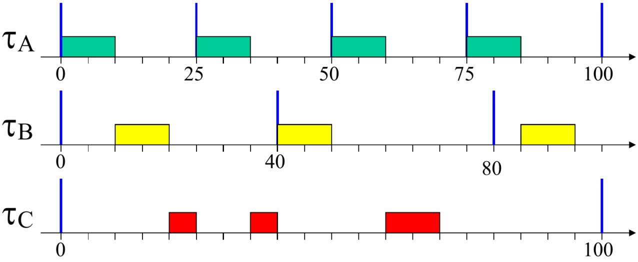
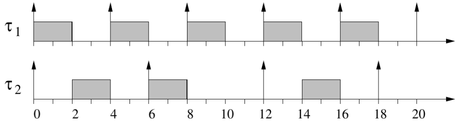


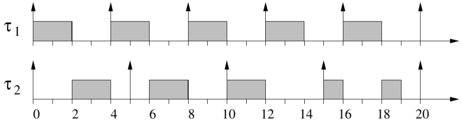
Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Upper bound $U_{\mathrm{ub}}(\Gamma, A)$ of the processor utilization factor U

- Upper bound $U_{\mathrm{ub}}(\Gamma, A)$ of U for a task set Γ under a scheduling algorithm A
  - If $U = U_{\mathrm{ub}}(\Gamma, A) \Rightarrow \Gamma$ is said to **fully utilize** the processor (if task WCETs are further increased $\Rightarrow$ the task set becomes unfeasible)
  - Each task set may have a different upper bound!
- Example (the processor is assigned to tasks in increasing order of periods)
  - Task set with $U_{\mathrm{ub}} = \frac{2}{4} + \frac{2}{6} = \frac{5}{6} \simeq 0.833$



  - Task set with $U_{\mathrm{ub}} = \frac{2}{4} + \frac{2}{5} = \frac{9}{10} = 0.9$



Images from “Hard real-time computing systems” by Prof. G. Buttazzo

Least upper bound $U_{\text{lub}}(A)$ of the processor utilization factor U

- Least upper bound $U_{\text{lub}}(A)$ of U under a scheduling algorithm A (min of utilization factors over all task sets that fully utilize the processor):
$$U_{\text{lub}}(A) := \min_{\Gamma} U_{\text{ub}}(\Gamma, A)$$
- Is a task set schedulable by A? If $U \leq U_{\text{lub}}(A) \Rightarrow$ YES; If $U > 1 \Rightarrow$ NO

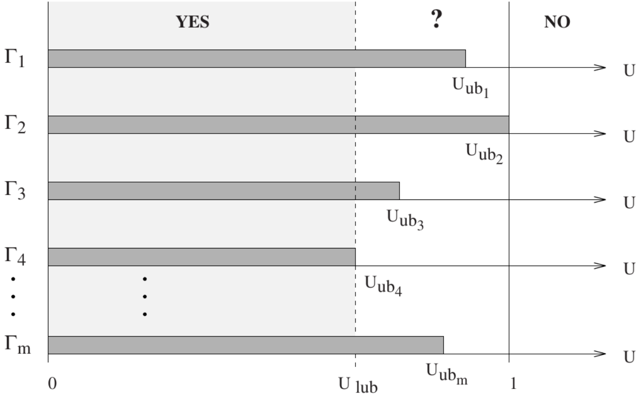


Image from "Hard real-time computing systems" by Prof. G. Buttazzo

Maximum value of the least upper bound $U_{\text{lub}}(A)$

Theorem

If the processor utilization factor of a task set Γ is larger than $1 \Rightarrow \Gamma$ is not feasible

Proof.

$U > 1 \Rightarrow UH > H$ since $H > 0 \Rightarrow \sum_{i=1}^{n} \frac{C_i}{T_i} H > H$ by U definition $\Rightarrow \sum_{i=1}^{n} \frac{H}{T_i} C_i > H$

$\frac{H}{T_i}$ is the (integer) number of times task τ_i is executed in the hyper-period

$\Rightarrow \frac{H}{T_i} C_i$ is the computation time requested by task τ_i in the hyper-period

$\Rightarrow \sum_{i=1}^{n} \frac{H}{T_i} C_i$ is the computation time requested by the task set in the hyper-period

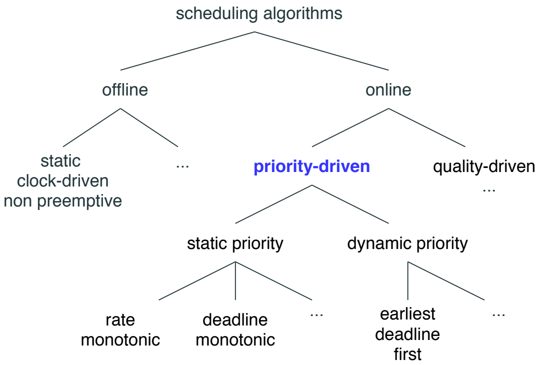
If demand exceeds the available processor time H, the task set is not feasible $\square$

Remark

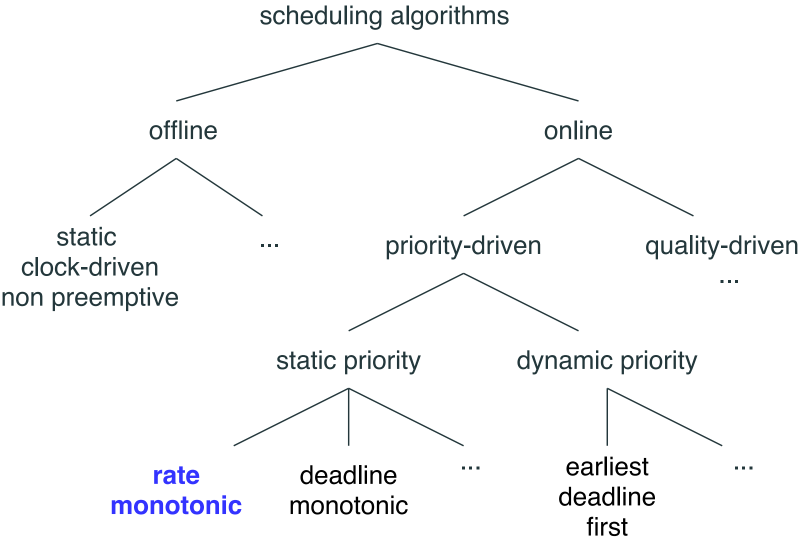
This result holds for any scheduling algorithm

Priority-driven scheduling

- How it works
  - Assign priority to each task based on its timing constraints
  - Verify the feasibility of the schedule using analytical techniques
  - Execute tasks on a priority-based kernel
- Schedulability analysis goal: construct an **optimal schedule** by considering the **processor utilization** and by computing the **response time** of each task



Rate Monotonic (RM) scheduling



RM: salient traits

- Preemptive **static** online scheduling algorithm
- Addressing scheduling of pure periodic tasks (i.e., $D_i = T_i \ \forall$ task τ_i)
- A task is assigned a **fixed priority** inversely proportional to its period
- Example: priority τ_A > priority τ_B > priority τ_C

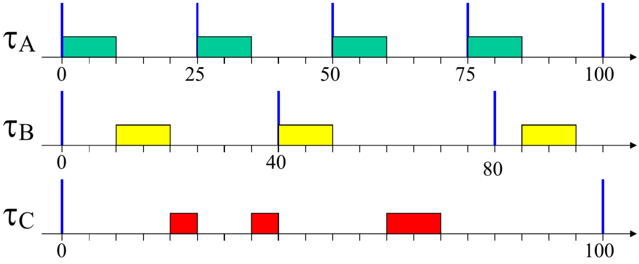


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

RM: optimality (Liu & Layland, 1973)

Theorem

RM is **optimal** *in the sense of* **feasibility** *among all* **fixed priority** *algorithms (for scheduling of periodic tasks with deadlines equal to periods):*

- *If a fixed priority schedule is feasible for a task set* $\Gamma \Rightarrow$
  $\Rightarrow$ *The RM schedule is feasible for* Γ
- *If the RM schedule is not feasible for a task set* $\Gamma \Rightarrow$
  $\Rightarrow$ *No fixed priority schedule is feasible for* Γ

- Note that the two statements are equivalent ($a \Rightarrow b$ if and only if $\neg b \Rightarrow \neg a$)
- Given that each task achieves its worst response time at its critical instant, then it is sufficient to **check optimality at the critical instants**:

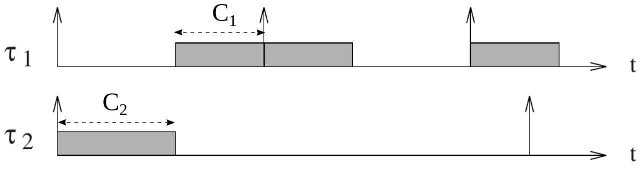
Theorem

If a fixed priority schedule is feasible for a task set Γ *at the critical instants* $\Rightarrow$
$\Rightarrow$ *The RM schedule is feasible for* Γ *at the critical instants*

C. L. Liu, J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", Journal of the ACM, 20(1), 1973

RM: proof of optimality for a task set made of 2 tasks (1/3)

- Consider a task set Γ made of 2 periodic tasks τ_1 and τ_2 with $T_1 < T_2$ (the proof can be easily extended to a task set made of n tasks)
- If priorities are not assigned according to RM $\Rightarrow$ priority τ_2 > priority τ_1 $\Rightarrow$ $\Rightarrow$ The schedule is feasible for Γ at the critical instants if $C_1 + C_2 \leq T_1$



Remark

It is sufficient to prove that:
If $C_1 + C_2 \leq T_1 \Rightarrow$ The RM schedule is feasible for Γ at the critical instants

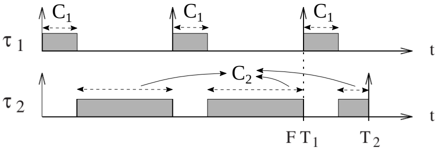
Image adapted from "Hard real-time computing systems" by Prof. G. Buttazzo

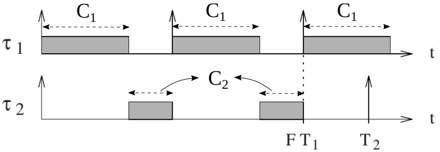
RM: proof of optimality (2/3)

- Number of periods of τ_1 entirely contained in T_2: $F := \lfloor T_2/T_1 \rfloor$
- If priorities are assigned according to RM $\Rightarrow$ priority τ_1 > priority τ_2

- Case 1 (left): $C_1 < T_2 - F\,T_1$ (i.e., all the jobs of τ_1 released within $[0, T_2)$ are completed before the second job of τ_2 is released)
  - The task set is schedulable if $(F+1)C_1 + C_2 \leq T_2$
  - We prove that $C_1 + C_2 \leq T_1 \Rightarrow (F+1)C_1 + C_2 \leq T_2$

- Case 2 (right): $C_1 \geq T_2 - F\,T_1$ (i.e., some job of τ_1 released within $[0, T_2)$ is not completed before the second job of τ_2 is released)
  - The task set is schedulable if $F\,C_1 + C_2 \leq F\,T_1$
  - We prove that $C_1 + C_2 \leq T_1 \Rightarrow F\,C_1 + C_2 \leq F\,T_1$





Images adapted from “Hard real-time computing systems” by Prof. G. Buttazzo

RM: proof of optimality (3/3)

