Fixed Priority Scheduling

Giuseppe Lipari
http://feanor.sssup.it/~lipari

Scuola Superiore Sant'Anna – Pisa

November 23, 2011

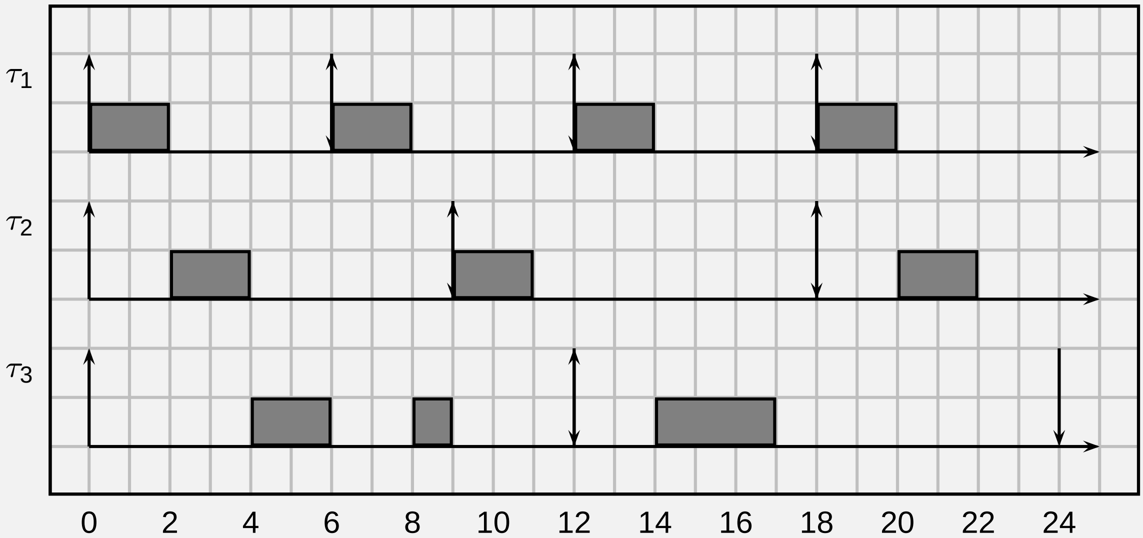
Outline

The fixed priority scheduling algorithm

- very simple scheduling algorithm;
  - every task τ_i is assigned a fixed priority p_i;
  - the active task with the highest priority is scheduled.
- Priorities are integer numbers: the higher the number, the higher the priority;
  - In the research literature, sometimes authors use the opposite convention: the lowest the number, the highest the priority.
- In the following we show some examples, considering periodic tasks, and constant execution time equal to the period.

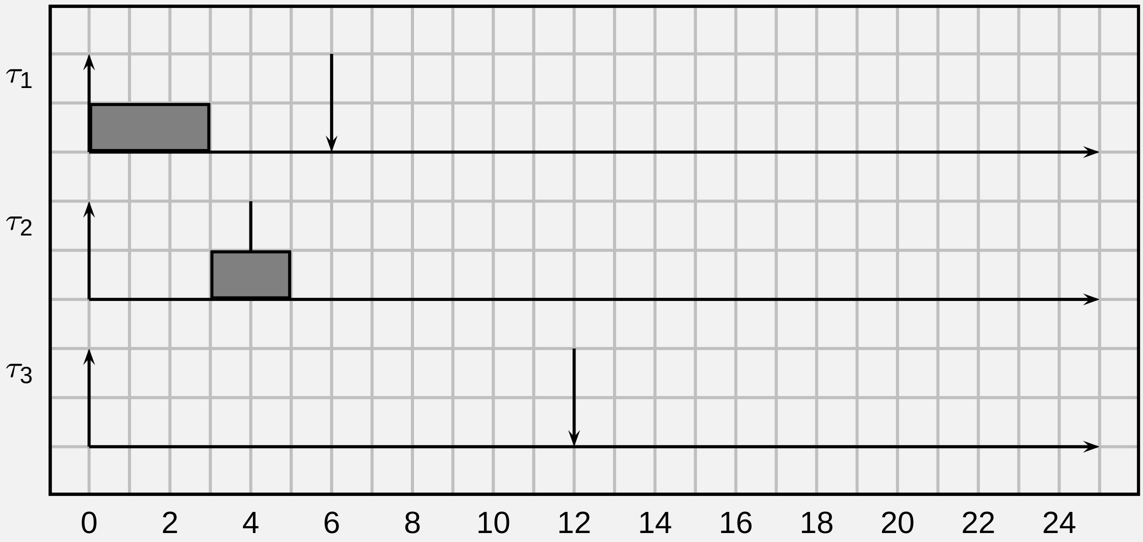
Example of schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest).



Another example (non-schedulable)

- Consider the following task set: $\tau_1 = (3, 6, 6)$, $p_1 = 3$, $\tau_2 = (2, 4, 8)$, $p_2 = 2$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$.



In this case, task τ_3 misses its deadline!

Note

- Some considerations about the schedule shown before:
  - The response time of the task with the highest priority is minimum and equal to its WCET.
  - The response time of the other tasks depends on the *interference* of the higher priority tasks;
  - The priority assignment may influence the schedulability of a task.

