Chapter 7

Periodic Tasks Scheduling

Periodic tasks

We consider a simplified programming environment satisfying the following hypotheses:

- The number of tasks to be executed is fixed.
- Each task is characterized by a distinct and constant priority.
- The execution requests for each task occur periodically, i.e., with a constant delay between two successive requests.

  In particular, the timing of execution requests for a task cannot depend on operations performed by other tasks.
- The execution time of each task is constant.

- The following real-time constraint must be satisfied:

  Each execution of a task must finish before or at the same time as the next request for executing this task.

- Context switches are instantaneous and preemptive.

Critical instants and critical zones

In addition to its priority, each task τ_i is characterized by

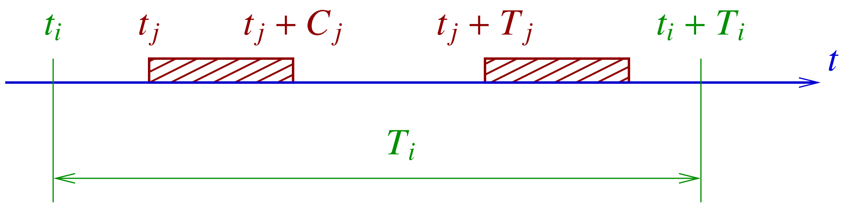
- its period T_i, and
- its execution time (for each period) C_i.

Definitions:

- The response time of an execution request for τ_i is the delay between this request and the end of the corresponding execution of this task.
- A critical instant for the task τ_i is an occurrence of an execution request for τ_i that leads to the largest possible response time for this task.
- A critical zone for τ_i is an interval of duration T_i that starts at a critical instant (for τ_i).

Theorem 1: A critical instant for τ_i occurs when an execution request for this task coincides with requests for executing all the tasks that have a higher priority than τ_i.

Proof: Assume that an execution request for τ_i occurs at $t = t_i$, and that an execution request for a higher-priority task τ_j is received at $t = t_j$.



Advancing the request for τ_j from t_j to t_i can never decrease the response time of τ_i.

(Indeed, for each instruction I of τ_i, advancing τ_j by one instruction either leaves unchanged the execution time of I, or postpones it.)

The same reasoning can be applied to all the tasks that have a higher priority than τ_i.

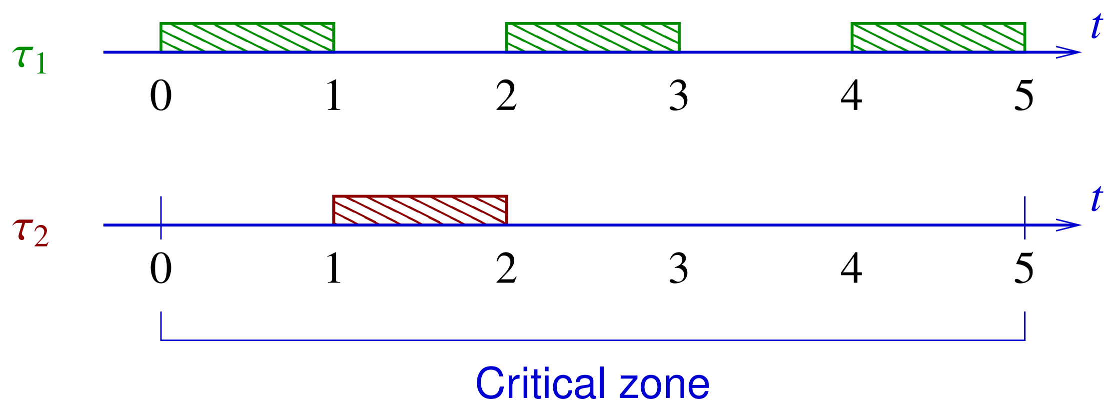
Schedulable tasks

Definition: A set of tasks is schedulable (with respect to a given assignment of priorities) if the response time of each task τ_i is always less than or equal to its period T_i.

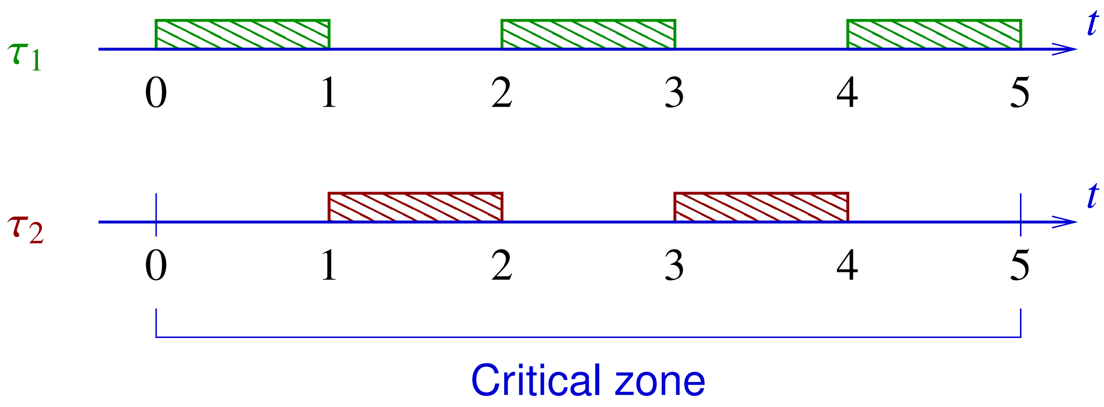
Thanks to Theorem 1, checking whether a given set of tasks is schedulable reduces to simulating the scheduling strategy in the particular case of simultaneous execution requests for all tasks at $t = 0$.