- Case 1: We prove that $C_1 + C_2 \leq T_1 \Rightarrow (F+1)C_1 + C_2 \leq T_2$
  - $C_1 + C_2 \leq T_1 \Rightarrow F\,C_1 + FC_2 \leq F\,T_1$ given that $F \geq 1 \Rightarrow$
    $\Rightarrow F\,C_1 + C_2 \leq F\,C_1 + F\,C_2 \leq F\,T_1$ given that $F \geq 1 \Rightarrow$
    $\Rightarrow (F+1)C_1 + C_2 \leq F\,T_1 + C_1 \Rightarrow$
    $\Rightarrow (F+1)C_1 + C_2 \leq F\,T_1 + C_1 < T_2$ given that $C_1 < T_2 - F\,T_1 \Rightarrow$
    $\Rightarrow (F+1)C_1 + C_2 < T_2$
- Case 2: We prove that $C_1 + C_2 \leq T_1 \Rightarrow FC_1 + C_2 \leq FT_1$
  - $C_1 + C_2 \leq T_1 \Rightarrow F\,C_1 + FC_2 \leq F\,T_1$ given that $F \geq 1 \Rightarrow$
    $\Rightarrow F\,C_1 + C_2 \leq F\,C_1 + F\,C_2 \leq F\,T_1$ given that $F \geq 1 \Rightarrow$
    $\Rightarrow F\,C_1 + C_2 \leq F\,T_1$

RM guarantee test (Liu & Layland, 1973)

Theorem

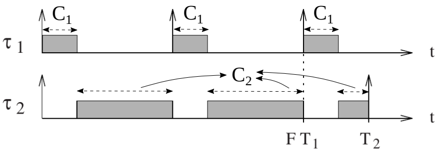
If $U \le n(2^{1/n} - 1)$ for a set Γ of n pure periodic tasks $\Rightarrow$ Γ is schedulable by RM

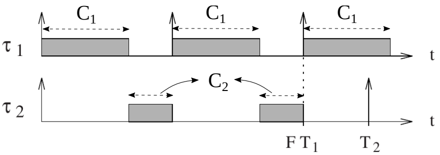
- The test is **only sufficient**
- Polynomial complexity $O(n)$ with respect to the number n of tasks
- Proof methodology
  - Assign priorities to tasks according to RM
  - Assume simultaneous task arrivals (worst case scenario for the task set)
  - Increase all computation times so as to fully utilize the processor
  - Compute the upper bound U_{ub} on U
  - Minimize U_{ub} with respect to all the other parameters so as to derive U_{lub}

C. L. Liu, J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", Journal of the ACM, 20(1), 1973

RM guarantee test: proof for 2 tasks (1/5)

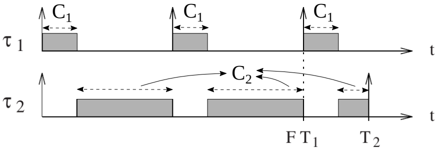
- Consider a task set Γ made of 2 periodic tasks τ_1 and τ_2 with $T_1 < T_2$
- Number of periods of τ_1 entirely contained in τ_2: $F := \lfloor T_2/T_1 \rfloor$
- Case 1 (left): $C_1 < T_2 - F\,T_1$ (i.e., all the jobs of τ_1 released within $[0, T_2)$ are completed before the second job of τ_2 is released)
- Case 2 (right): $C_1 \geq T_2 - F\,T_1$ (i.e., some job of τ_1 released within $[0, T_2)$ is not completed before the second job of τ_2 is released)
- In both cases, we maximize C_2 and we derive U_{ub} as a function of C_1

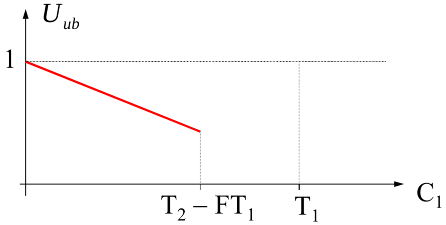




Images adapted from "Hard real-time computing systems" by Prof. G. Buttazzo

- Case 1: $C_1 < T_2 - F\,T_1$ (i.e., all the jobs of τ_1 released within $[0, T_2)$ are completed before the second job of τ_2 is released)
  - $C_2^{\max} = T_2 - (F+1)C_1$
  - $U_{\text{ub}} := \frac{C_1}{T_1} + \frac{C_2^{\max}}{T_2} = \frac{C_1}{T_1} + 1 - \frac{C_1}{T_2}(F+1) = 1 + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - (F+1)\right)$
  - $\frac{T_2}{T_1} - (F+1) \leq 0$ given that $F := \lfloor T_2/T_1 \rfloor \Rightarrow$
    $\Rightarrow U_{\text{ub}}$ decreases with $C_1 \Rightarrow$
    $\Rightarrow$ The minimum value of U_{ub} occurs for $C_1 = T_2 - F\,T_1$

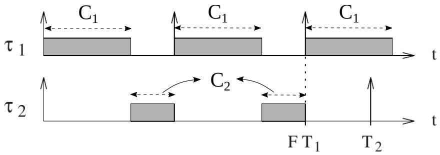


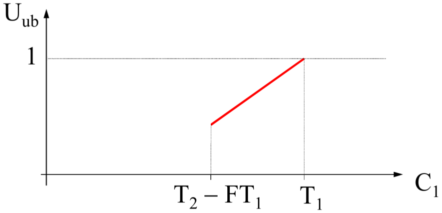


Images adapted from Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

RM guarantee test: proof for 2 tasks (3/5)

- Case 2: $C_1 \geq T_2 - F\,T_1$ (i.e., some job of τ_1 released within $[0, T_2)$ is not completed before the second job of τ_2 is released)
  - $C_2^{\max} = F(T_1 - C_1)$
  - $U_{\text{ub}} := \dfrac{C_1}{T_1} + \dfrac{F(T_1 - C_1)}{T_2} = F\dfrac{T_1}{T_2} + \dfrac{C_1}{T_2}\left(\dfrac{T_2}{T_1} - F\right)$
  - $\dfrac{T_2}{T_1} - F \geq 0$ given that $F := \lfloor T_2/T_1 \rfloor \Rightarrow$
    $\Rightarrow U_{\text{ub}}$ increases with $C_1 \Rightarrow$
    $\Rightarrow$ The minimum value of U_{ub} occurs for $C_1 = T_2 - F\,T_1$





Images adapted from Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

RM guarantee test: proof for 2 tasks (4/5)

- Compute the minimum value $U_{\text{ub}}^{\min,C_1}$ of U_{ub} with respect to C_1:

$$U_{\text{ub}}^{\min,C_1} = U_{\text{ub}}\,\big|_{C_1=T_2-F\,T_1} = F\frac{T_1}{T_2} + \frac{T_2-F\,T_1}{T_2}\left(\frac{T_2}{T_1}-F\right) =$$

$$= F\frac{T_1}{T_2} + \left(1-F\frac{T_1}{T_2}\right)\left(\frac{T_2}{T_1}-F\right) = F\frac{T_1}{T_2} + \frac{T_1}{T_2}\frac{T_2}{T_1}\left(1-F\frac{T_1}{T_2}\right)\left(\frac{T_2}{T_1}-F\right) =$$

$$= F\frac{T_1}{T_2} + \frac{T_1}{T_2}\left(\frac{T_2}{T_1}-F\right)\left(\frac{T_2}{T_1}-F\right) = \frac{T_1}{T_2}\left(F+\left(\frac{T_2}{T_1}-F\right)^2\right)$$

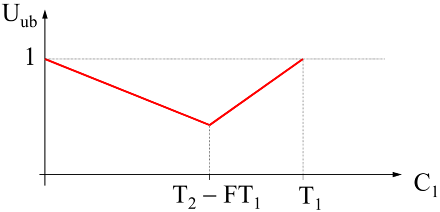


Image from Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

RM guarantee test: proof for 2 tasks (5/5)

- Decimal part of T_2/T_1: $G := T_2/T_1 - F$ where $F := \lfloor T_2/T_1 \rfloor$
- Compute $U_{\text{ub}}^{\min,C_1} = \frac{T_1}{T_2}\left(F + \left(\frac{T_2}{T_1} - F\right)^2\right)$ as a function of G:

$$U_{\text{ub}}^{\min,C_1} = \frac{F+G^2}{T_2/T_1} = \frac{F+G^2}{T_2/T_1+F-F} = \frac{F+G^2}{F+G}$$
$$= \frac{F+G-G+G^2}{F+G} = 1 - \frac{G(1-G)}{F+G}$$

$0 \leq G < 1 \Rightarrow G(1-G) \geq 0 \Rightarrow U_{\text{ub}}^{\min,C_1}$ increases with F

- Compute the minimum value $U_{\text{ub}}^{\min,C_1,F}$ of $U_{\text{ub}}^{\min,C_1}$ with respect to F:

$$U_{\text{ub}}^{\min,C_1,F} = U_{\text{ub}}^{\min,C_1}\,|_{F=1} = \frac{T_1}{T_2}\left(1 + \left(\frac{T_2}{T_1} - 1\right)^2\right) = \frac{k^2-2k+2}{k} \text{ with } k = \frac{T_2}{T_1}$$

- Compute U_{lub} as the minimum value of $U_{\text{ub}}^{\min,C_1,F}$ with respect to k:

$$\frac{dU_{\text{ub}}^{\min,C_1,F}}{dk} = \frac{(2k-2)k-(k^2-2k+2)}{k} = \frac{k^2-2}{k^2} = 0 \text{ for } k = \sqrt{2} \Rightarrow$$

$$\Rightarrow U_{\text{lub}} = U_{\text{ub}}^{\min,C_1,F}\,|_{k=\sqrt{2}} = 2(\sqrt{2}-1) \simeq 0.83$$

RM guarantee test: the case of tasks with harmonic periods

- Consider two periodic tasks τ_1 and τ_2 with **harmonic periods** $T_1 < T_2$
  - $\frac{T_2}{T_1} \in \mathbb{N}$ by definition of harmonic periods and by the fact that $T_1 < T_2 \Rightarrow$
    $\Rightarrow F := \left\lfloor \frac{T_2}{T_1} \right\rfloor = \frac{T_2}{T_1} \Rightarrow U_{\text{lub}} = \frac{T_1}{T_2} \left(F + \left(\frac{T_2}{T_1} - F \right)^2 \right) = 1$
  - Example: $C_1 = 2$, $T_1 = 4$, $C_2 = 4$, $T_2 = 8 \Rightarrow U = \frac{2}{4} + \frac{4}{8} = 1$

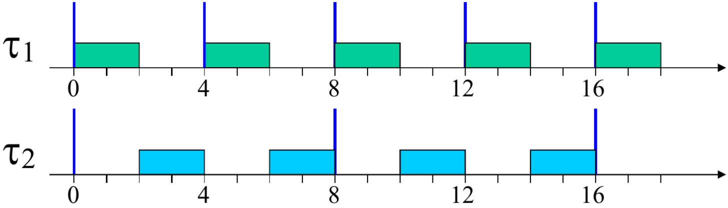
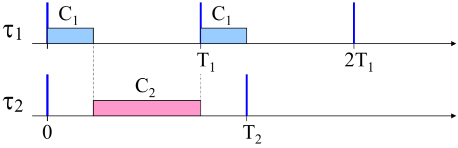
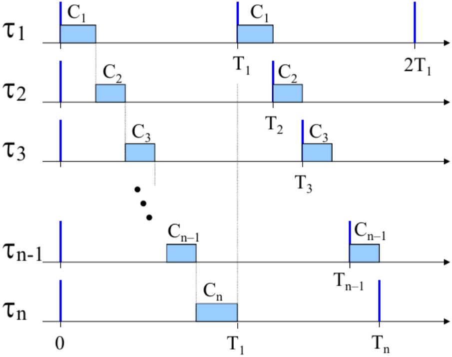