Priority assignment

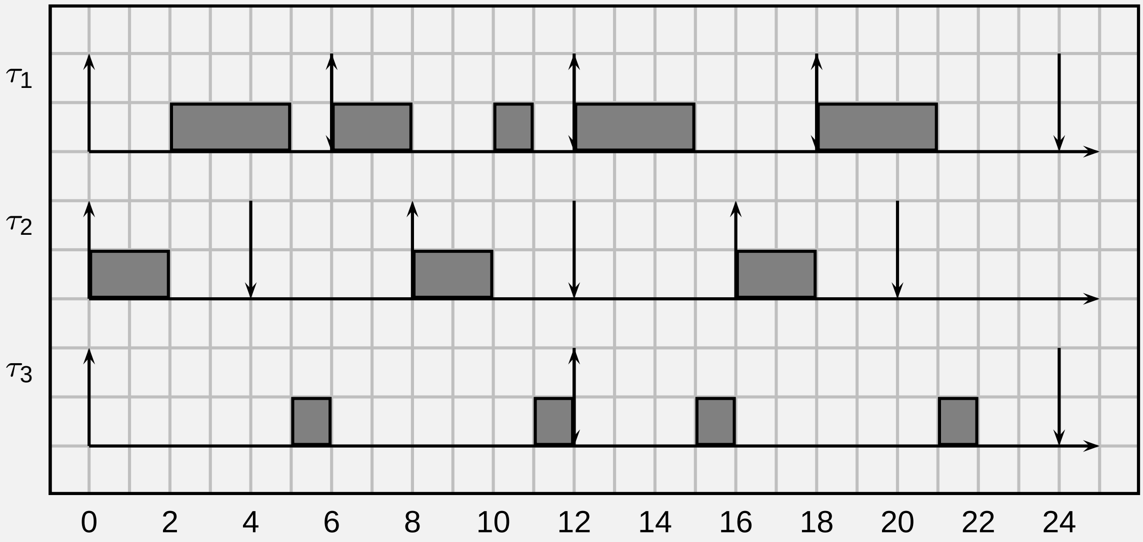
- Given a task set, how to assign priorities?
- There are two possible objectives:
  - Schedulability (i.e. find the priority assignment that makes all tasks schedulable)
  - Response time (i.e. find the priority assignment that minimize the response time of a subset of tasks).
- By now we consider the first objective only
- An *optimal* priority assignment *Opt* is such that:
  - If the task set is schedulable with another priority assignment, then it is schedulable with priority assignment *Opt*.
  - If the task set is not schedulable with *Opt*, then it is not schedulable by any other assignment.

Optimal priority assignment

- Given a periodic task set with all tasks having deadline equal to the period ($\forall i, \quad D_i = T_i$), and with all offsets equal to 0 ($\forall i, \quad \phi_i = 0$):
  - The best assignment is the *Rate Monotonic* assignment
  - Tasks with shorter period have higher priority
- Given a periodic task set with deadline different from periods, and with all offsets equal to 0 ($\forall i, \quad \phi_i = 0$):
  - The best assignement is the *Deadline Monotonic* assignment
  - Tasks with shorter relative deadline have higher priority
- For sporadic tasks, the same rules are valid as for periodic tasks with offsets equal to 0.

Example revised

- Consider the example shown before with deadline monotonic: $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 10, 12)$, $p_3 = 1$.

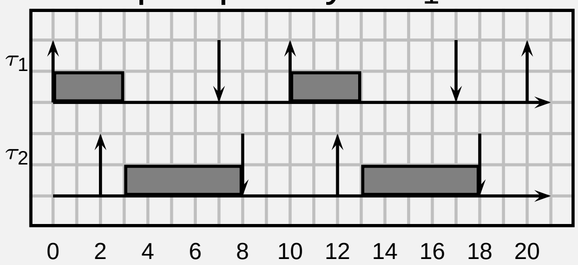


Presence of offsets

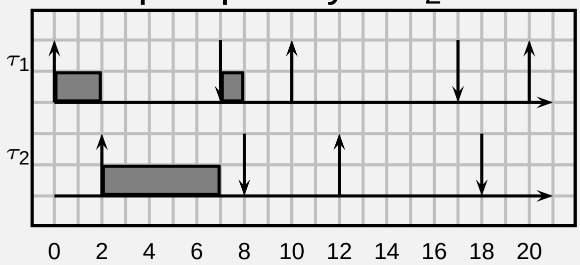
- If instead we consider periodic tasks with offsets, then *there is no optimal priority assignment*
  - In other words,
    - if a task set $\mathcal{T}_1$ is schedulable by priority O_1 and not schedulable by priority assignment O_2,
    - it may exist another task set $\mathcal{T}_2$ that is schedulable by O_2 and not schedulable by O_1.
  - For example, $\mathcal{T}_2$ may be obtained from $\mathcal{T}_1$ simply changing the offsets!

Example of non-optimality with offsets

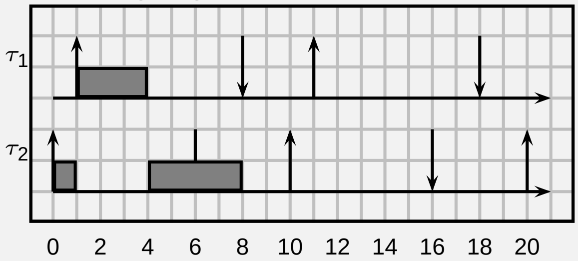
Example: priority to τ_1:



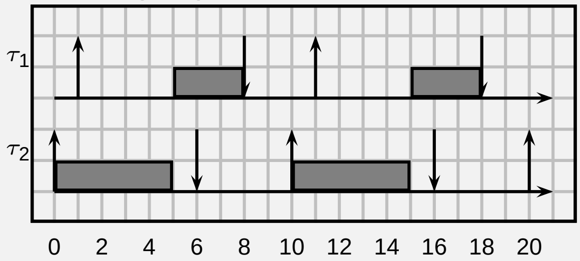
Example: priority to τ_2:



Changing the offset:



Changing the offset:



Analysis

- Given a task set, how can we guarantee if it is schedulable of not?
- The first possibility is to *simulate* the system to check that no deadline is missed;
- The execution time of every job is set equal to the WCET of the corresponding task;
  - In case of periodic task with no offsets, it is sufficient to simulate the schedule until the *hyperperiod* ($H = lcm_i(T_i)$).
  - In case of offsets, it is sufficient to simulate until $2H + \phi_{\max}$ (Leung and Merril).
  - If tasks periods are prime numbers the hyperperiod can be very large!

Example

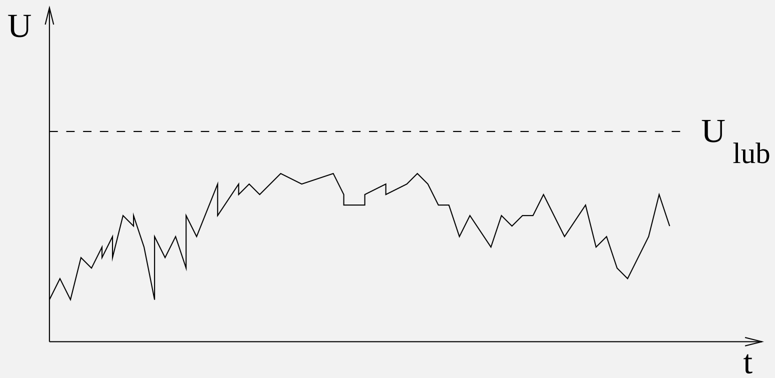
- Exercise: Compare the hyperperiods of this two task sets:
  1. $T_1 = 8$, $T_2 = 12$, $T_3 = 24$;
  2. $T_1 = 7$, $T_2 = 12$, $T_3 = 25$.
- In case of sporadic tasks, we can assume them to arrive at the highest possible rate, so we fall back to the case of periodic tasks with no offsets!
- In case 1, $H = 24$;
- In case 2, $H = 2100$!

