Real-Time Operating Systems (0_KRI)
Utilization-Based Sched. Tests

Ivan Cibrario Bertolotti

IEIIT-CNR / Politecnico di Torino

Academic Year 2006-2007

Outline

1. Motivation and Definitions
2. Necessary Schedulability Test
3. Sufficient Schedulability Test for RMS
4. Necessary/Sufficient Sched. Test for EDF

Motivation and Initial Hypotheses

Our goal is to define several schedulability tests for RMS and EDF which, from very simple calculations based on the tasks' period and execution time, state either a necessary or a sufficient schedulability condition (or both) for a given task set.

- In the following, unless otherwise specified, we will assume that the basic process model is acceptable and being used.
- Moreover, we assume that a single-processor system is being used.
- Of course, sufficient (but not necessary) schedulability tests will not be exact, but somewhat "pessimistic".

Motivation and Initial Hypotheses

Our goal is to define several schedulability tests for RMS and EDF which, from very simple calculations based on the tasks' period and execution time, state either a necessary or a sufficient schedulability condition (or both) for a given task set.

- In the following, unless otherwise specified, we will assume that the basic process model is acceptable and being used.
- Moreover, we assume that a single-processor system is being used.
- Of course, sufficient (but not necessary) schedulability tests will not be exact, but somewhat "pessimistic".

Motivation and Initial Hypotheses

Our goal is to define several schedulability tests for RMS and EDF which, from very simple calculations based on the tasks' period and execution time, state either a necessary or a sufficient schedulability condition (or both) for a given task set.

- In the following, unless otherwise specified, we will assume that the basic process model is acceptable and being used.
- Moreover, we assume that a single-processor system is being used.
- Of course, sufficient (but not necessary) schedulability tests will not be exact, but somewhat "pessimistic".

Motivation and Initial Hypotheses

Our goal is to define several schedulability tests for RMS and EDF which, from very simple calculations based on the tasks' period and execution time, state either a necessary or a sufficient schedulability condition (or both) for a given task set.

- In the following, unless otherwise specified, we will assume that the basic process model is acceptable and being used.
- Moreover, we assume that a single-processor system is being used.
- Of course, sufficient (but not necessary) schedulability tests will not be exact, but somewhat "pessimistic".

Processor Utilization Factor

Definition

Given a set of N periodic tasks $\Gamma = \{\tau_i, \ldots, \tau_N\}$ the processor utilization factor U is the fraction of processor time spent in the execution of the task set.

- Since C_i/T_i is the fraction of processor time spent executing task τ_i, the utilization for the whole set of N tasks is:

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

- The processor utilization factor is a measure of the computational load imposed on the processor by a given task set.

Processor Utilization Factor

Definition

Given a set of N periodic tasks $\Gamma = \{\tau_i, \ldots, \tau_N\}$ the processor utilization factor U is the fraction of processor time spent in the execution of the task set.

- Since C_i/T_i is the fraction of processor time spent executing task τ_i, the utilization for the whole set of N tasks is:

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

- The processor utilization factor is a measure of the computational load imposed on the processor by a given task set.

Processor Utilization Factor

Definition

Given a set of N periodic tasks $\Gamma = \{\tau_i, \ldots, \tau_N\}$ the processor utilization factor U is the fraction of processor time spent in the execution of the task set.

- Since C_i/T_i is the fraction of processor time spent executing task τ_i, the utilization for the whole set of N tasks is:

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

- The processor utilization factor is a measure of the computational load imposed on the processor by a given task set.

Full Utilization

- Given a task set Γ, its processor utilization factor can be increased by increasing the execution times C_i of the tasks.
- For a given scheduling algorithm A, there exists a maximum value of U below which the task set Γ is schedulable, but for which any increase in the C_i of any of the tasks in the task set will make it no longer schedulable.
- The limit depends on the task set Γ (namely, on the relationships among tasks' periods), and on the scheduling algorithm A.