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

RM guarantee test: proof for n tasks (1/3)

- Worst case conditions for the schedulabiliy of 2 tasks with $T_1 < T_2$
  - $C_1 = T_2 - F T_1$ and $F = 1 \Rightarrow T_2 < 2T_1$ (given that $F = 1$),
    $C_1 = T_2 - FT_1 = T_2 - T_1$, $C_2 = F(T_1 - C_1) = T_1 - C_1 = T_1 - (T_2 - T_1) = 2T_1 - T_2$
- Worst case conditions for the schedulabiliy of n tasks with $T_1 < T_2 < \cdots < T_n$
  - $T_n < 2T_1$, $C_1 = T_2 - T_1$, $C_2 = T_3 - T_2$, . . . ,
    $C_n = T_1 - (\sum_{i=1}^{n-1} C_i) = T_1 - (T_2 - T_1) - (T_3 - T_2) \cdots - (T_n - T_{n-1}) = 2T_1 - T_n$





Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

RM guarantee test: proof for n tasks (2/3)

- Compute the upper bound on U based on worst case schedulability conditions

$$U_{\mathrm{ub}} = \sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{T_2 - T_1}{T_1} + \frac{T_3 - T_2}{T_2} + \ldots + \frac{T_n - T_{n-1}}{T_{n-1}} + \frac{2T_1 - T_n}{T_n}$$

$$= \frac{T_2}{T_1} + \frac{T_3}{T_2} + \ldots + \frac{T_n}{T_{n-1}} + \frac{2T_1}{T_n} - n$$

- Define $R_i = \dfrac{T_{i+1}}{T_i}\ \forall\, i \in \{1, \ldots, n-1\}$ and note that $\prod_{i=1}^{n-1} R_i = \dfrac{T_n}{T_1}$

$$U_{\mathrm{ub}} = \sum_{i=1}^{n} R_i + \frac{2}{R_1\, R_2 \cdots R_{n-1}} - n$$

- Minimize U_{ub} with respect to $R_i\ \forall\, i \in \{1, \ldots, n-1\}$

$$\frac{\partial U_{\mathrm{ub}}}{\partial R_i} = 1 - \frac{2}{R_i^2} \frac{1}{R_1\, R_2 \cdots R_{i-1}\, R_{i+1} \cdots R_n} = 1 - \frac{2}{R_i\, P} \text{ where } P := \prod_{i=1}^{n-1} R_i \Rightarrow$$

$$\Rightarrow \frac{\partial U_{\mathrm{ub}}}{\partial R_i} = 0 \text{ for } R_i\, P = 2 \Rightarrow U_{\mathrm{ub}} \text{ is minimum if } R_i\, P = 2\ \forall\, i \in \{1, \ldots, n-1\}$$

- $R_i\, P = 2\ \forall\, i \in \{1, \ldots, n-1\}$ if $R_i = 2^{\frac{1}{n}}$, which yields $P = (2^{\frac{1}{n}})^{n-1}$

RM guarantee test: proof for n tasks (3/3)

- Compute the least upper bound on U

$$U_{\text{lub}} = \sum_{i=1}^{n} R_i + \frac{2}{P} - n \Big|_{R=2^{\frac{1}{n}},P=(2^{\frac{1}{n}})^{n-1}} = (n-1)2^{\frac{1}{n}} + \frac{2}{(2^{\frac{1}{n}})^{n-1}} - n =$$

$$= n\,2^{\frac{1}{n}} - 2^{\frac{1}{n}} + \frac{2}{(2^{\frac{1}{n}})^{n-1}} - n = n\,2^{\frac{1}{n}} - 2^{\frac{1}{n}} + \frac{2}{(2^{\frac{n-1}{n}})} - n$$

$$= n\,2^{\frac{1}{n}} - 2^{\frac{1}{n}} + \frac{2}{(2^{1-\frac{1}{n}})} - n = n\,2^{\frac{1}{n}} - 2^{\frac{1}{n}} + 2^{\frac{1}{n}} - n = n\,(2^{\frac{1}{n}} - 1)$$

n	U_{lub}
1	1.000
2	0.828
3	0.780
4	0.757
5	0.743
10	0.718
20	0.705
50	0.698
100	0.696
1000	0.693

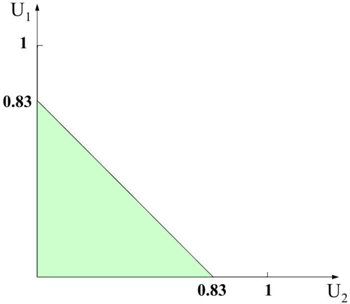


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

RM hyperbolic bound (Bini et al, 2003)

Theorem

If $\prod_{i=1}^{n}(U_i+1) \leq 2$ *for a set* Γ *of* n *pure periodic tasks* $\Rightarrow \Gamma$ *is schedulable by RM*

- Worst case conditions for the schedulabiliy of n tasks with $T_1 < T_2 < \dots < T_n$: $T_n < 2T_1$, $C_1 = T_2 - T_1$, $C_2 = T_3 - T_2$, ..., $C_n = T_1 - (\sum_{i=1}^{n-1} C_i) = 2T_1 - T_n$

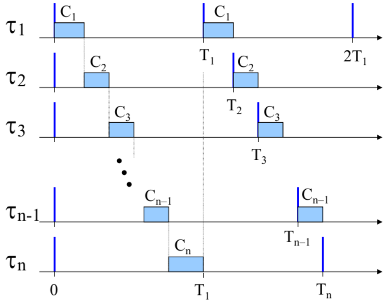


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

E. Bini, G. C. Buttazzo, G. M. Buttazzo, "Rate monotonic scheduling: The hyperbolic bound", IEEE Trans. on Comp., 52(7):933–942, July 2003

RM hyperbolic bound: proof

- $R_i := \dfrac{T_{i+1}}{T_i} \; \forall\, i \in \{1, \ldots, n-1\} \Rightarrow R_i = \dfrac{T_{i+1} - T_i + T_i}{T_i} = U_i + 1$ and $\prod_{i=1}^{n-1} R_i = \dfrac{T_n}{T_1}$

Proof.

- The worst case schedulability condition is $\sum_{i=1}^{n} C_i \le T_1 \Rightarrow$

$\Rightarrow \sum_{i=1}^{n-1} C_i + C_n \le T_1 \Rightarrow C_n \le T_1 - \sum_{i=1}^{n-1} C_i \Rightarrow C_n \le 2T_1 - T_n$ given that $\sum_{i=1}^{n-1} C_i = T_n - T_1$ by the worst case schedulability conditions $\Rightarrow$

$$\Rightarrow U_n \le \frac{2T_1}{T_n} - 1 \Rightarrow U_n + 1 \le \frac{2T_1}{T_n} = \frac{2}{\prod_{i=1}^{n-1} R_i} = \frac{2}{\prod_{i=1}^{n-1} (U_i + 1)} \Rightarrow$$

$$\Rightarrow \prod_{i=1}^{n} (U_i + 1) \le 2$$

□

RM: Liu & Layland bound vs hyperbolic bound

- The hyperbolic bound is **tight**, i.e., if the hyperbolic bound is not satisfied ⇒ ⇒ an unfeasible RM schedule exists with that processor utilization
- **Gain** achieved by hyperbolic (HB) bound over the Liu & Layland (LL) bound
  - Ratio between the hypervolumes in the U-space of the task sets found schedulable by the HB bound and by the LL bound
  - Increases with n and tends to $\sqrt{2}$ when n tends to infinity

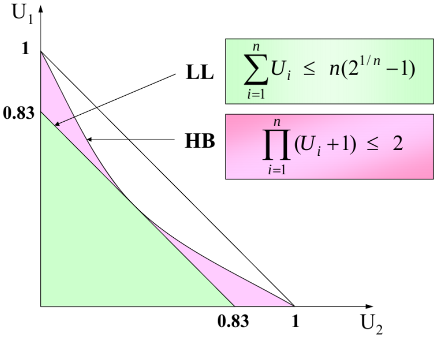
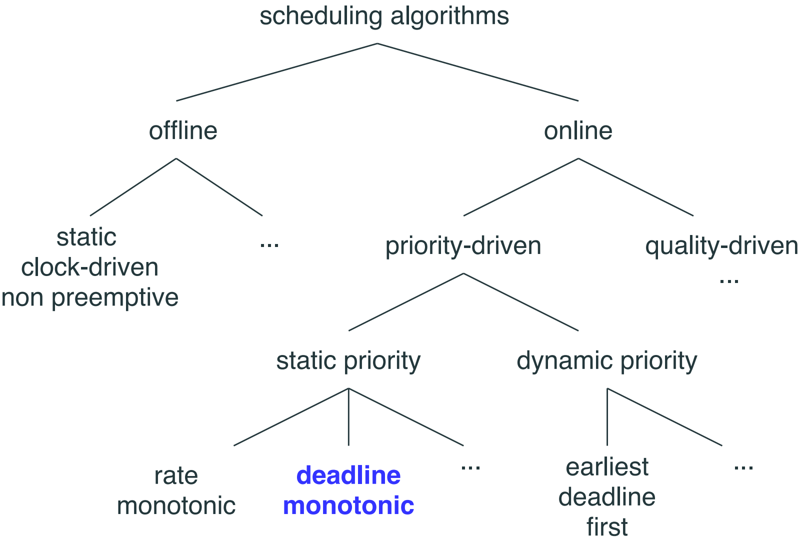


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Deadline Monotonic (DM) scheduling



DM: salient traits (Leung & Whitehead, 1982)

- Extension of RM to periodic tasks with **constrained deadlines** (i.e., $D_i \leq T_i$)
- Preemptive **static** online scheduling algorithm
- A task has a **fixed priority** inversely proportional to its **relative** deadline
- Example: priority τ_1 > priority τ_2

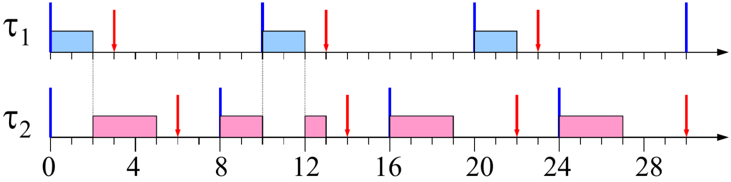


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

J. Leung, J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks", Performance Evaluation, 2(4):237–250, 1982

DM: optimality

Theorem

DM is **optimal** *in the sense of* **feasibility** *among all* **fixed priority** *algorithms (for scheduling of periodic task with constrained deadlines):*

- *If a fixed priority schedule is feasible for a task set* $\Gamma \Rightarrow$
  $\Rightarrow$ *The DM schedule is feasible for* Γ
- *If the DM schedule is not feasible for a task set* $\Gamma \Rightarrow$
  $\Rightarrow$ *No fixed priority schedule is feasible for* Γ

- Note that the two statements are equivalent ($a \Rightarrow b$ if and only if $\neg b \Rightarrow \neg a$)

DM: problem with the LL bound and the HB bound

- Use the LL bound and the HB bound by replacing periods with deadlines: the processor workload is overestimated ⇒ the test result is too pessimistic!
- Example where tests based on the processor utilization are not conclusive:
  - The LL bound is not satisfied: $\dfrac{C_1}{D_1} + \dfrac{C_2}{D_2} = \dfrac{2}{3} + \dfrac{3}{6} = \dfrac{7}{6} > 1$
  - The HB bound is not satisfied: $\left(\dfrac{C_1}{D_1} + 1\right)\left(\dfrac{C_2}{D_2} + 1\right) = \left(\dfrac{2}{3} + 1\right)\left(\dfrac{3}{6} + 1\right) = \dfrac{5}{2} > 2$
  - But the task set is schedulable!