Utilization analysis

- In many cases it is useful to have a very simple test to see if the task set is schedulable.
- A sufficient test is based on the *Utilization bound*:

Definition

The *utilization least upper bound* for scheduling algorithm $\mathcal{A}$ is the smallest possible utilization U_{lub} such that, for any task set $\mathcal{T}$, if the task set's utilization U is not greater than U_{lub} ($U \leq U_{lub}$), then the task set is schedulable by algorithm $\mathcal{A}$.



Utilization bound for RM

Theorem (Liu and Layland, 1973)

Consider n periodic (or sporadic) tasks with relative deadline equal to periods, whose priorities are assigned in Rate Monotonic order. Then,

$$U_{lub} = n(2^{1/n} - 1)$$

- U_{lub} is a decreasing function of n;
- For large n: $U_{lub} \approx 0.69$

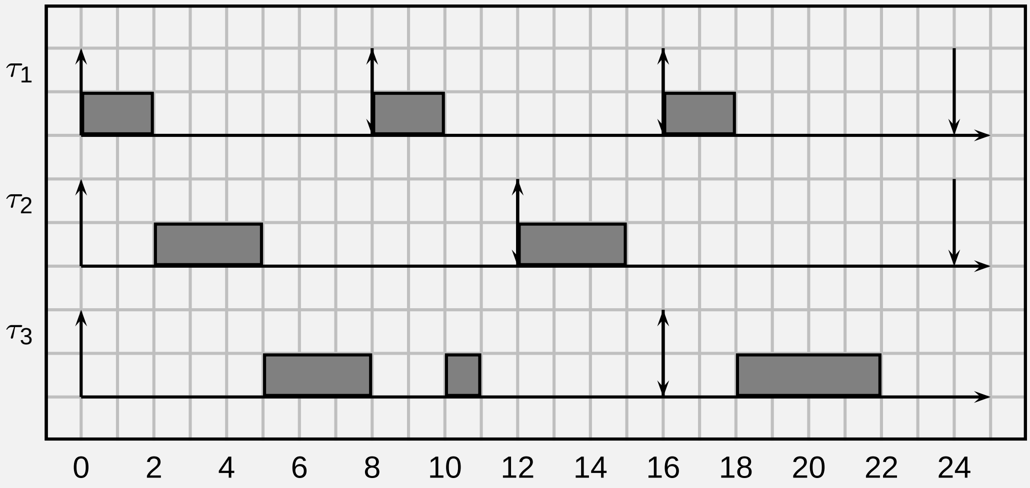
n	U_{lub}	n	U_{lub}
2	0.828	7	0.728
3	0.779	8	0.724
4	0.756	9	0.720
5	0.743	10	0.717
6	0.734	11	...

Schedulability test

- Therefore the schedulability test consist in:
  - Compute $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$;
  - if $U \leq U_{lub}$, the task set is schedulable;
  - if $U > 1$ the task set is not schedulable;
  - if $U_{lub} < U \leq 1$, the task set may or may not be schedulable;

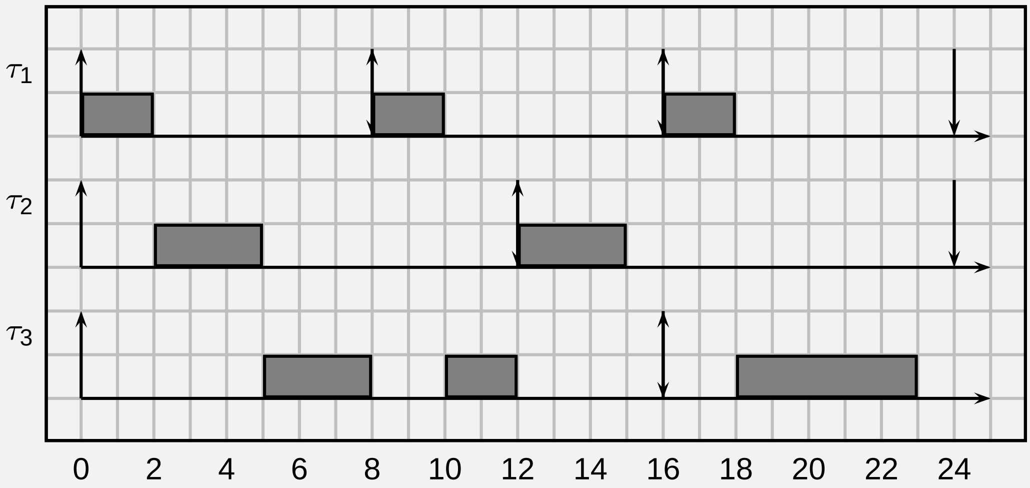
Example

- Example in which we show that for 3 tasks, if $U < U_{lub}$, the system is schedulable.
  $\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$
  $U = 0.75 < U_{lub} = 0.77$



Example 2

- By increasing the computation time of task τ_3, the system may still be schedulable . . .
  $\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (5, 16);$
  $U = 0.81 > U_{lub} = 0.77$



Utilization bound for DM

- If relative deadlines are less than or equal to periods, instead of considering $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$, we can consider:

$$U' = \sum_{i=1}^{n} \frac{C_i}{D_i}$$

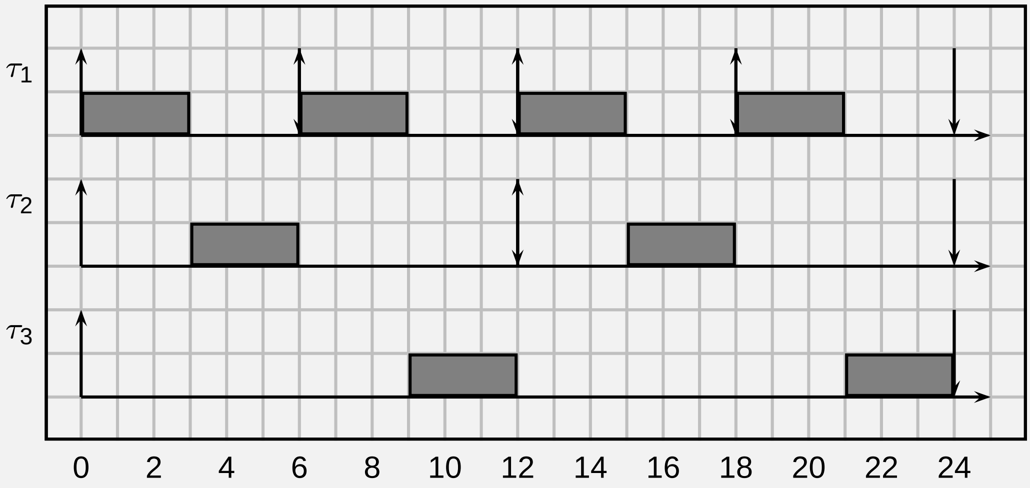
- Then the test is the same as the one for RM (or DM), except that we must use U' instead of U.