Examples: Consider two tasks τ_1 and τ_2, with $T_1 = 2$, $T_2 = 5$, $C_1 = 1$ and $C_2 = 1$.

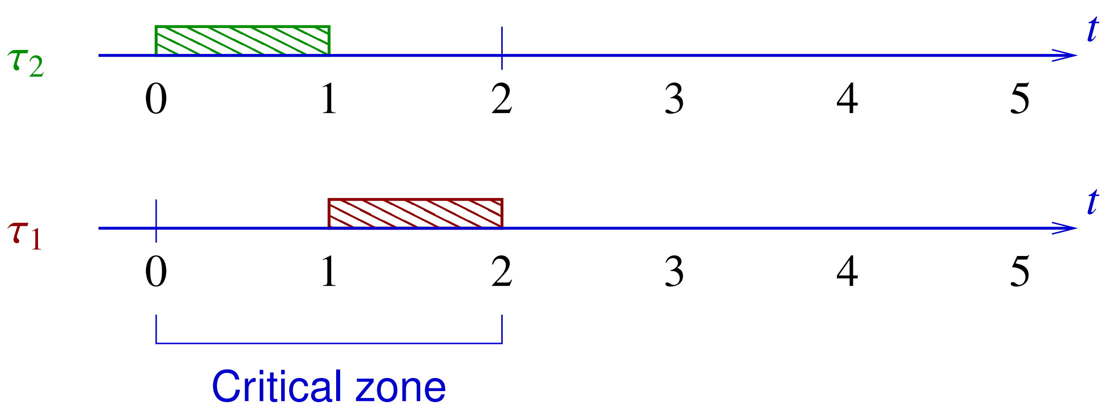
- If τ_1 has a higher priority than τ_2.



The tasks are schedulable, and remain schedulable even if the execution time of τ_2 is increased by one time unit ($C_2 = 2$):



- If τ_2 has a higher priority than τ_1.



The tasks are schedulable.

Note: In this case, the execution time of τ_1 and τ_2 cannot be increased anymore.

Rate-Monotonic Scheduling

In the previous example, the best strategy was to assign the highest priority to the task that has the smallest period.

Definition: Given a set of tasks $\tau_1, \tau_2, \ldots, \tau_n$ with respective periods $T_1, T_2, \ldots, T_n$, the Rate-Monotonic Scheduling (RMS) strategy consists in assigning distinct priorities $P_1, P_2, \ldots P_n$ to the tasks, such that for all i, j:

$$T_i < T_j \;\Rightarrow\; P_i > P_j.$$

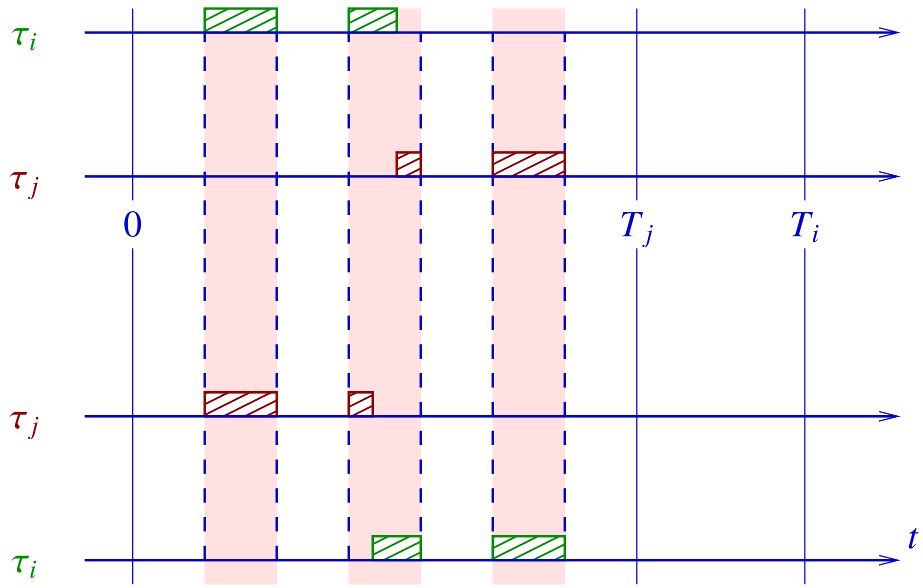
The following result establishes that the RMS strategy is optimal:

Theorem 2: If a set of tasks is schedulable with respect to some priorities assignment, then it is schedulable as well with respect to priorities defined by the RMS strategy.

Proof: Consider a set of tasks $\tau_1, \tau_2, \ldots, \tau_n$ for which there exists a priorities assignment $P_1, P_2, \ldots P_n$ that makes them schedulable.

Let τ_i and τ_j two tasks with adjacent priorities P_i and P_j, such that $P_i > P_j$.

If $T_i > T_j$, then the priorities of τ_i and τ_j can be swapped:



The resulting set of tasks remains schedulable.