Definition

A task set Γ is said to fully utilize the processor with a given scheduling algorithm A if it is schedulable by A, but any increase in the C_i of any of its tasks will make it no longer schedulable. The corresponding upper bound of the utilization factor is denoted as $U_{\text{ub}}(\Gamma, A)$.

Full Utilization

- Given a task set Γ, its processor utilization factor can be increased by increasing the execution times C_i of the tasks.
- For a given scheduling algorithm A, there exists a maximum value of U below which the task set Γ is schedulable, but for which any increase in the C_i of any of the tasks in the task set will make it no longer schedulable.
- The limit depends on the task set Γ (namely, on the relationships among tasks' periods), and on the scheduling algorithm A.

Definition

A task set Γ is said to fully utilize the processor with a given scheduling algorithm A if it is schedulable by A, but any increase in the C_i of any of its tasks will make it no longer schedulable. The corresponding upper bound of the utilization factor is denoted as $U_{\mathrm{ub}}(\Gamma, A)$.

Full Utilization

- Given a task set Γ, its processor utilization factor can be increased by increasing the execution times C_i of the tasks.
- For a given scheduling algorithm A, there exists a maximum value of U below which the task set Γ is schedulable, but for which any increase in the C_i of any of the tasks in the task set will make it no longer schedulable.
- The limit depends on the task set Γ (namely, on the relationships among tasks' periods), and on the scheduling algorithm A.

Definition

A task set Γ is said to fully utilize the processor with a given scheduling algorithm A if it is schedulable by A, but any increase in the C_i of any of its tasks will make it no longer schedulable. The corresponding upper bound of the utilization factor is denoted as $U_{\text{ub}}(\Gamma, A)$.

Full Utilization

- Given a task set Γ, its processor utilization factor can be increased by increasing the execution times C_i of the tasks.
- For a given scheduling algorithm A, there exists a maximum value of U below which the task set Γ is schedulable, but for which any increase in the C_i of any of the tasks in the task set will make it no longer schedulable.
- The limit depends on the task set Γ (namely, on the relationships among tasks' periods), and on the scheduling algorithm A.

Definition

A task set Γ is said to fully utilize the processor with a given scheduling algorithm A if it is schedulable by A, but any increase in the C_i of any of its tasks will make it no longer schedulable. The corresponding upper bound of the utilization factor is denoted as $U_{\mathrm{ub}}(\Gamma, A)$.

Least Upper Bound

- It is interesting (and useful) to ask how large the utilization factor can be, in order to guarantee the schedulability of any task set Γ by a given scheduling algorithm A.
- In order to do this, we must determine the minimum value of $U_{\text{ub}}(\Gamma, A)$ over all task sets Γ which fully utilize the processor.

Definition

For a given scheduling algorithm A, the least upper bound $U_{\text{lub}}(A)$ of the utilization factor is defined as:

$$U_{\text{lub}}(A) = \min_{\Gamma} U_{\text{ub}}(\Gamma, A)$$

where Γ represents all task sets which fully utilize the processor.

Least Upper Bound

- It is interesting (and useful) to ask how large the utilization factor can be, in order to guarantee the schedulability of any task set Γ by a given scheduling algorithm A.
- In order to do this, we must determine the minimum value of $U_{\text{ub}}(\Gamma, A)$ over all task sets Γ which fully utilize the processor.

Definition

For a given scheduling algorithm A, the least upper bound $U_{\text{lub}}(A)$ of the utilization factor is defined as:

$$U_{\text{lub}}(A) = \min_{\Gamma} U_{\text{ub}}(\Gamma, A)$$

where Γ represents all task sets which fully utilize the processor.

Least Upper Bound

- It is interesting (and useful) to ask how large the utilization factor can be, in order to guarantee the schedulability of any task set Γ by a given scheduling algorithm A.
- In order to do this, we must determine the minimum value of $U_{\text{ub}}(\Gamma, A)$ over all task sets Γ which fully utilize the processor.