Pessimism

- The bound is very pessimistic: most of the times, a task set with $U > U_{lub}$ is schedulable by RM.
- A particular case is when tasks have periods that are *harmonic*:
  - A task set is *harmonic* if, for every two tasks τ_i, tau_j, either P_i is multiple of P_j or P_j is multiple of P_i.
- For a harmonic task set, the utilization bound is $U_{lub} = 1$.
- In other words, Rate Monotonic is an *optimal* algoritm for harmonic task sets.

Example of harmonic task set

- $\tau_1 = (3, 6)$, $\tau_2 = (3, 12)$, $\tau_3 = (6, 24)$;
  $U = 1$;



Response time analysis

- A necessary and sufficient test is obtained by computing the *worst-case response time* (WCRT) for every task.
- For every task τ_i:
  - Compute the WCRT R_i for task τ_i;
  - If $R_i \leq D_i$, then the task is schedulable;
  - else, the task is not schedulable; we can also show the situation that make task τ_i miss its deadline!
- To compute the WCRT, we do not need to do any assumption on the priority assignment.
- The algorithm described in the next slides is valid for an arbitrary priority assignment.
- The algorithm assumes periodic tasks with no offsets, or sporadic tasks.

Response time analysis - II

- The *critical instant* for a set of periodic real-time tasks, with offset equal to 0, or for sporadic tasks, is when all jobs start at the same time.

Theorem (Liu and Layland, 1973)

The WCRT for a task corresponds to the response time of the job activated at the critical instant.

- To compute the WCRT of task τ_i:
  - We have to consider its computation time
  - and the computation time of the higher priority tasks (*interference*);
  - higher priority tasks can *preempt* task τ_i, and increment its response time.

Response time analysis - III

- Suppose tasks are ordered by decreasing priority. Therefore, $i < j \rightarrow prio_i > prio_j$.
- Given a task τ_i, let $R_i^{(k)}$ be the WCRT computed at step k.

$$R_i^{(0)} = C_i + \sum_{j=1}^{i-1} C_j$$

$$R_i^{(k)} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j$$

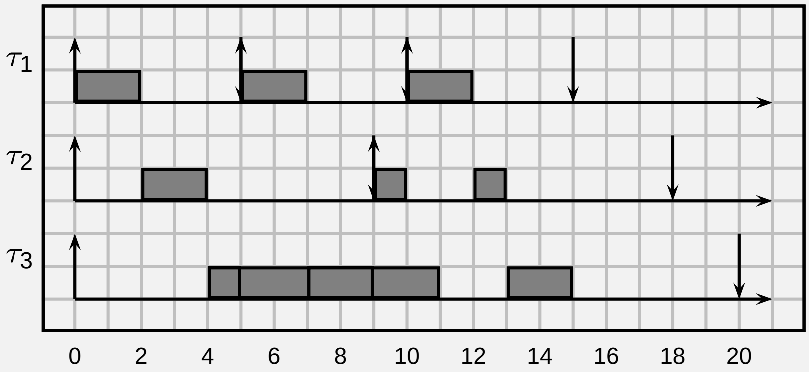
- The iteration stops when:
  - $R_i^{(k)} = R_i^{(k+1)}$ *or*
  - $R_i^{(k)} > D_i$ (non schedulable);

Example

- Consider the following task set: $\tau_1 = (2,5)$, $\tau_2 = (2,9)$, $\tau_3 = (5,20)$; $U = 0.872$.

$$R_i^{(k)} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j$$

- $R_3^{(0)} = C_3 + 1 \cdot C_1 + 1 \cdot C_2 = 9$
- $R_3^{(1)} = C_3 + 2 \cdot C_1 + 1 \cdot C_2 = 11$
- $R_3^{(2)} = C_3 + 3 \cdot C_1 + 2 \cdot C_2 = 15$
- $R_3^{(3)} = C_3 + 3 \cdot C_1 + 2 \cdot C_2 = 15 = R_3^{(2)}$

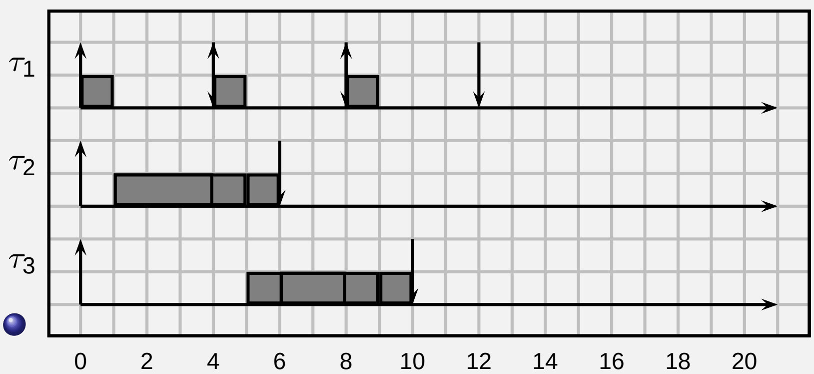


Another example with DM

- The method is valid for different priority assignments and deadlines different from periods
- $\tau_1 = (1,4,4), p_1 = 3$, $\tau_2 = (4,6,15), p_2 = 2$, $\tau_3 = (3,10,10), p_3 = 1$; $U = 0.72$

$$R_i^{(k)} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j$$

- $R_3^{(0)} = C_3 + 1 \cdot C_1 + 1 \cdot C_2 = 8$
- $R_3^{(1)} = C_3 + 2 \cdot C_1 + 1 \cdot C_2 = 9$
- $R_3^{(2)} = C_3 + 3 \cdot C_1 + 2 \cdot C_2 = 10$
- $R_3^{(3)} = C_3 + 3 \cdot C_1 + 2 \cdot C_2 = 10 = R_3^{(2)}$