By performing repeatedly this operation, one eventually obtains a priorities assignment corresponding to the RMS strategy.

The processor load factor

Consider a set of tasks $\tau_1, \tau_2, \ldots, \tau_n$ with respective periods and execution times $T_1, T_2, \ldots, T_n$ and $C_1, C_2, \ldots, C_n$.

The processor load factor U corresponding to this set of tasks represents the relative amount of CPU time needed for executing them:

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

Definition: A set of tasks fully uses the processor if

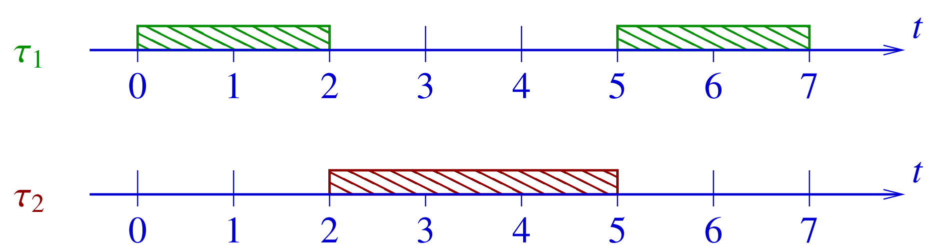
- this set of tasks is schedulable, and
- any increase of the execution time of a task (and hence of the processor load factor) yields a set of tasks that is not schedulable anymore.

Notes:

- Thanks to Theorem 2, checking whether a set of tasks is schedulable or not can be done by assigning RMS priorities to those tasks.
- A set of tasks that has a processor load factor less than 1 is not necessarily schedulable:

  Example:

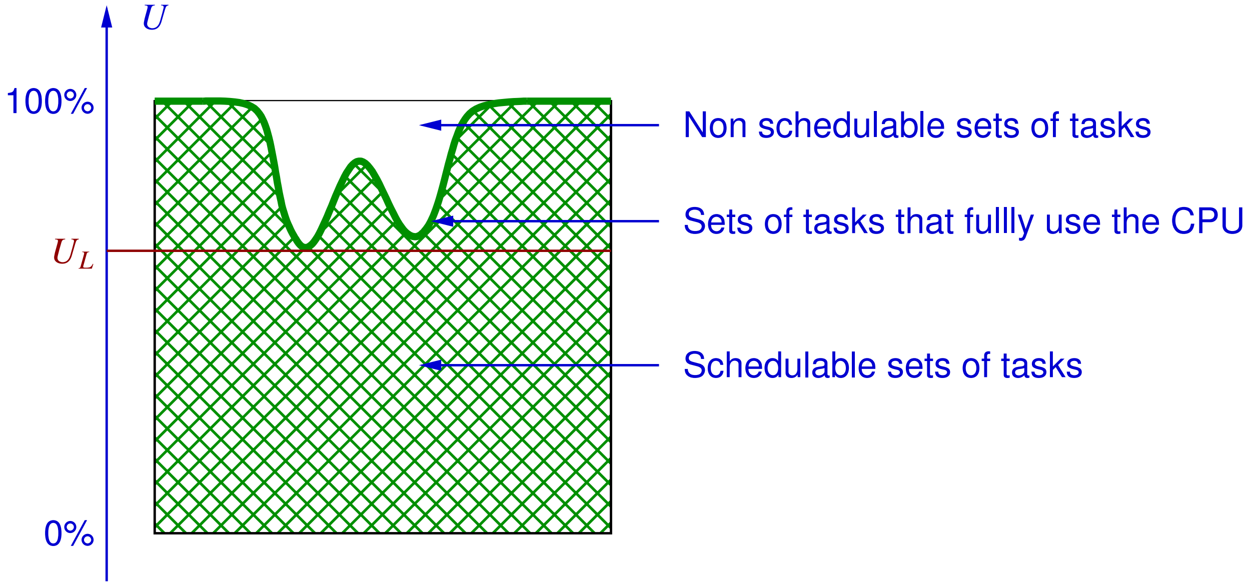
$$\left.\begin{array}{ll} \tau_1 : & T_1 = 5,\ C_1 = 2 \\ \tau_2 : & T_2 = 7,\ C_2 = 4 \end{array}\right\} \ U = \frac{2}{5} + \frac{4}{7} \approx 97\%$$



Classifying sets of tasks

The set of sets of tasks can be partitioned into three classes:

- The non schedulable sets of tasks.
- The sets of tasks that fully use the processor.
- The schedulable sets of tasks that do not fully use the processor.



The best lower bound U_L on the processor load factor of the sets of tasks that fully use the processor is such that:

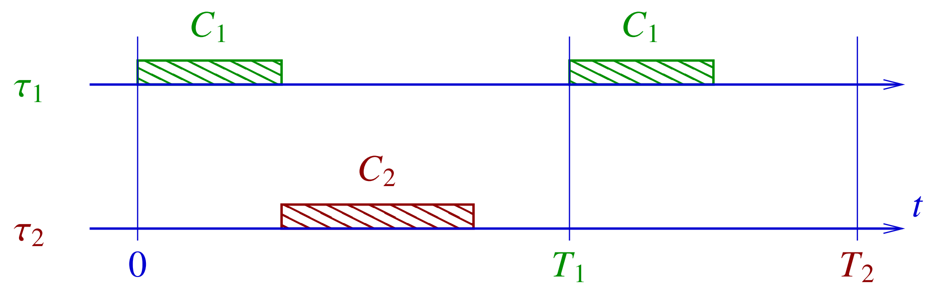
- If the processor load factor of a set of tasks is less than or equal to U_L, then this set of tasks is schedulable (regardless of the periods and execution times of the tasks!).
- If the processor load factor of a set of tasks is greater than U_L, then this set of tasks may or may not be schedulable, depending on the details of the tasks.