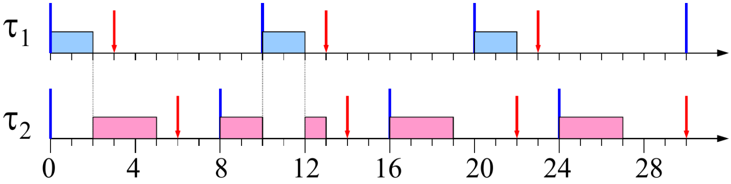


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

DM: response time analysis (Audlsey et al, 1993)

- For each task τ_i:
  1. Compute the **interference** I_i due to higher priority tasks in the interval $[0, R_i]$:

     $$I_i = \sum_{\tau_k \mid D_k < D_i} z_{ik} C_k \text{ where } z_{ik} := \text{number of releases of } \tau_k \text{ in } [0, R_i] \Rightarrow$$

     $$\Rightarrow I_i = \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k \text{ assuming tasks ordered by increasing relative deadline}$$

  2. Compute the **response time** R_i:

     $$R_i = C_i + I_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

  3. Verify that $R_i \leq D_i$
- The worst case response time is the smallest value satisfying the equation

N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, A. J. Wellings, "Applying new scheduling theory to static priority preemptive scheduling", Software Engineering Journal, 8(5):284–292, Sept. 1993

DM: response time analysis - iterative solution

- Iterative solution to derive the smallest R_i satisfying $R_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$
  - Step 0: $R_i^{(0)} = \sum_{k=1}^{i} C_k$ (min response time with synchronous task arrivals)
  - Step $j > 0$: $R_i^{(j)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(j-1)}}{T_k} \right\rceil C_k$
  - Iterate while $\left(R_i^{(j)} > R_i^{(j-1)} \;\&\&\; R_i^{(j)} \leq D_i \right) \; \forall\, j > 0$

input : A set Γ of n periodic tasks $\tau_1, \ldots, \tau_n$ with constrained deadlines
output : TRUE if the task set Γ is schedulable by DM, FALSE otherwise

foreach *task* $\tau_i \in \Gamma$ **do**
    $I_i = \sum_{k=1}^{i-1} C_k$
    do
        $I_i = R_i + C_i$
        if $R_i > D_i$ **then**
            return FALSE
        end
        $I_i = \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$
    while $I_i + C_i > R_i$;
end
return TRUE

DM: response time analysis - complexity

- **Pseudo-polynomial** complexity $O(n \cdot N)$, i.e., polynomial complexity in the number of elements of an input set and in the values of the input set
  - Polynomial complexity in the number n of tasks
  - Polynomial complexity in the maximum number N of iterations per task, which mainly depends on the relations among task periods

```
input  : A set Γ of n periodic tasks τ_1, ..., τ_n with constrained deadlines
output : TRUE if the task set Γ is schedulable by DM, FALSE otherwise

foreach task τ_i ∈ Γ do
    I_i = Σ_{k=1}^{i-1} C_k
    do
        R_i = I_i + C_i
        if R_i > D_i then
            return FALSE
        end
        I_i = Σ_{k=1}^{i-1} ⌈R_i / T_k⌉ C_k
    while I_i + C_i > R_i;
end
return TRUE
```

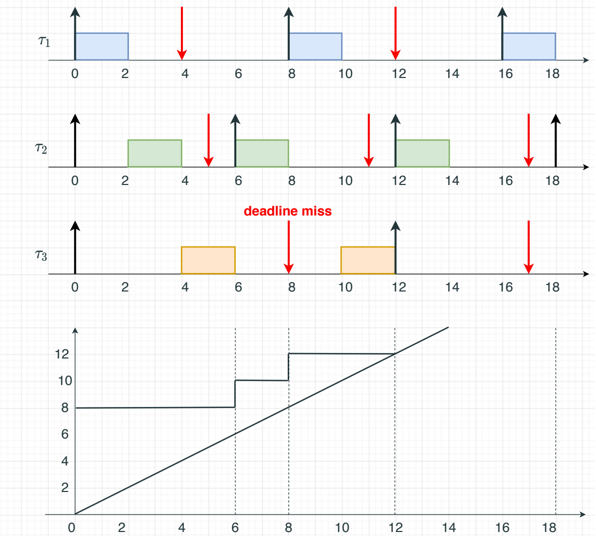
DM: example of an unfeasible schedule (1/2)

task τ_i	C_i	T_i	D_i
τ_1	2	8	4
τ_2	2	6	5
τ_3	4	12	8

- $R_1^{(0)} = C_1 = 2 < D_1$; $R_1^{(1)} = C_1 = 2 < D_1 \Rightarrow R_1 = 2$
- $R_2^{(0)} = C_1 + C_2 = 4 < D_2$; $R_2^{(1)} = C_2 + \lceil R_2^{(0)}/T_1 \rceil C_1 = C_2 + C_1 = 4 < D_2 \Rightarrow R_2 = 4$ (note that the task response is the maximum among the job response times)
- $R_3^{(0)} = C_1 + C_2 + C_3 = 8 = D_3$; $R_3^{(1)} = C_3 + \lceil R_3^{(0)}/T_1 \rceil C_1 + \lceil R_3^{(0)}/T_2 \rceil C_2 =$ $= C_3 + C_1 + 2C_2 = 10 > D_3 \Rightarrow$ the DM schedule is **unfeasible** for the task set

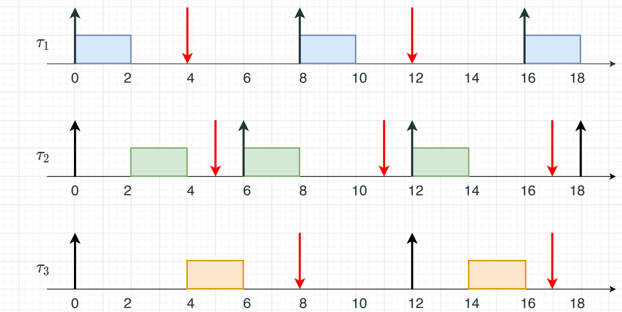
DM: example of an unfeasible schedule (2/2)

- The estimated response time increases at each task release: $R_3 = 12$



DM: example of a feasible schedule (1/2)

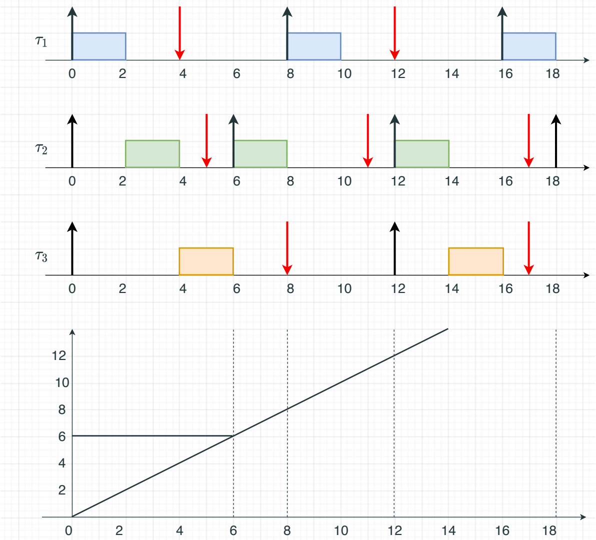
task τ_i	C_i	T_i	D_i
τ_1	2	8	4
τ_2	2	6	5
τ_3	2	12	8



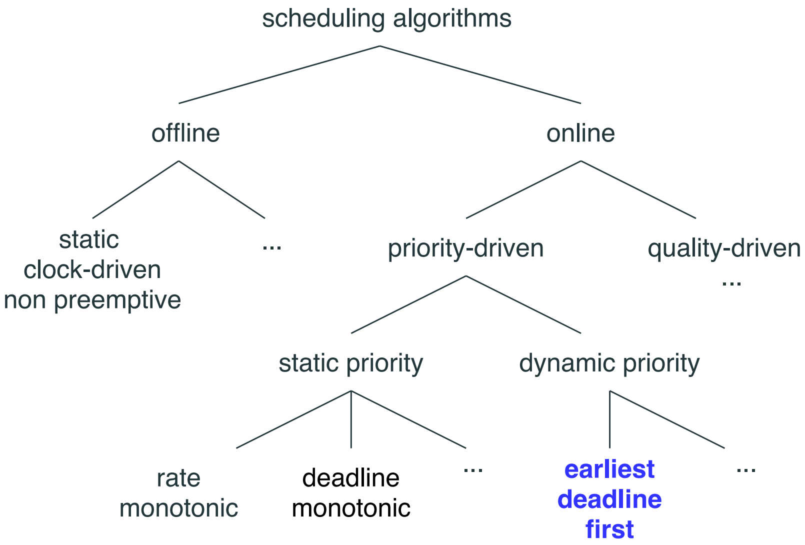
- The response times of τ_1 and τ_2 remain the same: $R_1 = 2 < D_1$, $R_2 = 4 < D_2$
- $R_3^{(0)} = C_1 + C_2 + C_3 = 6 < D_3$; $R_3^{(1)} = C_3 + \lceil R_3^{(0)}/T_1 \rceil C_1 + \lceil R_3^{(0)}/T_2 \rceil C_2 =$ $= C_3 + C_1 + C_2 = 6 < D_3 \Rightarrow$ the DM schedule is **feasible** for the task set

DM: example of a feasible schedule (2/2)

- The estimated response time increases at each task release: $R_3 = 6$



Earliest Deadline First (EDF) scheduling



EDF: salient traits

- Preemptive **dynamic** online scheduling algorithm
- Addressing scheduling of pure periodic tasks (i.e., $D_i = T_i\ \forall$ task τ_i)
- A task has a **dynamic priority** inversely proportional to its **absolute** deadline
- Example: $C_1 = 3$, $T_1 = D_1 = 6$; $C_2 = 4$, $T_1 = D_1 = 9$

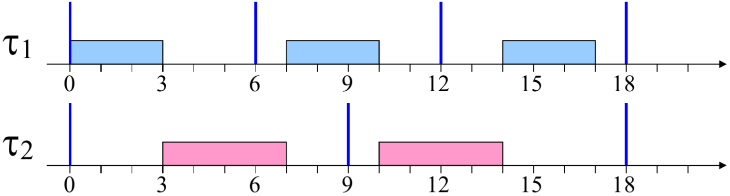
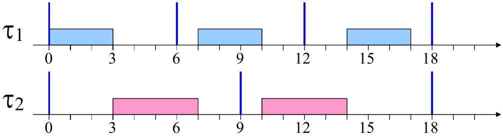


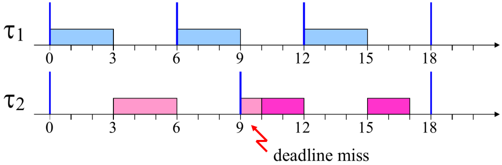
Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

EDF schedule vs RM schedule

- The EDF schedule is feasible for the task set:



- The RM schedule is not feasible for the task set:



- Tests based on the Liu & Layland and hyperbolic bounds are inconclusive, i.e., they do not permit assessing the feasibility of the RM schedule:
  - $U = C_1/T_1 + C_2/T_2 = 3/6 + 4/9 = 0.944 > 2(\sqrt{2} - 1) = 0.828$
  - $(U_1 + 1)(U_2 + 1) = (3/6 + 1)(4/9 + 1) = 13/6 > 2$

Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

EDF guarantee test (1/2) (Liu & Layland, 1973)

Theorem

A set of pure periodic tasks is schedulable by EDF if and only if $U \leq 1$

- The test is **necessary and sufficient**
- Polynomial complexity $O(n)$ with respect to the number n of tasks
- Necessity: if a set of pure periodic tasks is schedulable by EDF $\Rightarrow U \leq 1$ (i.e., if $U > 1 \Rightarrow$ a set of pure periodic tasks is not schedulable by EDF)
- Sufficiency: if $U \leq 1 \Rightarrow$ a set of pure periodic tasks is schedulable by EDF (i.e., if a set of pure periodic tasks is not schedulable by EDF $\Rightarrow U > 1$)

Proof of necessity.