Considerations

- The response time analysis is an efficient algorithm
  - In the worst case, the number of steps N for the algorithm to converge is exponential
    - It depends on the total number of jobs of higher priority tasks that may be contained in the interval $[0, D_i]$:

$$N \propto \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil$$

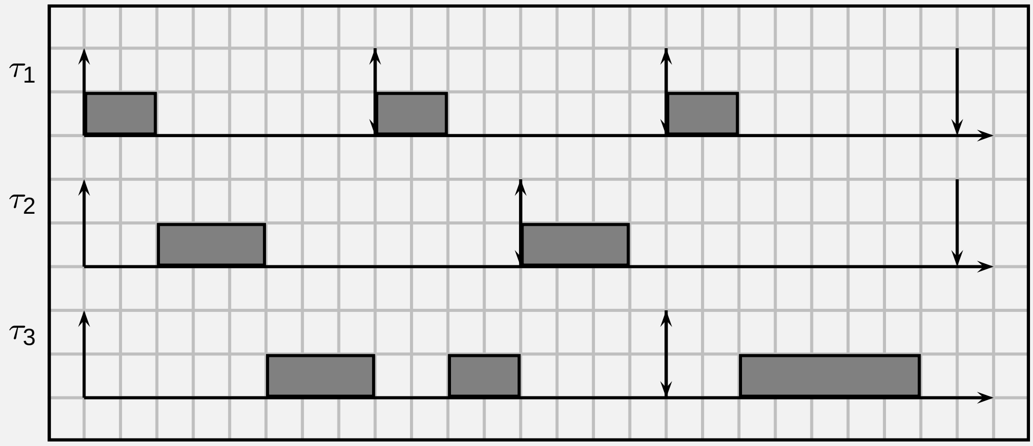
    - If s is the minimum granularity of the time, then in the worst case $N = \frac{D_i}{s}$;
  - However, such worst case is very rare: usually, the number of steps is low.

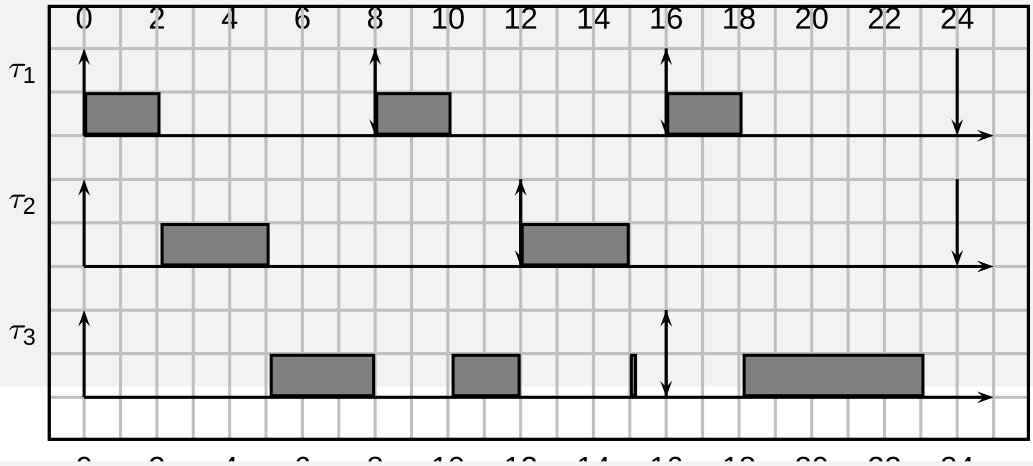
Considerations on WCET

- The response time analysis is a necessary and sufficient test for fixed priority.
- However, the result is very sensitive to the value of the WCET.
  - If we are wrong in estimating the WCET (and for example we put a value that is too low), the actual system may be not schedulable.
- The value of the response time is not helpful: even if the response time is well below the deadline, a small increase in the WCET of a higher priority task makes the response time *jump*;
- We may see the problem as a sensitivity analysis problem: we have a function $R_i = f_i(C_1, T_1, C_2, T_2, \ldots, C_{i-1}, T_{i-1}, C_i)$ that is non-continuous.

Example of discontinuity

- Let's consider again the example done *before*; we increment the computation time of τ_1 of 0.1.





Singularities

- The response time of a task τ_i is the first time at which all tasks $\tau_1, \ldots, \tau_i$ have completed;
- At this point,
  - either a lower priority task τ_j ($p_j < p_i$) is executed
  - or the system becomes idle
  - or it coincides with the arrival time of a higher priority task.
- In the last case, such an instant is also called *i*-level singularity point.
- In the previous example, time 12 is a 3-level singularity point, because:
  1. task τ_3 has just finished;
  2. and task τ_2 ha just been activated;
- A singularity is a dangerous point!

Sensitivity on WCETs

- A rule of thumb is to increase the WCET by a certain percentage before doing the analysis. If the task set is still feasible, be are more confident about the schedulability of the original system.
- There are analytical methods for computing the amount of variation that it is possible to allow to a task's WCET without compromising the schedulability

A different analysis approach

- Definition of workload for task τ_i:

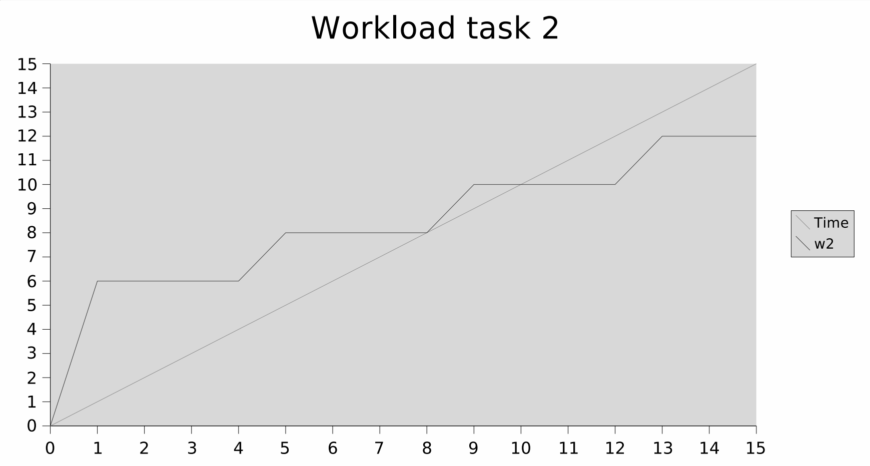
$$W_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