U_L: Case of two tasks

Let τ_1 and τ_2 be two tasks with respective periods and execution times T_1, T_2 and C_1, C_2. We assume $T_1 < T_2$. According to the RMS strategy, we assign a higher priority to τ_1.

During a critical zone of τ_2, the number of execution requests for τ_1 is equal to $\left\lceil \frac{T_2}{T_1} \right\rceil$.

- If all the executions of τ_1 in the interval $[0, T_2]$ terminate earlier than or at $t = T_2$.



The following condition is satisfied:

$$C_1 \leq T_2 - T_1 \left\lfloor \frac{T_2}{T_1} \right\rfloor.$$

For a given value of C_1, the largest possible value of C_2 is given by

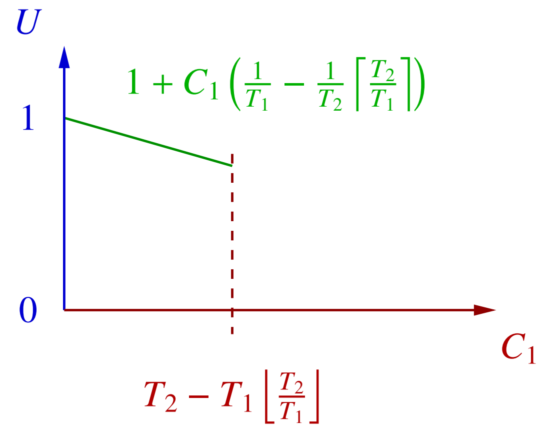
$$C_2 = T_2 - C_1 \left\lceil \frac{T_2}{T_1} \right\rceil.$$

It follows that the highest possible processor load factor is equal to

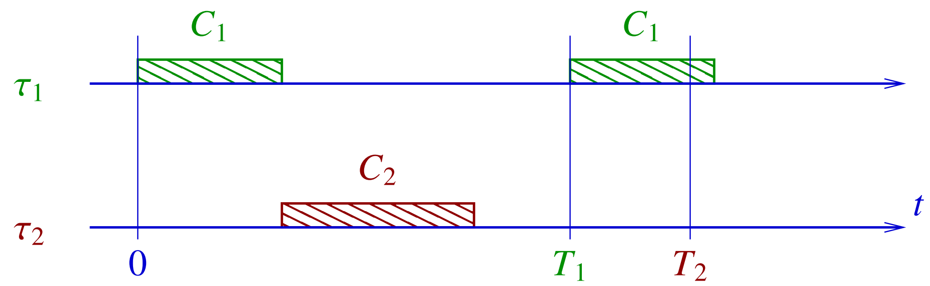
$$\begin{aligned} U &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\ &= 1 + C_1 \left(\frac{1}{T_1} - \frac{1}{T_2} \left\lceil \frac{T_2}{T_1} \right\rceil \right). \end{aligned}$$

Note that we have $\frac{1}{T_1} - \frac{1}{T_2} \left\lceil \frac{T_2}{T_1} \right\rceil \leq 0$.

Therefore, for given values of T_1 and T_2, the maximum processor load factor decreases with C_1.



- If an execution of τ_1 is still unfinished at $t = T_2$.



The following condition is satisfied:

$$C_1 > T_2 - T_1 \left\lfloor \frac{T_2}{T_1} \right\rfloor .$$

For a given value of C_1, the largest possible value of C_2 is given by

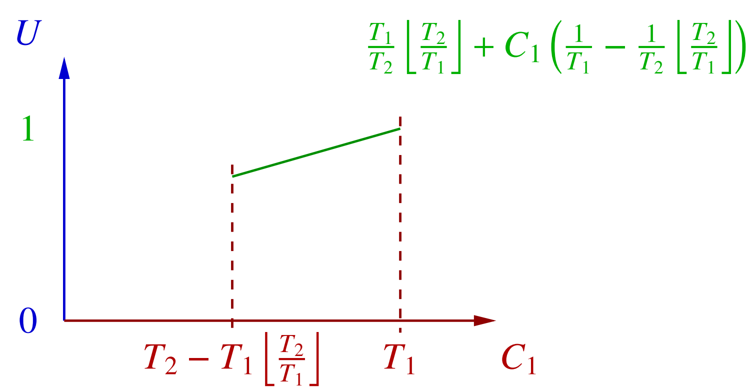
$$C_2 = (T_1 - C_1) \left\lfloor \frac{T_2}{T_1} \right\rfloor .$$