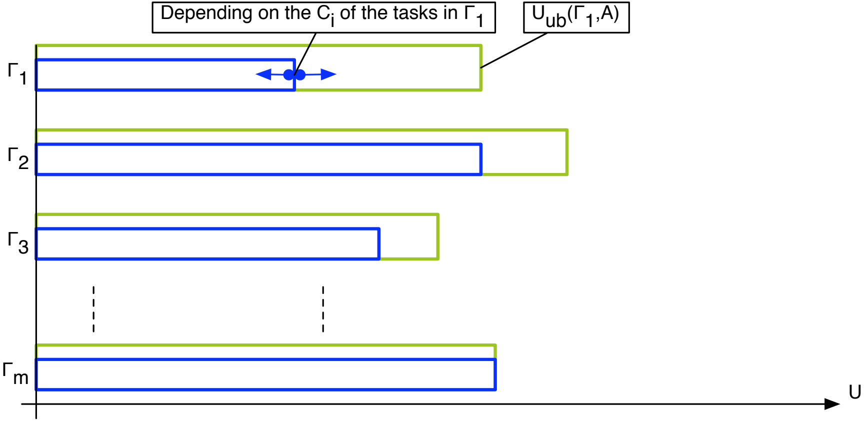
Definition

For a given scheduling algorithm A, the least upper bound $U_{\text{lub}}(A)$ of the utilization factor is defined as:

$$U_{\text{lub}}(A) = \min_{\Gamma} U_{\text{ub}}(\Gamma, A)$$

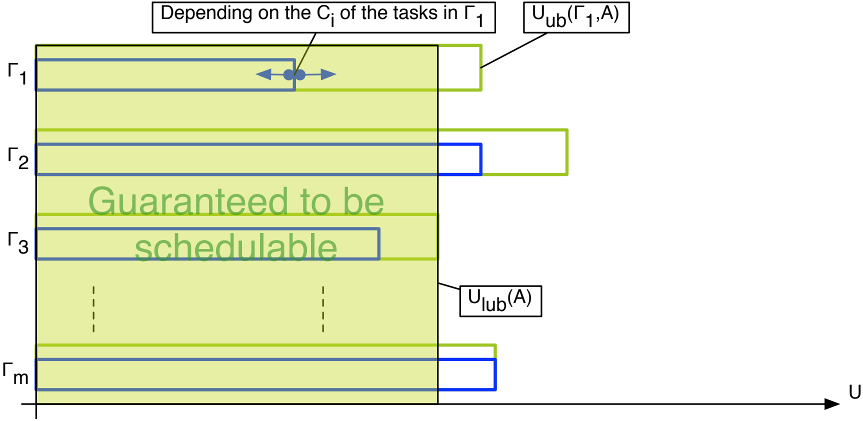
where Γ represents all task sets which fully utilize the processor.

Pictorial Representation



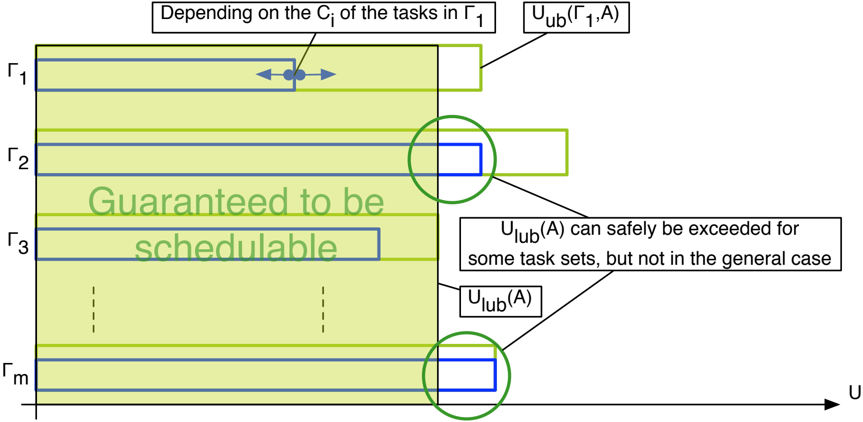
- The task sets $\Gamma_1, \ldots, \Gamma_m$ differ for the number of tasks, as well as for the relationship among task periods.
- When scheduled by A, each of them can reach a certain upper bound of utilization $U_{\mathrm{ub}}(\Gamma_i, A)$, by varying the task execution times.