- The workload is the amount of "work" that the set of tasks $\{\tau_1, \ldots, \tau_i\}$ requests in $[0, t]$
- Example: $\tau_1 = (2, 4)$, $\tau_2 = (4, 15)$:

$$W_2(10) = \left\lceil \frac{10}{4} \right\rceil 2 + \left\lceil \frac{10}{15} \right\rceil 4 = 6 + 4 = 10$$

Workload function

- The workload function for the previous example
  - $\tau_1 = (2, 4)$, $\tau_2 = (4, 15)$:



Main theorem

Theorem (Lehokzcy 1987)

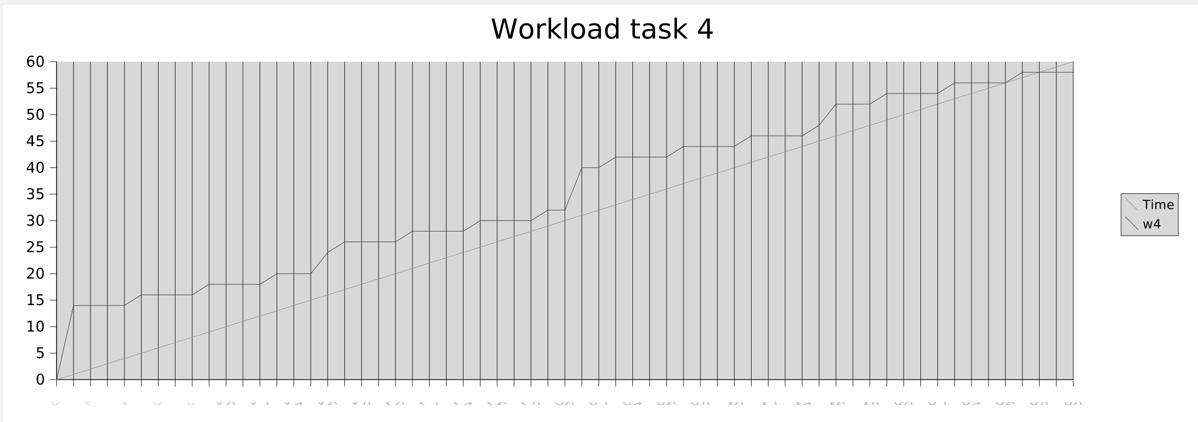
Let $\mathcal{P}_i = \{\forall j < i, \forall k, kT_j \leq D_i | kT_j\} \cup \{D_i\}$. Then, task τ_i is schedulable if and only if

$$\exists t \in \mathcal{P}_i, \quad W_i(t) \leq t$$

- Set $\mathcal{P}_i$ is the set of time instants that are multiple of some period of some task τ_j with higher priority than τ_i, plus the deadline of task τ_i (they are potential singularity points)
- In other words, the theorem says that, if the workload is less than t for any of the points in $\mathcal{P}_i$, then τ_i is schedulable
- Later, Bini simplified the computation of the points in set $\mathcal{P}_i$

Example with 4 tasks

- $\tau_1 = (2, 4)$, $\tau_2 = (4, 15)$, $\tau_3 = (4, 30)$, $\tau_4 = (4, 60)$



- Task τ_4 is schedulable, because $W_4(56) = 56$ and $W_4(60) = 58 < 60$
- (see schedule on fp_schedule_1.0_ex4.ods)

Sensitivity analysis

- Proposed by Bini and Buttazzo, 2005
- Let us rewrite the equations for the workload:

$$\exists t \in \mathcal{P}_i \quad \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t$$

- If we consider the C_j as variables, we have a set of linear inequalities in *OR* mode
- each inequality defines a plane in the space R^i of variables $C_1, \ldots, Ci$
- the result is a concave hyper-solid in that space

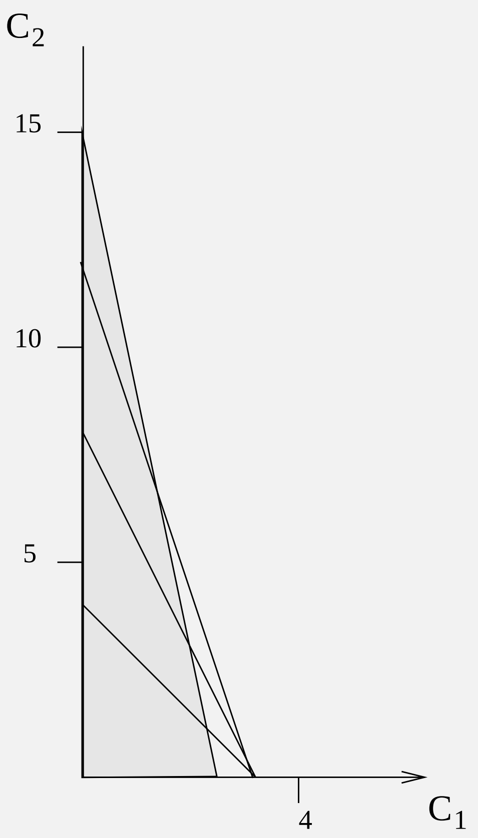
Example with two tasks

- $\tau_1 = (x, 4)$, $\tau_2 = (y, 15)$
- $\mathcal{P} = \{4, 8, 12, 15\}$

$$\left\|\begin{array}{ll} C_1 + C_2 & \leq 4 \\ 2C_1 + C_2 & \leq 8 \\ 3C_1 + C_2 & \leq 12 \\ 4C_1 + C_2 & \leq 15 \end{array}\right.$$

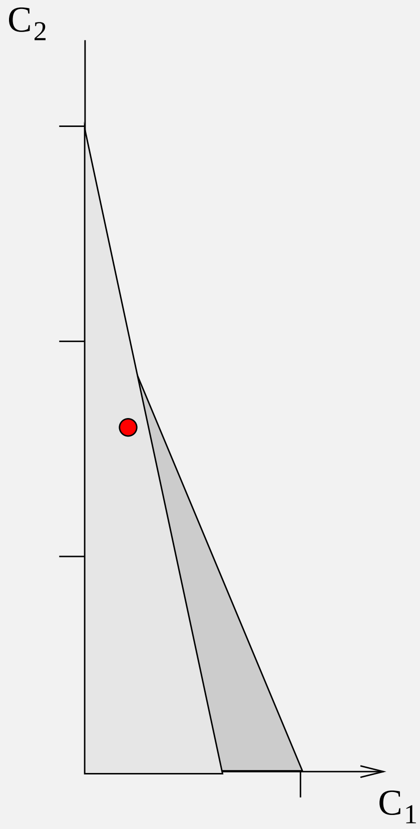
Graphical representation

- In the R^2 space:
- Notice that there are 4 overlapping regions, however only two concur to define the admissible region
- Also, notice that the final region is the *union* of all the regions



Example, cont.