Hence, the highest possible processor load factor is equal to

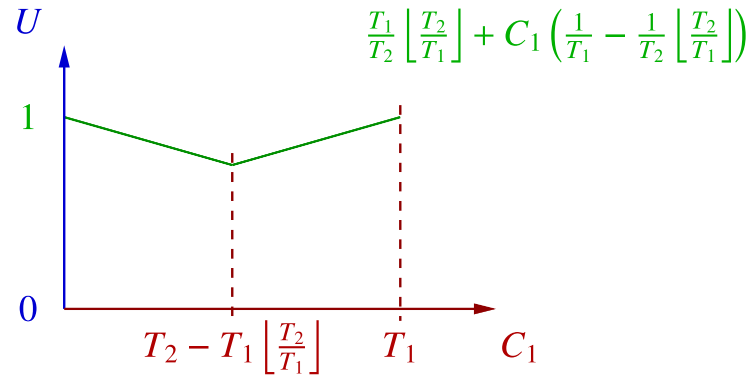
$$U = \frac{T_1}{T_2} \left\lfloor \frac{T_2}{T_1} \right\rfloor + C_1 \left(\frac{1}{T_1} - \frac{1}{T_2} \left\lfloor \frac{T_2}{T_1} \right\rfloor \right) .$$

For given values of T_1 and T_2, this expression increases with C_1, since

$$\frac{1}{T_1} - \frac{1}{T_2}\left\lfloor \frac{T_2}{T_1} \right\rfloor \geq 0.$$



Summary:



The smallest value of U corresponds to the boundary between the two cases, where we have

$$C_1 = T_2 - T_1\left\lfloor\frac{T_2}{T_1}\right\rfloor.$$

By introducing this value in the expression of U, one obtains

$$\begin{aligned}
U &= \frac{T_1}{T_2}\left\lfloor\frac{T_2}{T_1}\right\rfloor + \left(T_2 - T_1\left\lfloor\frac{T_2}{T_1}\right\rfloor\right)\left(\frac{1}{T_1} - \frac{1}{T_2}\left\lfloor\frac{T_2}{T_1}\right\rfloor\right)\\
&= \frac{T_1}{T_2}\left\lfloor\frac{T_2}{T_1}\right\rfloor + \frac{T_2}{T_1} - 2\left\lfloor\frac{T_2}{T_1}\right\rfloor + \frac{T_1}{T_2}\left\lfloor\frac{T_2}{T_1}\right\rfloor^2.
\end{aligned}$$

Let us define $I = \left\lfloor \frac{T_2}{T_1} \right\rfloor$ and $f = \frac{T_2}{T_1} - \left\lfloor \frac{T_2}{T_1} \right\rfloor$.

The previous expression becomes

$$\begin{aligned} U &= \frac{I}{I+f} + (I+f) - 2I + \frac{I^2}{I+f} \\ &= 1 - f\frac{1-f}{I+f}. \end{aligned}$$

The smallest possible value of U is obtained with $I = 1$. We then have

$$U = 1 - f\frac{1-f}{1+f},$$

and

$$\frac{dU}{df} = \frac{f^2+2f-1}{(1+f)^2}.$$

The best lower bound U_L on U is thus obtained with $I = 1$ and $f = -1 + \sqrt{2}$:

$$U_L = 1 - (\sqrt{2}-1)\left(\frac{2-\sqrt{2}}{\sqrt{2}}\right) = 2(\sqrt{2}-1) \approx 0.83.$$

Case of two tasks: Conclusions

Theorem 3: If a set of two periodic tasks has a processor load factor that is less than or equal to $2(\sqrt{2} - 1)$, then this set of tasks is schedulable.

Notes:

- This sufficient criterion is independent from the periods and execution times of the tasks.
- In the particular case where T_2 is an integer multiple of T_1, one has $f = 0$, hence

$$U_L = 1.$$

All pairs of tasks satisfying this condition (and such that $\frac{C_1}{T_1} + \frac{C_2}{T_2} \leq 1$!) are thus schedulable.

U_L: Case of n tasks

The goal is now to compute the value of U_L

- for a given number n of tasks, and
- for any number of tasks.

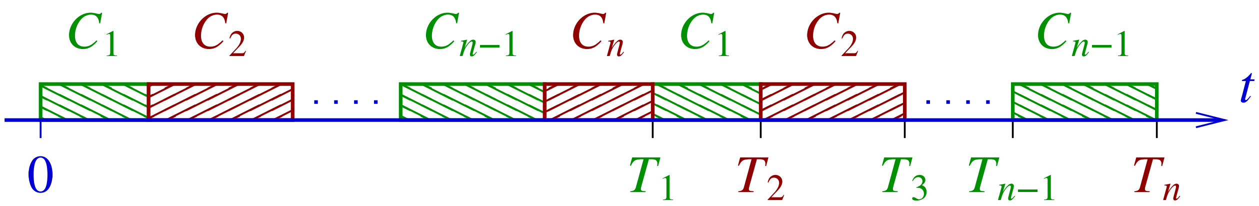
The first step is to establish an intermediate result:

Lemma 1: Let $\tau_1, \tau_2, \ldots, \tau_n$ be periodic tasks with the respective periods and execution times $T_1, T_2, \ldots, T_n$ and $C_1, C_2, \ldots, C_n$, such that

- This set of tasks fully uses the processor,
- $0 < T_1 < T_2 < \cdots < T_{n-1} < T_n < 2T_1$,
- The processor load factor of this set of tasks is minimum among all sets of tasks that fully use the processor.