Pictorial Representation



- Each task set has its own upper bound, and they will all be different in general.
- However, since $U_{lub}(A)$ is the minimum upper bound over all possible task sets, any task set whose utilization factor is below $U_{lub}(A)$ will be schedulable by A.

Pictorial Representation



- On the other hand, $U_{\text{lub}}(A)$ can sometimes be exceeded, but not in the general case.
- Namely, this is possible only if the task periods are suitably related.

Necessary Schedulability Test

Theorem

If the processor utilization factor U of a task set Γ is greater than one (that is, if $U > 1$), then the task set is not schedulable, regardless of the scheduling algorithm.

- Recalling the definition of U for a set of N tasks $\Gamma = \{\tau_1, \ldots, \tau_N\}$, the necessary condition can be written as:

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

- Intuitively: It is impossible to allocate to the tasks a fraction of CPU time greater than the total “quantity” of CPU time available.
- On the other hand, if the necessary condition is satisfied, we still do not have any guarantee of schedulability.

Necessary Schedulability Test

Theorem

If the processor utilization factor U of a task set Γ is greater than one (that is, if $U > 1$), then the task set is not schedulable, regardless of the scheduling algorithm.

- Recalling the definition of U for a set of N tasks $\Gamma = \{\tau_1, \ldots, \tau_N\}$, the necessary condition can be written as:

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

- Intuitively: It is impossible to allocate to the tasks a fraction of CPU time greater than the total "quantity" of CPU time available.
- On the other hand, if the necessary condition is satisfied, we still do not have any guarantee of schedulability.

Necessary Schedulability Test

Theorem

If the processor utilization factor U of a task set Γ is greater than one (that is, if $U > 1$), then the task set is not schedulable, regardless of the scheduling algorithm.

- Recalling the definition of U for a set of N tasks $\Gamma = \{\tau_1, \ldots, \tau_N\}$, the necessary condition can be written as:

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

- Intuitively: It is impossible to allocate to the tasks a fraction of CPU time greater than the total "quantity" of CPU time available.
- On the other hand, if the necessary condition is satisfied, we still do not have any guarantee of schedulability.

Necessary Schedulability Test

Theorem

If the processor utilization factor U of a task set Γ is greater than one (that is, if $U > 1$), then the task set is not schedulable, regardless of the scheduling algorithm.

- Recalling the definition of U for a set of N tasks $\Gamma = \{\tau_1, \ldots, \tau_N\}$, the necessary condition can be written as:

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

- Intuitively: It is impossible to allocate to the tasks a fraction of CPU time greater than the total "quantity" of CPU time available.
- On the other hand, if the necessary condition is satisfied, we still do not have any guarantee of schedulability.

Proof (I)

We will prove the theorem by *reductio ad absurdum*.

- Let T be the product of all the task periods:

$$T = \prod_{i=1}^{N} T_i$$

- Within the time interval T, each task τ_i will be executed an integral number of times, T / T_i.
- For each instance, the task needs a computation time of C_i to be schedulable, hence the total computation demand of τ_i within the time interval T is:

$$\frac{T}{T_i} C_i$$

Proof (I)

We will prove the theorem by *reductio ad absurdum*.

- Let T be the product of all the task periods:

$$T = \prod_{i=1}^{N} T_i$$

- Within the time interval T, each task τ_i will be executed an integral number of times, T/T_i.
- For each instance, the task needs a computation time of C_i to be schedulable, hence the total computation demand of τ_i within the time interval T is:

$$\frac{T}{T_i} C_i$$

Proof (I)

We will prove the theorem by *reductio ad absurdum*.