- If $U > 1 \Rightarrow U\,T > T$ given that $T = T_1 T_2 \cdots T_n > 0 \Rightarrow$
  $\Rightarrow \sum_{i=1}^{n} \frac{C_i}{T_i} T > T$ by definition of $U \Rightarrow \sum_{i=1}^{n} \frac{T}{T_i} C_i > T \Rightarrow$
  $\Rightarrow$ the total demand in $[0, T)$ is larger than $T \Rightarrow$ the task set is not feasible

$\square$

C. L. Liu, J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", Journal of the ACM, 20(1), 1973

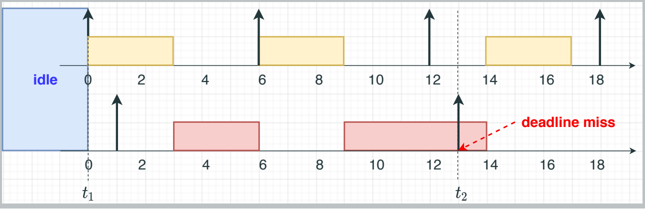
EDF guarantee test (2/2) (Liu & Layland, 1973)

Proof of sufficiency (by contradiction).

- We assume that the task set is not schedulable by EDF and $U < 1$:
  - $t_2 :=$ first time instant at which a deadline is missed
  - $[t_1, t_2] :=$ longest interval of continuous utilization before t_2 during which only jobs $\tau_{i,k}$ with arrival time $a_{i,k} \geq t_1$ and absolute deadline $d_{i,k} \leq t_2$ are executed
  - $C_p(t_1, t_2) :=$ total processor demand during the interval $[t_1, t_2] \Rightarrow$

$$C_p(t_1, t_2) = \sum_{\tau_{i,k} | a_{i,k} \geq t_1 \wedge d_{i,k} \leq t_2} C_i = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \leq \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1) U$$

- $C_p(t_1, t_2) > t_2 - t_1$ given that a deadline is missed at $t_2 \Rightarrow$
  $\Rightarrow t_2 - t_1 < C_p(t_1, t_2) \leq (t_2 - t_1) U \Rightarrow U > 1$, which is a contradiction



□

EDF optimality (Dertouzos, 1974)

Theorem

EDF is **optimal** *in the sense of* **feasibility** *among* **all algorithms***:*

- *If a schedule is feasible for a task set* $\Gamma \Rightarrow$
  $\Rightarrow$ *The EDF schedule is feasible for* Γ
- *If the EDF schedule is not feasible for a task set* $\Gamma \Rightarrow$
  $\Rightarrow$ *No schedule is feasible for* Γ

- Note that the two statements are equivalent ($a \Rightarrow b$ if and only if $\neg b \Rightarrow \neg a$)
- Note that this result is **independent of periodicity**

M. L. Dertouzos, "Control robotics: the procedural control of physical processes", Information Processing, 74, 1974

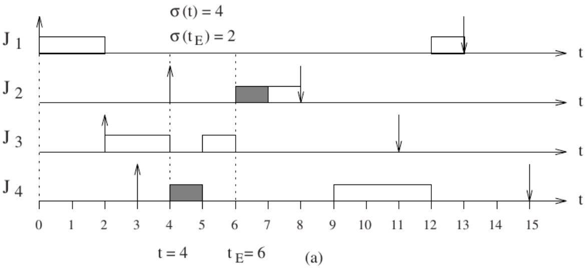
EDF optimality: proof (1/3)

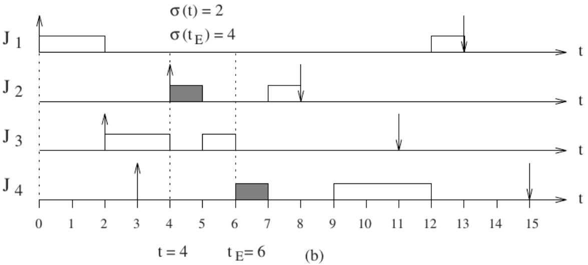
- A feasible schedule σ for task set Γ is divided into time slices of one time unit (note that the time slice duration could be arbitrary small):
  - $\sigma(t)$:= task executing during the time slice $[t, t+1)$
  - $E(t)$:= index of the task with minimum absolute deadline at t
  - t_E := time $\geq t$ at which $\tau_{E(t)}$ is executed first
  - $d_{\max} := \max_{i \in \{1,\ldots,n\}} \{d_i\}$ (maximum absolute deadline)
- Schedule σ is transformed into an EDF schedule σ_{EDF}:

```
input   : A feasible schedule σ for task set Γ
output  : An EDF schedule σ_EDF for task set Γ

foreach t ∈ {0, 1, ..., d_max − 1} do
    if σ(t) ≠ σ_EDF(t) then
        σ(t_E) = σ(t) // [t_E, t_E + 1) allocated to the task executed during [t, t + 1)
        σ(t) = E(t)  // [t, t + 1) allocated to the task with min absolute deadline at t
    end
end
return TRUE
```

EDF optimality: proof (2/3)





Images from "Hard real-time computing systems" by Prof. G. Buttazzo

EDF optimality: proof (3/3)

- A transposition preserves schedulability, i.e., σ_{EDF} is feasible for Γ
  - If a time slice of task τ_i is anticipated $\Rightarrow$ The feasibility of τ_i is preserved
  - If a time slice of task τ_i is postponed at t_E $\Rightarrow$
    $\Rightarrow t_E + 1 \leq d_E$ with $d_E :=$ earliest absolute deadline at t, since σ is feasible $\Rightarrow$
    $\Rightarrow t_E + 1 \leq d_E \leq d_i$ $\forall$ task τ_i by definition of d_E $\Rightarrow$
    $\Rightarrow$ the time slice postponed at t_E is schedulable

EDF optimality wrt minimizing max lateness (Jackson, 1955)

Theorem

Given a set of n independent tasks, any algorithm that executes the tasks in order of non-decreasing absolute deadlines is optimal with respect to minimizing the maximum lateness $L_{\max} := \max_{i \in \{1,\ldots,n\}} \{L_i\}$

- Note that this result is **independent of periodicity**
- If an algorithm minimizes $L_{\max}$ ⇒ It is optimal in the sense of feasibility (the opposite is not true)

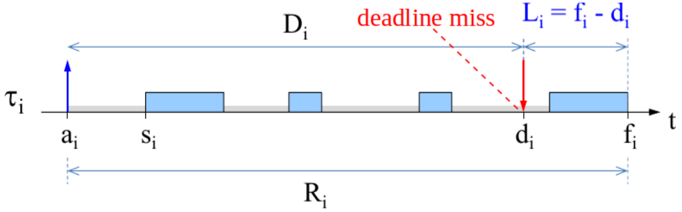


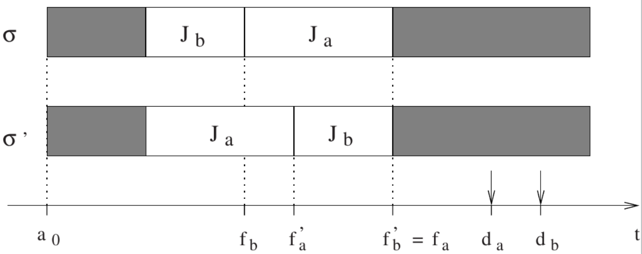
Image adapted from Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

J. R. Jackson, "Scheduling a production line to minimize maximum tardiness", Management Science Research Proj. 43, Univ. of California, LA, USA, 1955

EDF optimality wrt minimizing max lateness: proof

Proof.

- Consider the transposition by Dertouzos (see EDF optimality proof)
- A transposition between two slices Σ_a and Σ_b cannot increase $L_{\max}$
  - Let Σ_a be anticipated (i.e., $f'_a < f_a$) and Σ_b be postponed (i.e., $f'_b > f_b$)
  - If $L'_a \geq L'_b \Rightarrow L'_{\max} = L'_a = f'_a - d_a < f_a - d_a = L_{\max}$ given that $f'_a < f_a$
  - If $L'_a \leq L'_b \Rightarrow L'_{\max} = L'_b = f'_b - d_b = f_a - d_b < f_a - d_a = L_{\max}$ given that $d_a < d_b$
- By a finite number of transpositions, σ can be transformed in σ_{EDF}, and, given that the maximum lateness cannot increase, σ_{EDF} is optimal



□

Image from "Hard real-time computing systems" by Prof. G. Buttazzo

EDF with constrained deadlines (Baruah et al, 1990)

Processor demand criterion

A set of n periodic tasks $\{\tau_1, \dots, \tau_n\}$ with $D_i \le T_i$ $\forall$ task τ_i is schedulable by EDF if and only if in any interval of time $[t_1, t_2]$ the processor demand $g(t_1, t_2)$ does not exceed the available time, i.e., $g(t_1, t_2) \le t_2 - t_1$ $\forall$ t_1, t_2 with $t_1 < t_2$

- The processor demand in the time interval $[t_1, t_2]$ is the processing time requested by jobs activated in $[t_1, t_2]$ with absolute deadline $\le t_2$:

$$g(t_1, t_2) = \sum_{i=1}^{n} \eta_i(t_1, t_2) C_i$$

where $\eta_i(t_1, t_2)$ is the no. of jobs of τ_i contributing to demand in $[t_1, t_2]$:

  - $\eta_i(t_1, t_2) := |\{\tau_{i,k} \mid a_{i,k} \in [t_1, t_2] \wedge d_{i,k} \le t_2\}| = \max\{0, K_2^i - K_1^i\}$
  - $K_2^i := |\{\tau_{i,k} \mid a_{i,k} \in [\phi_i, t_2] \wedge d_{i,k} \le t_2\}| = \lfloor t_2 + T_i - D_i - \phi_i / T_i \rfloor$
  - $K_1^i := |\{\tau_{i,k} \mid a_{i,k} \in [\phi_i, t_1]\}| = \lceil t_1 - \phi_i / T_i \rceil$

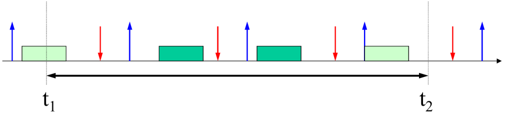
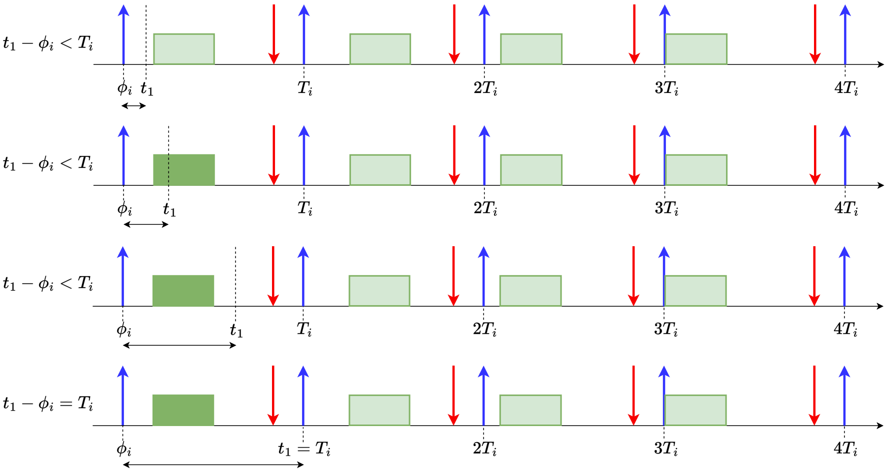