In this case, one has

$$
\begin{aligned}
C_1 &= T_2 - T_1, \\
C_2 &= T_3 - T_2, \\
&\vdots \\
C_{n-1} &= T_n - T_{n-1}, \\
C_n &= T_n - 2(C_1 + C_2 + \cdots C_{n-1}) \\
&= 2T_1 - T_n.
\end{aligned}
$$

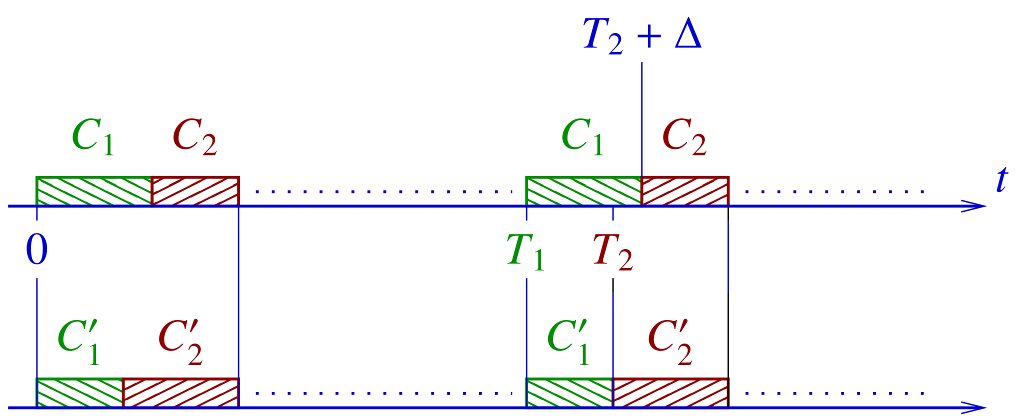


Proof: By contradiction, let us show that we must have $C_1 = T_2 - T_1$.

- If $C_1 = T_2 - T_1 + \Delta$, with $\Delta > 0$.

  We modify the execution time of tasks in the following way:

$$\begin{aligned}
C'_1 &= C_1 - \Delta, \\
C'_2 &= C_2 + \Delta, \\
C'_3 &= C_3, \\
&\vdots \\
C'_{n-1} &= C_{n-1}, \\
C'_n &= C_n.
\end{aligned}$$



After the modification, the new set of tasks still fully uses the processor. However, the processor load factor now becomes

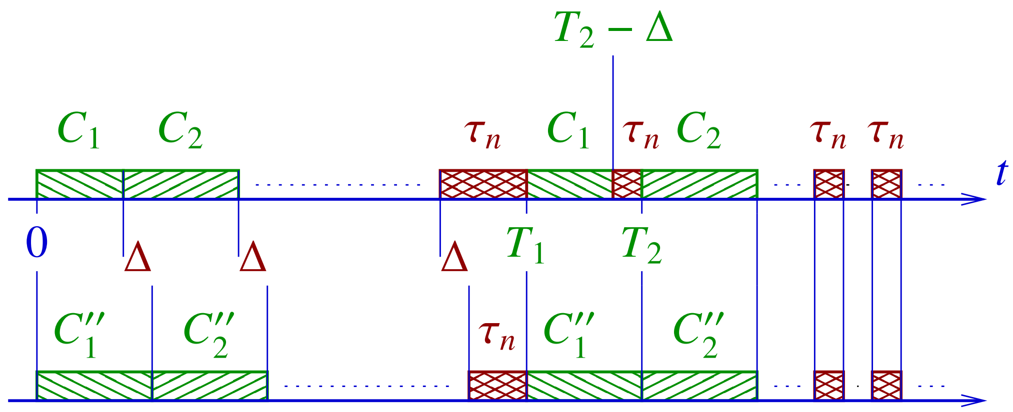
$$U' = U - \frac{\Delta}{T_1} + \frac{\Delta}{T_2} < U,$$

which contradicts the hypothesis that U is minimum.

- If $C_1 = T_2 - T_1 - \Delta$, with $\Delta > 0$.

  We now modify the execution time of tasks as follows:

$$\begin{aligned} C_1'' &= C_1 + \Delta, \\ C_2'' &= C_2, \\ C_3'' &= C_3, \\ &\vdots \\ C_{n-1}'' &= C_{n-1}, \\ C_n'' &= C_n - 2\Delta. \end{aligned}$$



The resulting set of tasks fully uses the processor. The processor load factor becomes

$$U' = U + \frac{\Delta}{T_1} - \frac{2\Delta}{T_n}.$$

Since we have by hypothesis $T_n < 2T_1$, this property contradicts $U' < U$.