- Let T be the product of all the task periods:

$$T = \prod_{i=1}^{N} T_i$$

- Within the time interval T, each task τ_i will be executed an integral number of times, T/T_i.
- For each instance, the task needs a computation time of C_i to be schedulable, hence the total computation demand of τ_i within the time interval T is:

$$\frac{T}{T_i} C_i$$

Proof (I)

We will prove the theorem by *reductio ad absurdum*.

- Let T be the product of all the task periods:

$$T = \prod_{i=1}^{N} T_i$$

- Within the time interval T, each task τ_i will be executed an integral number of times, T/T_i.
- For each instance, the task needs a computation time of C_i to be schedulable, hence the total computation demand of τ_i within the time interval T is:

$$\frac{T}{T_i} C_i$$

Proof (II)

- We can write the total computation demand of all tasks C_{tot} within the time interval T as:

$$C_{\text{tot}} = \sum_{i=1}^{N} \frac{T}{T_i} C_i$$

- On the other hand, if $U > 1$ we also have $TU > T$ and, recalling the definition of U, we can write:

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

- The same inequation can also be written as:

$$\sum_{i=1}^{N} T \frac{C_i}{T_i} > T \quad \text{that is,} \quad \sum_{i=1}^{N} \frac{T}{T_i} C_i > T$$

Proof (II)

- We can write the total computation demand of all tasks C_{tot} within the time interval T as:

$$C_{\text{tot}} = \sum_{i=1}^{N} \frac{T}{T_i} C_i$$

- On the other hand, if $U > 1$ we also have $TU > T$ and, recalling the definition of U, we can write:

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

- The same inequation can also be written as:

$$\sum_{i=1}^{N} T \frac{C_i}{T_i} > T \;\; \text{that is,} \;\; \sum_{i=1}^{N} \frac{T}{T_i} C_i > T$$

Proof (II)

- We can write the total computation demand of all tasks C_{tot} within the time interval T as:

$$C_{\text{tot}} = \sum_{i=1}^{N} \frac{T}{T_i} C_i$$

- On the other hand, if $U > 1$ we also have $TU > T$ and, recalling the definition of U, we can write:

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

- The same inequation can also be written as:

$$\sum_{i=1}^{N} T \frac{C_i}{T_i} > T \text{ that is, } \sum_{i=1}^{N} \frac{T}{T_i} C_i > T$$

Proof (conclusion)

Absurdity

The left-hand side of the last inequation is C_{tot}, hence we have:

$$C_{\text{tot}} > T$$

- This is absurd, because we are stating that the total CPU time demand of all tasks C_{tot}, within a certain time interval T, exceeds the time interval itself.
- The absurd derives from having supposed that the task set was schedulable.
- We did not use any property of the scheduling algorithm, hence this result does not depend on it (and is valid for any scheduling algorithm).

Proof (conclusion)

Absurdity

The left-hand side of the last inequation is C_{tot}, hence we have:

$$C_{\text{tot}} > T$$

- This is absurd, because we are stating that the total CPU time demand of all tasks C_{tot}, within a certain time interval T, exceeds the time interval itself.
- The absurd derives from having supposed that the task set was schedulable.
- We did not use any property of the scheduling algorithm, hence this result does not depend on it (and is valid for any scheduling algorithm).

Proof (conclusion)

Absurdity

The left-hand side of the last inequation is C_{tot}, hence we have:

$$C_{\text{tot}} > T$$

- This is absurd, because we are stating that the total CPU time demand of all tasks C_{tot}, within a certain time interval T, exceeds the time interval itself.
- The absurd derives from having supposed that the task set was schedulable.
- We did not use any property of the scheduling algorithm, hence this result does not depend on it (and is valid for any scheduling algorithm).