Image by Prof. G. Buttazzo,
Scuola Superiore Sant'Anna, Pisa

S. K. Baruah, L. E. Rosier, R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor", Journal of Real-Time Systems, 2, 1990

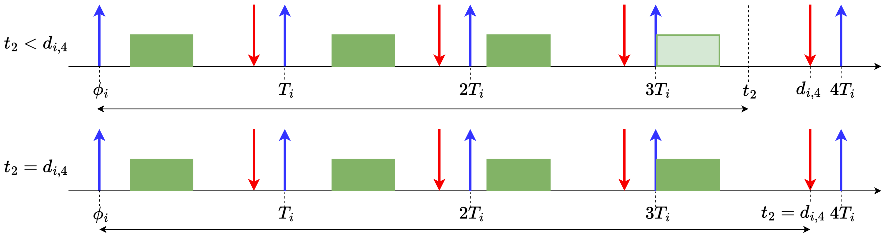
EDF with constrained deadlines: evaluation of K_1

- $K_1^i := |\{\tau_{i,k} \mid a_{i,k} \in [\phi_i, t_1]\}| = \lceil t_1 - \phi_i / T_i \rceil$



EDF with constrained deadlines: evaluation of K_2

- $K_2^i := \left|\{\tau_{i,k} \mid a_{i,k} \in [\phi_i, t_2] \wedge d_{i,k} \le t_2\}\right| = \lfloor t_2 + T_i - D_i - \phi_i / T_i \rfloor$



EDF with constrained deadlines: demand bound function

- Worst case scenario: all tasks activated at time 0 (i.e., $\phi_i = 0\ \forall$ task τ_i):

$$\text{dbf}(t) := g(0,t) = \sum_{i=1}^{n} \eta_i(0,t) C_i = \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i$$

Remark

A synchronous set of n periodic tasks $\{\tau_1, \ldots, \tau_n\}$ with $D_i \le T_i\ \forall$ task τ_i is schedulable by EDF if and only if $\text{dbf}(t) \le t\ \forall\, t > 0$

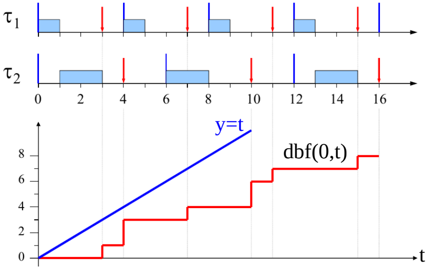


Image adapted from Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

EDF with constrained deadlines: bounding complexity (1/2)

1. Synchronous set of periodic tasks $\Rightarrow$ Verify the criterion only for $t \leq H$ (where H is the hyper-period of the task-set)

2. dbf(t) is a step function that increases when t equals an absolute deadline $\Rightarrow$
   $\Rightarrow$ if dbf$(t) < t$ for $t = d_i$ then dbf$(t) < t \ \forall\, t \mid d_i \leq t < d_{i+1}$ $\Rightarrow$
   $\Rightarrow$ Verify the criterion only for values of t equal to absolute deadlines

3. Verify the criterion at least until $d_{\max} := \max_i\{d_i\} \leq H$

EDF with constrained deadlines: bounding complexity (2/2)

4. $\mathrm{dbf}(t) = \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i \leq \sum_{i=1}^{n} \frac{t + T_i - D_i}{T_i} C_i = \sum_{i=1}^{n} (T_i - D_i) U_i + t\, U \Rightarrow$

$\Rightarrow G(0,t) := \sum_{i=1}^{n} (T_i - D_i) U_i + t\, U$ is an increasing function with slope $U \Rightarrow$

$\Rightarrow$ if $U < 1$ then $\exists\, t^\star \mid G(0, t^\star) = t^\star$ where $t^\star = \sum_{i=1}^{n} (T_i - D_i) U_i / (1 - U) \Rightarrow$

$\Rightarrow \mathrm{dbf}(t) \leq G(0,t) \leq t \;\; \forall\, t \geq t^\star \Rightarrow$ Verify the criterion only for $t < t^\star$

(note that the task set considered in figure below is NOT schedulable)

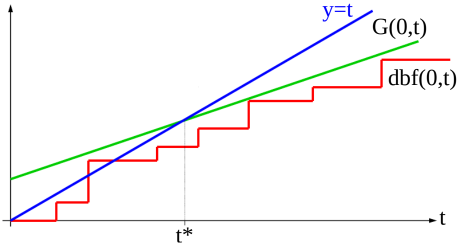


Image adapted from Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

EDF with constrained deadlines: processor demand test

- Recap: how to bound complexity?
  1. Verify the criterion only for $t \leq H$
  2. Verify the criterion only for values of t equal to absolute deadlines
  3. Verify the criterion at least until $d_{\max} := \max_i\{d_i\} \leq H$
  4. Verify the criterion only for $t < t^\star$

Processor demand test

A synchronous set of n periodic tasks $\{\tau_1, \ldots, \tau_n\}$ with $D_i \leq T_i$ $\forall$ task τ_i is schedulable by EDF if and only if $U < 1$ and $\text{dbf}(t) \leq t$ $\forall\, t \in \mathcal{D}$ where $\mathcal{D} = \{d_i \mid d_i \leq \min\{d_{\max}, t^\star\}\}$ and $t^\star = \sum_{i=1}^{n}(T_i - D_i)U_i/(1-U)$

RM vs EDF

- RM scheduling
  - Less efficient (processor utilization nearly equal to 69% in the worst case)
  - Simpler to implement in commercial Real-Time Operating Systems (RTOSs)
  - More predictable during overloads (but low-priority tasks are blocked while high-priority tasks are executed at the proper rate)
- EDF scheduling
  - More efficient (processor utilization equal to 100%)
  - Lower number of preemptions $\Rightarrow$ Lower overhead due to context switches
  - More flexible during overloads (all tasks executed at slower rate)
  - Better responsiveness in handling aperiodic tasks
  - More uniform jitter control

Periodic task scheduling: summary (1/2)

- 3 scheduling approaches
  1. offline (timeline scheduling)
  2. online static priority (RM, DM)
  3. online dynamic priority (EDF)

- 3 schedulability analysis techniques
  1. Utilization based analysis
     - LL bound for RM: $U \leq n(2^{1/n} - 1)$ (sufficient)
     - HB bound for RM: $\prod_{i=1}^{n}(U_i + 1) \leq 2$ (sufficient)
     - LL bound for harmonic task sets: $U \leq 1$ (necessary & sufficient)
     - EDF bound: $U \leq 1$ (necessary & sufficient)
     - Polyomial complexity $O(n)$
  2. Response time analysis
     - $R_i \leq D_i \;\forall\, i \in \{1, \ldots, n\}$ with $R_i = C_i + \sum_{k=1}^{i-1} \lceil R_i / T_k \rceil C_k$ (necessary & sufficient)
     - Pseudo-polyomial complexity
  3. Processor demand analysis
     - dbf$(t) \leq t \;\forall\, t \in \mathcal{D}$ (necessary & sufficient)
     - Pseudo-polyomial complexity

Periodic task scheduling: summary (2/2)

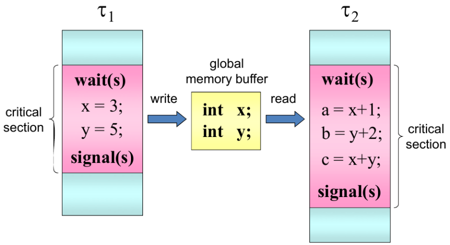
	$D_i = T_i \ \forall\, i \in \{1, \ldots, n\}$	$\exists\, i \in \{1, \ldots, n\} \mid D_i < T_i$
RM	LL bound for harmonic task sets LL bound, HB bound response time analysis	response time analysis
EDF	EDF bound	processor demand approach

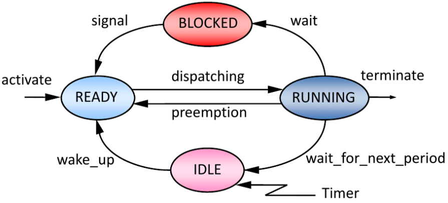
- 3 schedulability analysis techniques
  1. Utilization based analysis
     - LL bound for RM: $U \leq n(2^{1/n} - 1)$ (sufficient)
     - HB bound for RM: $\prod_{i=1}^{n}(U_i + 1) \leq 2$ (sufficient)
     - EDF bound: $U \leq 1$ (necessary & sufficient)
     - LL bound for harmonic task sets: $U \leq 1$ (necessary & sufficient)
     - Polyomial complexity $O(n)$
  2. Response time analysis
     - $R_i \leq D_i \ \forall\, i \in \{1, \ldots, n\}$ with $R_i = C_i + \sum_{k=1}^{i-1} \lceil R_i / T_k \rceil C_k$ (necessary & sufficient)
     - Pseudo-polyomial complexity
  3. Processor demand analysis
     - dbf$(t) \leq t \ \forall\, t \in \mathcal{D}$ (necessary & sufficient)
     - Pseudo-polyomial complexity

Resource access protocols

Resources

- **Resource**: any software structure used by a task to advance its execution (e.g., variables, files, devices, main memory areas)
  - **Private** resource: dedicated to a particular task
  - **Shared** resource: can be used by multiple tasks
- **Exclusive** resource: a shared resource protected against concurrent accesses
  - **Resource access protocols**: mechanisms that guarantee mutual exclusion
  - **Critical section**: piece of code executed under mutual exclusion





Images by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Priority inversion (1/2)

- Priority inversion is a phenomenon where a **high priority** task is blocked by a **low priority** task for a time interval of **unbounded duration**
  - The **blocking time** of a task is a delay caused by lower priority tasks
  - E.g., τ_1 and τ_3 ($P_1 > P_3$) share a resource managed by a binary semaphore S
  - Blocking time of τ_1 = Time needed by τ_3 to execute the critical section

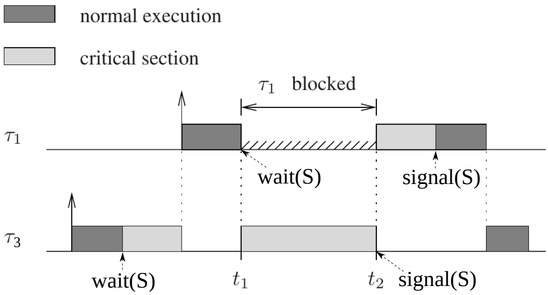
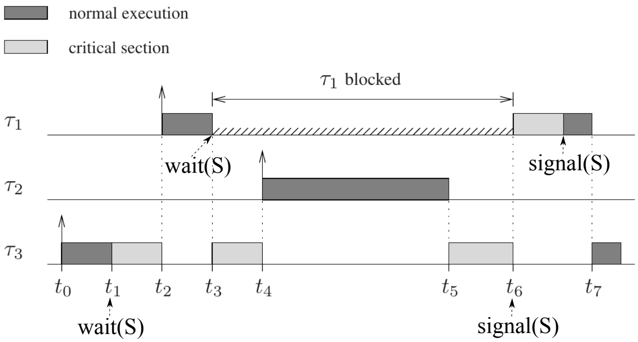