By similar reasoning, one obtains successively

$$\begin{aligned} C_2 &= T_3 - T_2, \\ C_3 &= T_4 - T_3, \\ &\vdots \\ C_{n-1} &= T_n - T_{n-1}. \end{aligned}$$

Since the processor is fully used, one finally gets

$$C_n = T_n - 2(C_1 + C_2 + \cdots C_{n-1}).$$

Corollary: For each set of tasks that satisfies the hypotheses of Lemma 1, the processor load factor is equal to

$$\begin{aligned} U &= \frac{T_2 - T_1}{T_1} + \frac{T_3 - T_2}{T_2} + \cdots + \frac{T_n - T_{n-1}}{T_{n-1}} \\ &\quad + \frac{2T_1 - T_n}{T_n} \\ &= \frac{T_2}{T_1} + \frac{T_3}{T_2} + \cdots + \frac{T_n}{T_{n-1}} + 2\frac{T_1}{T_n} - n. \end{aligned}$$

For each $i = 1, 2, \ldots, n-1$, let us define $q_i = \dfrac{T_{i+1}}{T_i}$. We then have

$$U = q_1 + q_2 + \cdots + q_{n-1} + \frac{2}{q_1 q_2 \cdots q_{n-1}} - n,$$

and thus for each i,

$$\frac{\partial U}{\partial q_i} = 1 - 2\frac{q_1 q_2 \cdots q_{i-1} q_{i+1} \cdots q_{n-1}}{(q_1 q_2 \cdots q_{n-1})^2}.$$

The best lower bound U_L of U therefore corresponds to

$$\begin{aligned} \frac{\partial U}{\partial q_i} &= 0 \\ 1 - \frac{1}{q_i}.\frac{2}{q_1 q_2 \cdots q_{n-1}} &= 0. \end{aligned}$$

For each i, one has

$$q_i = \frac{2}{q_1 q_2 \cdots q_{n-1}},$$

hence

$$q_1 = q_2 = \cdots = q_{n-1} = 2^{\frac{1}{n}}.$$

By introducing these values in the expression of U, one obtains

$$\begin{aligned} U_L &= (n-1)2^{\frac{1}{n}} + \frac{2}{2^{\frac{n-1}{n}}} - n \\ &= (n-1)2^{\frac{1}{n}} + 2^{\frac{1}{n}} - n \\ &= n(2^{\frac{1}{n}} - 1). \end{aligned}$$

We thus have the following result:

Theorem 4: If the periods $T_1, T_2, \ldots, T_n$ of a set of n tasks are such that

$$0 < T_1 < T_2 < \cdots < T_{n-1} < T_n < 2T_1,$$

with a processor load factor that is less than or equal to $n(2^{\frac{1}{n}} - 1)$, then this set of tasks is schedulable.

In the hypotheses of Theorem 4, the constraint over the task periods is actually not necessary:

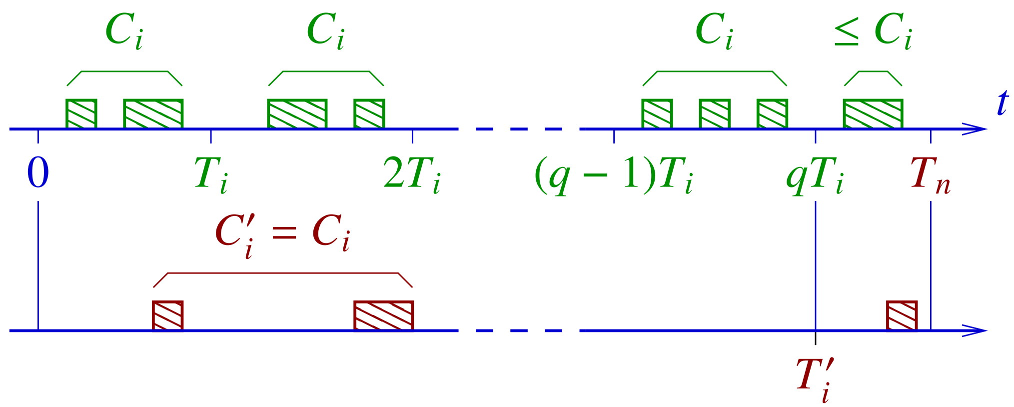
Theorem 5: If a set of n periodic tasks has a processor load factor that is less than or equal to $n(2^{\frac{1}{n}} - 1)$, then this set of tasks is schedulable.

Proof: Let $\tau_1, \tau_2, \ldots, \tau_n$ be tasks with respective periods and execution times $T_1, T_2, \ldots, T_n$ and $C_1, C_2, \ldots, C_n$. We assume that this set of tasks fully uses the processor.

If there exists $i \in \{1, 2, \ldots, n-1\}$ such that $2T_i \leq T_n$, then we define $q = \left\lfloor \dfrac{T_n}{T_i} \right\rfloor$ and $r = T_n - qT_i$ (we thus have $q > 1$ and $r \geq 0$).

We modify the set of tasks in the following way:

- We replace τ_i by τ'_i with the period $T'_i = qT_i$ and the execution time $C'_i = C_i$.
- We replace τ_n by τ'_n, with the period $T'_n = T_n$ and an execution time C'_n chosen so as to fully use the processor.



In the critical zone of τ_n, the difference between the execution times needed by τ_i and τ_i' is at most equal to $(q-1)C_i$. Therefore, one has

$$C_n' - C_n \leq (q-1)C_i.$$

After modifying the set of tasks, the processor load factor U' becomes equal to

$$U' \leq U + \frac{C_i'}{T_i'} - \frac{C_i}{T_i} + \frac{(q-1)C_i}{T_n}$$

where U is the processor load factor of the initial set of tasks.