- Simplifying non-useful constraints
- The red dot represent a (possible) pair of values for (C_1, C_2).
- The red dot must stay always inside the subspace



Sensitivity

- Distance from a constraint represents
  - how much we can increase (C_1, C_2) without exiting from the space
  - or how much we must decrease C_1 or C_2 to enter in the space
  - In the example before: starting from $C_1 = 1$ and $C_2 = 8$ we can increase C_1 of the following:

$$3(1 + \Delta) + 8 \leq 12$$

$$\Delta \leq \frac{4}{3} - 1 = \frac{1}{3}$$

- **Exercise:** verify schedulability of τ_2 with $C_1 = 1 + \frac{1}{3}$ and $C_2 = 8$ by computing its response time

More than 2 tasks

- In case of more than two tasks, schedulability must be checked on all tasks:
  - For a system to be schedulable, all tasks must be schedulable
- This means that for each task we must apply the procedure described above, obtaining a system of inequalities in OR.
- Then all systems must be valid, i.e. they must all be put in AND
  - there must be at least one valid equation for each system
- See the exercise below

Summary of schedulability tests for FP

- Utilization bound test:
  - depends on the number of tasks;
  - for large n, $U_{lub} = 0.69$;
  - only sufficient;
  - $\mathcal{O}(n)$ complexity;
- Response time analysis:
  - necessary and sufficient test for periodic tasks with arbitrary deadlines and with no offset
  - complexity: high (*pseudo-polynomial*);
- Hyperplane analysis
  - necessary and sufficient test for periodic tasks with arbitrary deadlines and with no offset
  - complexity: high (*pseudo-polynomial*);
  - allows to perform sensitivity analysis

Response time analysis - extensions

- Consider offsets
- Arbitrary patterns of arrivals. Burst, quasi–periodic, etc.

Esercizio

- Dato il seguente task set:

Task	C_i	D_i	T_i
τ_1	1	4	4
τ_2	2	9	9
τ_3	3	6	12
τ_4	3	20	20

- Calcolare il tempo di risposta dei vari task nell'ipotesi che le priorità siano assegnate con RM o con DM.
- **Risposta**: Nel caso di RM,

$$R(\tau_1) = 1 \quad R(\tau_2) = 3 \quad R(\tau_3) = 7 \quad R(\tau_4) = 18$$

- Nel caso di DM,

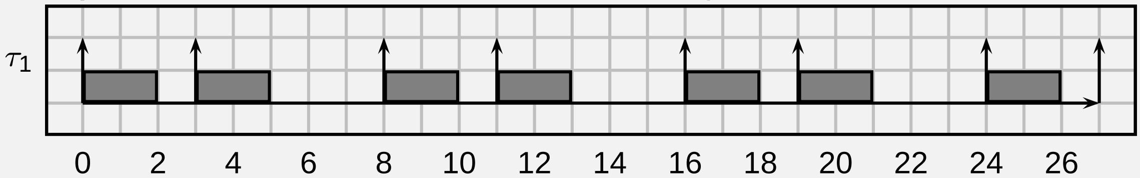
$$R(\tau_1) = 1 \quad R(\tau_2) = 7 \quad R(\tau_3) = 4 \quad R(\tau_4) = 18$$

Esercizio

- Consideriamo il seguente task τ_1 *non periodico*:
  - Se j è pari, allora $a_{1,j} = 8 \cdot \frac{j}{2}$;
  - Se j è dispari, allora $a_{1,j} = 3 + 8 \cdot \left\lfloor \frac{j}{2} \right\rfloor$;
  - In ogni caso, $c_{1,j} = 2$;
  - La priorità del task τ_1 è $p_1 = 3$.
- Nel sistema, consideriamo anche i task periodici $\tau_2 = (2, 12, 12)$ e $\tau_3 = (3, 16, 16)$, con priorità $p_2 = 2$ e $p_3 = 1$. Calcolare il tempo di risposta dei task τ_2 e τ_3.

Soluzione - I

- Il pattern di arrivo del task τ_1 è il seguente:



- Il task τ_1 è ad alta priorità, quindi il suo tempo di risposta è pari a 2.
- In che modo questo task interferisce con gli altri due task a bassa priorità

Soluzione - II

- Bisogna estendere la formula del calcolo del tempo di risposta. La generalizzazione è la seguente:

$$R_i^{(k)} = C_i + \sum_{j=1}^{i-1} Nist_j(R_i^{(k-1)})C_j$$

dove $Nist_j(t)$ rappresenta il numero di istanze del task τ_j che “arrivano” nell’intervallo $[0, t)$.

- Se il task τ_j è periodico, allora $Nist_j(t) = \left\lceil \frac{t}{T_j} \right\rceil$.
- Nel caso invece del task τ_1 (che non èperiodico):

$$Nist_1(t) = \left\lceil \frac{t}{8} \right\rceil + \left\lceil \frac{\max(0, t-3)}{8} \right\rceil$$

- Il primo termine tiene conto delle istanze con j pari, mentre il secondo termine tiene conto delle istanze con j dispari.