Image adapted from "Hard real-time computing systems" by Prof. G. Buttazzo

Priority inversion (2/2)

- The blocking time of the high priority task cannot be bounded by the duration of the critical section executed by the low priority task
  - E.g., Add a **mid priority** task τ_2 ($P_1 > P_2 > P_3$)
  - The maximum blocking time of τ_1 depends not only on the duration of the critical section of τ_3 but also on the WCET of τ_2



- The solution to unbounded blocking is using **resource access protocols**

Image adapted from "Hard real-time computing systems" by Prof. G. Buttazzo

Resource access protocols: problem formulation (1/2)

- Set of n **periodic tasks** $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with each task τ_i characterized by:
  - Phase Φ_i, WCET C_i, period T_i, relative deadline $D_i \leq T_i$
  - **Nominal** (static, fixed) **priority** P_i (assigned by the application developer)
  - **Dynamic** (active) **priority** $p_i \geq P_i$ (initialized to P_i)

- A set of m **shared resources** $\Psi = \{R_1, \ldots, R_m\}$
  - Each resource R_k guarded by a distinct **binary semaphore** S_k

- Assumptions
  - $\tau_1, \ldots, \tau_n$ are released upon arrival
  - $\tau_1, \ldots, \tau_n$ are subject to zero or negligible kernel overheads
  - $\tau_1, \ldots, \tau_n$ have different nominal priority
  - $\tau_1, \ldots, \tau_n$ are listed in increasing order of nominal prio (i.e., $P_1 > \ldots > P_n$)
  - $\tau_1, \ldots, \tau_n$ suspend themselves only:
    - to wait the beginning of next period
    - on locked semaphores
  - Critical sections are guarded by binary semaphores and properly nested
    - $z_{i,k}$:= critical section of task τ_i guarded by binary semaphore S_k
    - For any pair $z_{i,h}$, $z_{i,k}$, it holds that $z_{i,h} \subset z_{i,k}$ or $z_{i,k} \subset z_{i,h}$ or $z_{i,h} \cap z_{i,k} = \emptyset$

Resource access protocols: problem formulation (2/2)

- Goal: derive the **maximum blocking time** B_i that a task τ_i can experience
- Protocol key aspects
  - **Access** rule: decides whether to block and when
  - **Progress** rule: decides how to execute inside a critical section
  - **Release** rule: decides how to order the pending requests of the blocked tasks

Priority Inheritance Protocol (PIP) (Sha et al, 1990)

- **Access** rule: A task τ_i blocks at the entrance of a critical section $z_{i,k}$ if resource R_k is already held by a lower priority task τ_j
  - Task τ_i is said to be **blocked** by task τ_j
  - Blocked tasks are scheduled based on their dynamic priority
  - Blocked tasks with same priority scheduled by First Come First Served (FCFS)
- **Progress** rule: Inside a critical section associated with resource R_k a task executes with the highest priority of the tasks blocked on R_k
  - Task τ_j **inherits** the highest priority of the tasks it blocks:
    $$p_j = \max\{P_j, \max_i\{P_i \mid \tau_i \text{ is blocked on } R_k\}\}$$
  - **Transitivity** property: if τ_3 blocks τ_2 and τ_2 blocks $\tau_1 \Rightarrow p_3 = P_1$
- **Release** rule: When τ_j exits the critical section associated with resource R_k:
  - Semaphore S_k is unlocked
  - The highest priority task blocked on S_k (if any) is awakened
  - If no other task is blocked by τ_j then $p_j = P_j$ otherwise τ_j inherits the highest priority of the tasks it blocks

L. Sha, R. Rajkumar, J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", IEEE Trans. on Comp., 39(9), Sept. 1990

PIP: types of blocking

- **Direct blocking**: occurs when a high priority task blocks at the entrance of the critical section of a resource already held by a low priority task
  - Necessary to ensure the consistency of shared resources
- **Push-through blocking**: occurs when a mid priority task is blocked by a low priority task that has inherited a higher priority from a task
  - Necessary to avoid priority inversion
- 3 tasks τ_1, τ_2, τ_3 with $P_1 > P_2 > P_3$ and with τ_1 and τ_3 share resource R

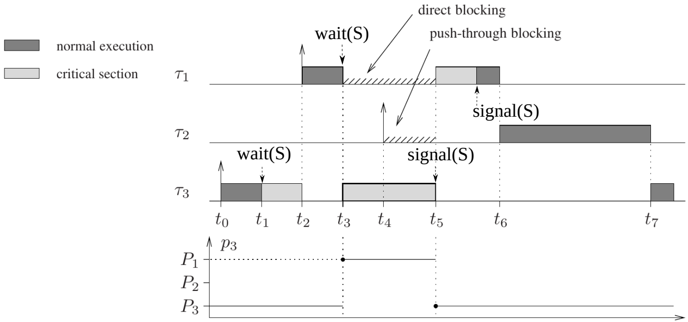


Image adapted from “Hard real-time computing systems” by Prof. G. Buttazzo

PIP: nested critical sections

- 3 tasks τ_1, τ_2, τ_3: τ_1 and τ_3 share resource R_a; τ_2 and τ_3 share resource R_b

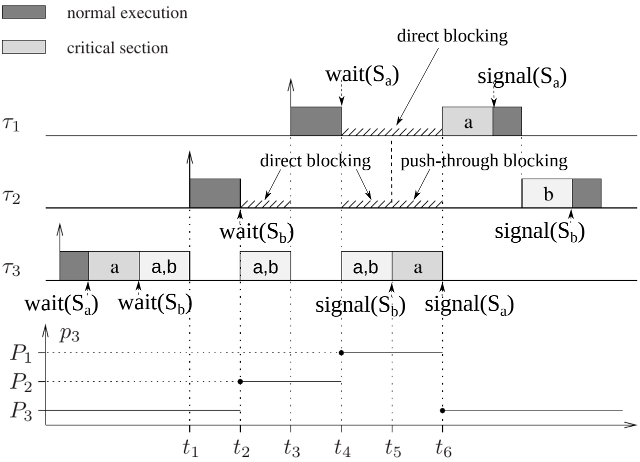


Image adapted from "Hard real-time computing systems" by Prof. G. Buttazzo

PIP: transitive priority inheritance

- 3 tasks τ_1, τ_2, τ_3: τ_1 and τ_2 share resource R_a; τ_2 and τ_3 share resource R_b
- At time t_4, task τ_3 inherits the priority of task τ_1 via task τ_2

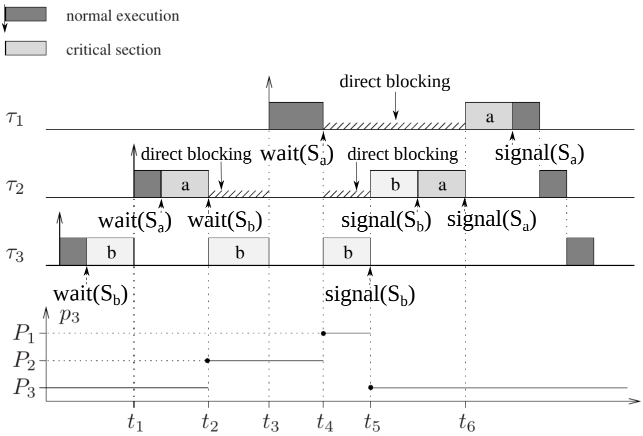


Image adapted from "Hard real-time computing systems" by Prof. G. Buttazzo

PIP: properties (1/5)

- When does blocking occur?

Lemma 1

A semaphore S_k can cause push-through blocking to a task τ_i only if S_k is accessed by a task with priority $< P_i$ and a task with priority $> P_i$

Proof.

Ab absurdo, assume that S_k is accessed by a task τ_l with priority $< P_i$ but not by a task with priority $> P_i \Rightarrow \tau_l$ cannot inherit a priority $> P_i \Rightarrow \tau_i$ will preempt τ_l $\square$

PIP: properties (2/5)

- When does transient priority inheritance occur?

Lemma 2

Transient priority inheritance can occur only in case of nested critical sections

Proof.

Transient priority inheritance occurs when a high priority task τ_h is blocked by a mid priority task τ_m that, in turn, is blocked by a low priority task $\tau_l \Rightarrow$
$\Rightarrow \tau_m$ holds a semaphore S_a (given that τ_m blocks τ_h) and
τ_l holds a different semaphore S_b (given that τ_l blocks τ_m) $\Rightarrow$
$\Rightarrow \tau_m$ attempted to lock S_b inside the critical section guarded by $S_a \Rightarrow$
$\Rightarrow$ The two critical sections are nested $\square$

PIP: properties (3/5)

- How many times can a task be blocked?

Lemma 3

If there are l_i lower priority tasks that can block a task $\tau_i \Rightarrow$
$\Rightarrow \tau_i$ can be blocked for **at most** the duration of l_i critical sections
(one for each lower prio task, regardless of the number of semaphores used by τ_i)

Proof.

τ_i can be blocked by a lower priority task τ_j only if τ_i has preempted τ_j
within a critical section $z_{j,k}$ that can block $\tau_i \Rightarrow$
$\Rightarrow \tau_j$ can be preempted by τ_i once it exits $z_{j,k} \Rightarrow$
$\Rightarrow \tau_i$ cannot be blocked by τ_j again $\Rightarrow$
$\Rightarrow \tau_i$ can be blocked at most l_i times $\square$

PIP: properties (4/5)

- How many times can a task be blocked?

Lemma 4

If there are s_i distinct semaphores that can block a task $\tau_i \Rightarrow$
$\Rightarrow \tau_i$ can be blocked for **at most** the duration of s_i critical sections
(one for each semaphore, regardless of the number of critical sections used by τ_i)

Proof.

Semaphores are binary $\Rightarrow$
$\Rightarrow$ only one of the lower prio task τ_j can be within a blocking critical section $\Rightarrow$
$\Rightarrow \tau_j$ can be preempted by τ_i once τ_j exits such critical section $\Rightarrow$
$\Rightarrow \tau_i$ cannot be blocked by τ_j again $\Rightarrow$
$\Rightarrow \tau_i$ can be blocked at most s_i times $\square$

PIP: properties (5/5)

- How many times can a task be blocked?

Theorem 1

Under the Priority Inheritance Protocol (PIP), a task τ_i can be blocked for **at most** the duration of $\alpha_i = \min\{l_i, s_i\}$ critical sections, where:

- l_i is the number of lower priority tasks that can block τ_i
- s_i is the number of semaphores that can block τ_i

Proof.