Proof (conclusion)

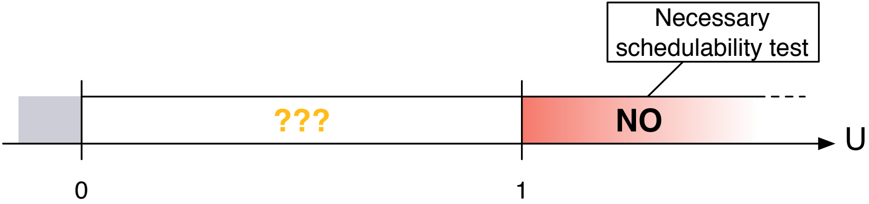
Absurdity

The left-hand side of the last inequation is C_{tot}, hence we have:

$$C_{\text{tot}} > T$$

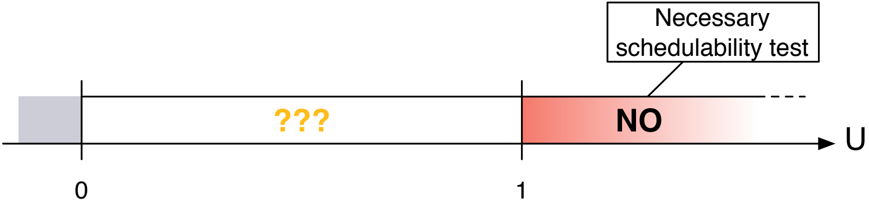
- This is absurd, because we are stating that the total CPU time demand of all tasks C_{tot}, within a certain time interval T, exceeds the time interval itself.
- The absurd derives from having supposed that the task set was schedulable.
- We did not use any property of the scheduling algorithm, hence this result does not depend on it (and is valid for any scheduling algorithm).

Summary and Usage Notes



- The necessary schedulability test provides a convenient way of concluding that a given task set Γ is not schedulable, by examining its utilization factor U.
- Instead, it shall not be applied (common mistake) to conclude that a task set is indeed schedulable because, if a task set passes the necessary schedulability test, this tell us nothing about its schedulability.

Summary and Usage Notes



- The necessary schedulability test provides a convenient way of concluding that a given task set Γ is not schedulable, by examining its utilization factor U.
- Instead, it shall not be applied (common mistake) to conclude that a task set is indeed schedulable because, if a task set passes the necessary schedulability test, this tell us nothing about its schedulability.

Extension to Multiprocessor Systems

The theorem can be extended to multiprocessor systems; in this case, for M CPUs, the necessary condition becomes:

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

- This kind of extension seems intuitive, but it is rarely possible.
- For example, the rate monotonic scheduling is provably optimum for single processor systems, but is not necessarily optimum for multiprocessor systems.

Extension to Multiprocessor Systems

The theorem can be extended to multiprocessor systems; in this case, for M CPUs, the necessary condition becomes:

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

- This kind of extension seems intuitive, but it is rarely possible.
- For example, the rate monotonic scheduling is provably optimum for single processor systems, but is not necessarily optimum for multiprocessor systems.

Extension to Multiprocessor Systems

The theorem can be extended to multiprocessor systems; in this case, for M CPUs, the necessary condition becomes:

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

- This kind of extension seems intuitive, but it is rarely possible.
- For example, the rate monotonic scheduling is provably optimum for single processor systems, but is not necessarily optimum for multiprocessor systems.

Sufficient Schedulability Test for RMS

Premise

We already showed that the RM priority assignment is optimum among all other static priority assignment schemes for single processor systems under the basic process model.

Roadmap:

- We will now show how to compute the least upper bound U_{lub} of the processor utilization for RM.
- From this, we will derive a sufficient schedulability test for RM so that, if a given task set Γ satisfies it, its schedulability will be guaranteed by RM.
- Of course, since the test will not be exact, its failure will give us no information about schedulability.

Sufficient Schedulability Test for RMS

Premise

We already showed that the RM priority assignment is optimum among all other static priority assignment schemes for single processor systems under the basic process model.

Roadmap:

- We will now show how to compute the least upper bound U_{lub} of the processor utilization for RM.
- From this, we will derive a sufficient schedulability test for RM so that, if a given task set Γ satisfies it, its schedulability will be guaranteed by RM.
- Of course, since the test will not be exact, its failure will give us no information about schedulability.

Sufficient Schedulability Test for RMS

Premise

We already showed that the RM priority assignment is optimum among all other static priority assignment schemes for single processor systems under the basic process model.