Soluzione - III

- Applicando la formula per calcolare il tempo di risposta del task τ_2:

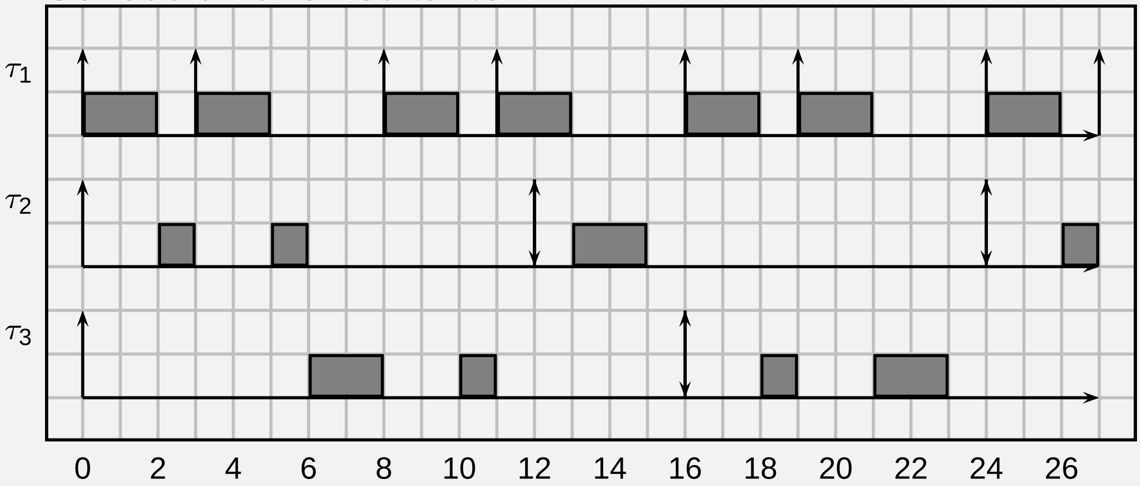
$$R_2^{(0)} = 2 + 2 = 4 \qquad R_2^{(1)} = 2 + 2 \cdot 2 = 6$$
$$R_2^{(2)} = 2 + 2 \cdot 2 = 6$$

- Per il task τ_3:

$$R_3^{(0)} = 3 + 2 + 2 = 7 \qquad R_3^{(1)} = 3 + 2 \cdot 2 + 1 \cdot 2 = 9$$
$$R_3^{(2)} = 3 + 3 \cdot 2 + 1 \cdot 2 = 11 \qquad R_3^{(3)} = 3 + 3 \cdot 2 + 1 \cdot 2 = 11$$

Soluzione - IV (schedulazione)

- Schedulazione risultante:



Esercizio sulla sensitivity

- Dato il seguente insieme di task: $\tau_1 = (2, 5)$, $\tau_2 = (3, 12)$
- Vedere se il sistema è schedulabile con l'analisi *Hyperplanes*
- Calcolare di quando può aumentare (o di quanto si può diminuire) il tempo di calcolo di τ_2 per farlo rimanere (diventare) schedulabile
- Calcolare di quanto si può diminuire la *potenza del processore* mantenendo il sistema schedulabile

Soluzione

- Le equazioni da considerare sono:

$$\left\| \begin{array}{ll} C_1 + C_2 & \leq 5 \\ 2C_1 + C_2 & \leq 10 \\ 3C_1 + C_2 & \leq 12 \end{array} \right.$$

- Tutte verificate per $C_1 = 2$ e $C_2 = 3$
- Fissando C_1, si ha:

$$\left\| \begin{array}{ll} C_2 & \leq 3 \\ C_2 & \leq 6 \\ C_2 & \leq 6 \end{array} \right.$$

- Ricordandoci che sono in *OR*, la soluzione è $C_2 \leq 6$, quindi possiamo aumentare C_2 di 3 mantenendo il sistema schedulabile

Soluzione 2

- Se il processore ha velocità variabile, le equazioni possono essere riscritte come:

$$\left\| \begin{array}{ll} \alpha C_1 + \alpha C_2 & \leq 5 \\ 2\alpha C_1 + \alpha C_2 & \leq 10 \\ 3\alpha C_1 + \alpha C_2 & \leq 12 \end{array} \right.$$

- E nel punto considerato:

$$\left\| \begin{array}{ll} \alpha & \leq 1 \\ 7\alpha & \leq 10 \\ 9\alpha & \leq 12 \end{array} \right.$$

- Quindi, $\alpha = 1.428571$, e possiamo rallentare il processore (cioé incrementare i tempi di calcolo) del 43% circa.

Esercizio con 4 tasks

Considerare il seguente insieme di task schedulati con DM.

- Controllare la schedulabilità.
- Dire di quanto è possibile aumentare/diminuire il tempo di calcolo del task τ_3 per mantenere il sistema schedulabile (o renderlo schedulabile se non lo è già).

Task	C	T	D
τ_1	1	5	5
τ_2	2	8	8
τ_3	3	15	10
τ_4	3	20	16

Soluzione

- Per controllare la schedulabilità possiamo usare il metodo del response time oppure direttamente il metodo degli hyperplanes. Per semplificare i calcoli usiamo questo secondo approccio
- Disequazioni per il task 2

$$\left\| \begin{array}{ll} C_1 + C_2 & \leq 5 \\ 2C_1 + C_2 & \leq 8 \end{array} \right.$$

- Entrambe sono pienamente rispettate. Passiamo al task 3

$$\left\| \begin{array}{ll} C_1 + C_2 + C_3 & \leq 5 \\ 2C_1 + C_2 + C_3 & \leq 8 \\ 2C_1 + 2C_2 + C_3 & \leq 10 \end{array} \right.$$

- Sostituendo si ha che la seconda e la terza sono rispettate con i dati iniziali

Soluzione – cont.

- Controlliamo ora il quarto task:

$$\left\| \begin{array}{ll} C_1 + C_2 + C_3 + C_4 & \leq 5 \\ 2C_1 + C_2 + C_3 + C_4 & \leq 8 \\ 2C_1 + 2C_2 + C_3 + C_4 & \leq 10 \\ 3C_1 + 2C_2 + C_3 + C_4 & \leq 15 \\ 4C_1 + 2C_2 + 2C_3 + C_4 & \leq 16 \end{array} \right.$$

- Notiamo che la terza equazione è rispettata
- Poiché c'è almeno una disequazione rispettata per ogni sistema, allora il task set è schedulabile

Soluzione – cont.

- Vediamo ora la sensitivity rispetto al task 3. Dobbiamo considerare il secondo e il terzo sistema (il primo non dipende dal terzo task)

$$\left\| \begin{array}{ll} C_3 & \leq 2 \\ C_3 & \leq 4 \\ C_3 & \leq 4 \end{array} \right. \qquad \left\| \begin{array}{ll} C_3 & \leq -1 \\ C_3 & \leq 1 \\ C_3 & \leq 1 \\ C_3 & \leq 5 \\ C_3 & \leq 5/2 \end{array} \right.$$

- Prendiamo il max di ogni sistema, e poi facciamo il minimo. Ne segue che $C_3 \leq 4$.