One then obtains

$$U' \leq U + C_i\left(\frac{1}{qT_i} - \frac{1}{T_i} + \frac{q-1}{T_n}\right).$$

Since we have $qT_i \leq T_n$, this leads to

$$\begin{aligned} \frac{1}{qT_i} - \frac{1}{T_i} + \frac{q-1}{T_n} &\leq \frac{1}{qT_i} - \frac{1}{T_i} + \frac{q-1}{qT_i} \\ &\leq 0. \end{aligned}$$

As a consequence, we have $U' \leq U$. This implies that our modification of the set of tasks did not increase the processor load factor.

By repeatedly performing such a modification, one eventually obtains a set of tasks to which Theorem 4 can be applied.

The limit processor load factor

The value of U_L decreases with the number n of tasks. Indeed,

$$\begin{aligned}\frac{dU_L}{dn} &= \left(1 - \frac{\ln 2}{n}\right) 2^{\frac{1}{n}} - 1 \\ &= (1-x)e^x - 1,\end{aligned}$$

by defining $x = \dfrac{\ln 2}{n}$. Let us show that we have

$$(1-x)e^x < 1$$

for all $x > 0$ (which implies $\dfrac{dU_L}{dn} < 0$ for all $n > 0$).

For all $x > 0$, we have $e^x = 1 + x + \dfrac{x^2}{2!} + \dfrac{x^3}{3!} + \cdots$, hence

$$\begin{aligned}(1-x)e^x &= (1-x) + (1-x)x + (1-x)\frac{x^2}{2!} + (1-x)\frac{x^3}{3!} + \cdots \\ &= 1 - \left(1 - \frac{1}{2!}\right)x^2 - \left(\frac{1}{2!} - \frac{1}{3!}\right)x^3 - \left(\frac{1}{3!} - \frac{1}{4!}\right)x^4 + \cdots \\ &< 1.\end{aligned}$$

For an asymptotically large number of tasks, we obtain

$$\begin{aligned}
\lim_{n\to\infty} U_L(n) &= \lim_{n\to\infty} n(2^{\frac{1}{n}} - 1) \\
&= \lim_{n\to\infty} \frac{2^{\frac{1}{n}} - 1}{\frac{1}{n}} \\
&= \lim_{n\to\infty} \frac{\frac{\ln 2}{n^2} 2^{\frac{1}{n}}}{\frac{1}{n^2}} \\
&= \ln 2 \\
&\approx 0.69
\end{aligned}$$

In summary, we have the following result:

Theorem 6: If a set of periodic tasks has a processor load factor that is less than or equal to ln 2, then this set of tasks is schedulable.

Conclusion: The following algorithm can be used for checking efficiently whether a set of n periodic tasks with a processor load factor equal to U is schedulable or not:

1. If $U > 100\%$, then the set of tasks is not schedulable;
2. If $U \leq 69\%$, then the set of tasks is schedulable;
3. If $U \leq n(2^{\frac{1}{n}} - 1)$, then the set of tasks is schedulable;
4. Otherwise, one performs an exact scheduling simulation, based on a RMS priorities assignment.

Notes

- In situations where $U \leq 69\%$ for the periodic tasks, the processor does not have to remain unused during 31% of the time! One can instead run low-priority tasks that are not bound by real-time constraints.

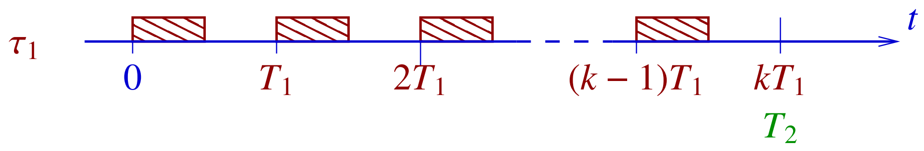
- For some specific class of sets of tasks, one can obtain $U_L = 100\%$, which guarantees that every set of tasks for which $U \leq 100\%$ is schedulable.

  Example: Let $\tau_1, \tau_2, \ldots, \tau_n$ be a set of tasks with respective periods and execution times $T_1, T_2, \ldots, T_n$ and $C_1, C_2, \ldots, C_n$, such that

  - $0 < T_1 \leq T_2 \leq \cdots \leq T_n$,
  - $\forall i, j \ : i < j \ \Rightarrow T_j$ is an integer multiple of T_i,
  - $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$.

Let us show that this set of tasks is schedulable.

The critical zone of τ_2 contains $\dfrac{T_2}{T_1}$ complete executions of τ_1:



Similarly, for each $j \in \{2, 3, \ldots, n\}$, the critical zone of τ_j contains

$\dfrac{T_j}{T_1}$ complete executions of τ_1,

⋮

$\dfrac{T_j}{T_{j-1}}$ complete executions of τ_{j-1}.

The condition that must be satisfied in order to finish the execution of τ_j before the end of its critical zone is thus

$$C_j \leq T_j - \frac{T_j}{T_1}C_1 - \frac{T_j}{T_2}C_2 - \cdots - \frac{T_j}{T_{j-1}}C_{j-1}$$

After simplification, this condition becomes

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_j}{T_j} \leq 1,$$

which immediately follows from the hypothesis $U \leq 1$.