The thesis directly follows from Lemmas 3 and 4 $\square$

- Not tight bounds on blocking times of tasks are derived based on Theorem 1

PIP: summary (1/2)

- Advantages
  - Low pessimism (a task is blocked only when really needed)
  - Transparency to the programmer
  - Bounded blocking time (at most the duration of α_i critical sections)
- Disadvantages (1/2)
  - Computation of blocking times is quite complex
    (due to direct blocking, push-through blocking, transitive priority inheritance)
  - Implementation somewhat hard (requires modifying kernel data structures)
  - Prone to **chained blocking** (each task τ_i blocked α_i times in the worst case)

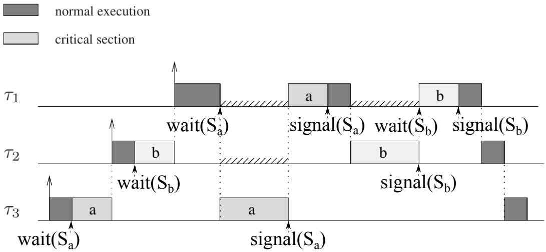


Image adapted from "Hard real-time computing systems" by Prof. G. Buttazzo

PIP: summary (2/2)

- Disadvantages (2/2)
  - Does not prevent **deadlocks** caused by wrong use of semaphores

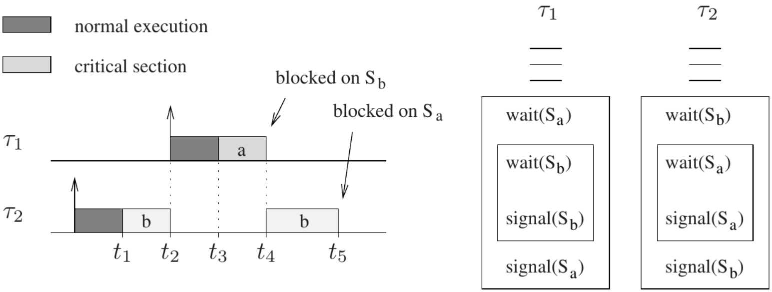


Image from "Hard real-time computing systems" by Prof. G. Buttazzo

Priority Ceiling Protocol (PCP) (1/2) (Sha et al, 1990)

- The **priority ceiling** $C(S_k)$ of a semaphore S_k is the highest priority among those of the tasks that can lock S_k, i.e., $C(S_k) := \max_{i \in \{1,\dots,n\}} \{P_i \mid \tau_i \text{ uses } S_k\}$

- **Access** rule: A task τ_i blocks at the entrance of a critical section if its priority is not higher than the maximum ceiling of the semaphores locked by other tasks, i.e., $P_i \leq \max\{C(S_k) \mid S_k \text{ locked by tasks } \neq \tau_i\}$
  - **PCP access test** for granting a lock request on a free semaphore
  - A task is not allowed to enter a critical section locked by a free semaphore if there are locked semaphores that could block it ⇒ Once a task enters its first critical section, it can never be blocked by lower prio tasks until its completion

- **Progress** rule is the same as in the PIP

- **Release** rule is the same as in the PIP

L. Sha, R. Rajkumar, J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", IEEE Trans. on Comp., 39(9), Sept. 1990

Priority Ceiling Protocol (PCP) (2/2)

- **Progress** rule: Inside a critical section associated with resource R_k a task executes with the highest priority of the tasks blocked on R_k
  - Task τ_j **inherits** the highest priority of the tasks it blocks:

$$p_j = \max\{P_j, \max_i\{P_i \mid \tau_i \text{ is blocked on } R_k\}\}$$

  - **Transitivity** property: if τ_3 blocks τ_2 and τ_2 blocks $\tau_1 \Rightarrow p_3 = P_1$

- **Release** rule: When τ_j exits the critical section associated with resource R_k:
  - Semaphore S_k is unlocked
  - The highest priority task blocked on S_k (if any) is awakened
  - If no other task is blocked by τ_j then $p_j = P_j$ otherwise τ_j inherits the highest priority of the tasks it blocks

PCP: ceiling blocking

- **Ceiling blocking**: occurs when a task is blocked given that it does not pass the PCP access test, i.e., $P_i \leq \max\{C(S_k) \mid S_k \text{ locked by tasks } \neq \tau_i\}$
  - Necessary to avoid deadlock and chained blocking
- 3 tasks τ_1, τ_2, τ_3 with $P_1 > P_2 > P_3$, and with τ_1 using resources R_A and R_B, τ_2 using resource R_C, and τ_3 using resources R_B and R_C

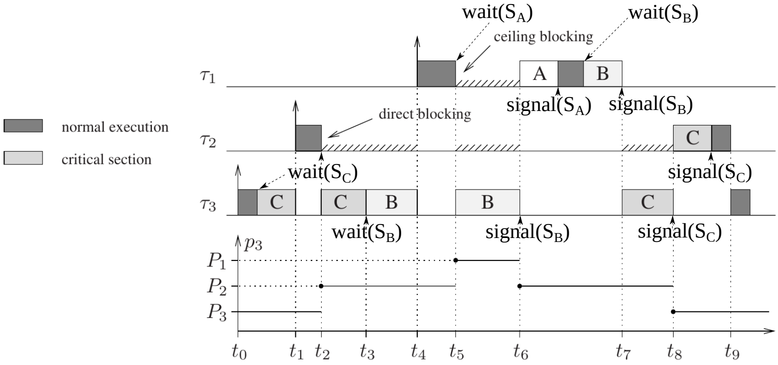


Image from "Hard real-time computing systems" by Prof. G. Buttazzo

PCP: properties (1/4)

Lemma 1

If a task τ_k is preempted within a critical section by a task τ_i that enters a critical section $z_{i,b} \Rightarrow \tau_k$ cannot inherit a priority $\geq$ the priority of τ_i until τ_i completes

Proof.

- If τ_k inherits a priority $\geq$ the priority of τ_i until τ_i completes $\Rightarrow$
  $\Rightarrow \exists$ task τ_h blocked by $\tau_k \mid P_h \geq P_i$
- If τ_i enters its critical section $\Rightarrow$
  $\Rightarrow P_i > C^\star$ where $C^\star :=$ max ceiling of semaphores locked by lower prio tasks
- Hence, $P_h \geq P_i > C^\star \Rightarrow \tau_h$ cannot be blocked by τ_k, which is a contradiction

$\square$

PCP: properties (2/4)

Lemma 2

The Priority Ceiling Protocol (PCP) prevents transitive blocking

Proof.

If a transitive blocking occurs $\Rightarrow$

$\Rightarrow \exists$ tasks τ_1, τ_2, $\tau_3 \mid P_1 > P_2 > P_3$, τ_1 is blocked by τ_2, τ_2 is blocked by $\tau_3 \Rightarrow$

$\Rightarrow \tau_3$ will inherit the priority of τ_1, which contradicts Lemma 1 $\square$

PCP: properties (3/4)

Theorem 1

The Priority Ceiling Protocol (PCP) prevents deadlocks

Proof.

If a deadlock occurs $\Rightarrow$

$\Rightarrow \exists$ tasks $\tau_1, \tau_2, \ldots, \tau_n \mid P_1 > P_2 > \cdots > P_n$, τ_1 is blocked by τ_2, τ_2 is blocked $\ldots \Rightarrow$

$\Rightarrow \tau_n$ will inherit the priority of τ_1, which contradicts Lemma 1 $\square$

PCP: properties (4/4)

Theorem 2

Under the Priority Ceiling Protocol (PCP), a task can be blocked for at most the duration of one critical section

Proof.

- Let task τ_i be blocked by τ_1 and τ_2 with $P_i > P_1 > P_2$
- Let τ_2 enter its blocking critical section first
- Let $C_2^\star$ be the max ceiling among the semaphores locked by τ_2
- If τ_1 enters a critical section $\Rightarrow P_1 > C_2^\star$
- If τ_i can be blocked by τ_2 $\Rightarrow P_i \leq C_2^\star$
- Hence $P_i \leq C_2^\star < P_1$, which contradicts the assumption that $P_i > P_1$

□

- The maximum blocking time of each task is derived based on Theorem 2

PCP: summary

- Advantages
  - Limits blocking to the duration of a critical section
  - Prevents deadlock and transitive blocking
- Disadvantages
  - Complex to implement
  - Pessimistic (it can cause unnecessary blocking)
  - Not trasparent to the programmer (ceilings specified in the source code)

Schedulability analysis of periodic tasks with shared resources

- Unbounded blocking times $\Rightarrow$ Unfeasible task set
- Bounded blocking times $\Rightarrow$ Extend schedulability tests for independent tasks
  - Guarantee one task at a time
  - Preemption by higher priority tasks and blocking by lower priority tasks
  - Blocking conditions derived in worst case scenarios that differ for each task and could never occur simultaneously $\Rightarrow$ Tests are **only sufficient**

Extended schedulability tests (1/3)

- Utilization based analysis
  - LL test for RM:
    A set of periodic tasks $\{\tau_1, \ldots, \tau_n\}$ with blocking factors and with $D_i = T_i$
    $\forall$ task τ_i is schedulable by RM if $\sum_{k|P_k>P_i} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leq i(2^{1/i} - 1)$ $\forall$ task τ_i
  - HB test for RM:
    A set of periodic tasks $\{\tau_1, \ldots, \tau_n\}$ with blocking factors and with $D_i = T_i$
    $\forall$ task τ_i is schedulable by RM if $\prod_{k|P_k>P_i} \left(\frac{C_k}{T_k} + 1\right)\left(\frac{C_i + B_i}{T_i} + 1\right) \leq 2$ $\forall$ task τ_i
  - LL test for EDF:
    A set of periodic tasks $\{\tau_1, \ldots, \tau_n\}$ with blocking factors and with $D_i = T_i$
    $\forall$ task τ_i is schedulable by EDF if $\sum_{k|P_k>P_i} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leq 1$ $\forall$ task τ_i

Extended schedulability tests (2/3)

- Response time analysis
  - A set of periodic tasks $\{\tau_1, \ldots, \tau_n\}$ with blocking factors and with $D_i \leq T_i$ $\forall$ task τ_i is schedulable by DM if $R_i = C_i + B_i + \sum_{k|P_k > P_i} \left\lceil \frac{R_i}{T_k} \right\rceil C_k \leq D_i$ $\forall$ task τ_i
  - Iterative solution to compute R_i

$$\begin{cases} R_i^{(0)} = \sum_{k|P_k>P_i} C_k + C_i + B_i \\ R_i^{(j)} = C_i + B_i + \sum_{k|P_k>P_i} \left\lceil \frac{R_i^{(j-1)}}{T_k} C_k \right\rceil \end{cases}$$

Extended schedulability tests (3/3)

- Processor demand analysis
  - A set of periodic tasks $\{\tau_1, \ldots, \tau_n\}$ with blocking factors and with $D_i \leq T_i$ $\forall$ task τ_i is schedulable by EDF if $U < 1$ and dbf$(t)+B(t) \leq t \;\forall\, t \in D$
    - dbf$(t) = \sum_i \left\lfloor \frac{t + T_i - D_i}{T_i} C_i \right\rfloor$
    - $B(t) = \max_{i,j \,|\, i \neq j} \{\beta_{ij} \mid D_i > t \wedge D_j \leq t\}$ is termed **blocking function** (maximum time for which τ_i with $D_i \leq T_i$ may be blocked by τ_j with $D_i > t$)
    - β_{ij} := maximum time for which τ_i holds a resource that is also needed by τ_j
    - $D := \{d_i \mid d_i \leq \max\{D_{\max}, \min\{H, t^\star\}\}\}$, $D_{\max} := max_i\{D_i\}$
    - $H :=$lcm$(T_1, \ldots, T_n)$, $t^\star = \sum_i (T_i - D_i) U_i / (1 - U)$

Credits & References

Credits

- Most of this material is taken from the slides of the course “Real-Time Systems” given by Prof. Giorgio Buttazzo in the A.Y. 2018/2019: `http://retis.sssup.it/~giorgio/rts-MECS.html`
- These slides are authorized for personal use only
- Any other use, redistribution, and profit sale of the slides (in any form) requires the consent of the copyright owners

References

- Giorgio Buttazzo, “Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications”, Third Edition, Springer, 2011
  - Chapters 1, 2, 4, 7, 11, 12; Paragraph 3.3