Roadmap:

- We will now show how to compute the least upper bound U_{lub} of the processor utilization for RM.
- From this, we will derive a sufficient schedulability test for RM so that, if a given task set Γ satisfies it, its schedulability will be guaranteed by RM.
- Of course, since the test will not be exact, its failure will give us no information about schedulability.

Sufficient Schedulability Test for RMS

Premise

We already showed that the RM priority assignment is optimum among all other static priority assignment schemes for single processor systems under the basic process model.

Roadmap:

- We will now show how to compute the least upper bound U_{lub} of the processor utilization for RM.
- From this, we will derive a sufficient schedulability test for RM so that, if a given task set Γ satisfies it, its schedulability will be guaranteed by RM.
- Of course, since the test will not be exact, its failure will give us no information about schedulability.

U_{lub} for Two Tasks

Let us consider a set of two periodic tasks $\{\tau_1, \tau_2\}$, with $T_1 < T_2$.

- According to the RM priority assignment, τ_1 will be the task with the highest priority.
- We will first compute the upper bound U_{ub} of their utilization factor, by setting the task computation times to fully utilize the processor.
- Then, to obtain U_{lub}, we will minimize U_{ub} over all the other task parameters.

U_{lub} for Two Tasks

Let us consider a set of two periodic tasks $\{\tau_1, \tau_2\}$, with $T_1 < T_2$.

- According to the RM priority assignment, τ_1 will be the task with the highest priority.
- We will first compute the upper bound U_{ub} of their utilization factor, by setting the task computation times to fully utilize the processor.
- Then, to obtain U_{lub}, we will minimize U_{ub} over all the other task parameters.

U_{lub} for Two Tasks

Let us consider a set of two periodic tasks $\{\tau_1, \tau_2\}$, with $T_1 < T_2$.

- According to the RM priority assignment, τ_1 will be the task with the highest priority.
- We will first compute the upper bound U_{ub} of their utilization factor, by setting the task computation times to fully utilize the processor.
- Then, to obtain U_{lub}, we will minimize U_{ub} over all the other task parameters.

U_{lub} for Two Tasks

Let us consider a set of two periodic tasks $\{\tau_1, \tau_2\}$, with $T_1 < T_2$.

- According to the RM priority assignment, τ_1 will be the task with the highest priority.
- We will first compute the upper bound U_{ub} of their utilization factor, by setting the task computation times to fully utilize the processor.
- Then, to obtain U_{lub}, we will minimize U_{ub} over all the other task parameters.

Achieving Full Utilization

As before, let F be the number of periods of τ_1 entirely contained within T_2:

$$F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$$

Without loss of generality, we will adjust C_2 to fully utilize the processor. Again, we must consider two cases:

1. The execution time C_1 is "short enough" so that the all the instances of τ_1 within the critical zone of τ_2 are completed before the next release of τ_2.
2. The execution of the last instance of τ_1 that starts within the critical zone of τ_2 overlaps the next release of τ_2.

Achieving Full Utilization

As before, let F be the number of periods of τ_1 entirely contained within T_2:

$$F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$$

Without loss of generality, we will adjust C_2 to fully utilize the processor. Again, we must consider two cases:

1. The execution time C_1 is "short enough" so that the all the instances of τ_1 within the critical zone of τ_2 are completed before the next release of τ_2.
2. The execution of the last instance of τ_1 that starts within the critical zone of τ_2 overlaps the next release of τ_2.

Achieving Full Utilization

As before, let F be the number of periods of τ_1 entirely contained within T_2:

$$F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$$

Without loss of generality, we will adjust C_2 to fully utilize the processor. Again, we must consider two cases:

1. The execution time C_1 is "short enough" so that the all the instances of τ_1 within the critical zone of τ_2 are completed before the next release of τ_2.
2. The execution of the last instance of τ_1 that starts within the critical zone of τ_2 overlaps the next release of τ_2.

Achieving Full Utilization

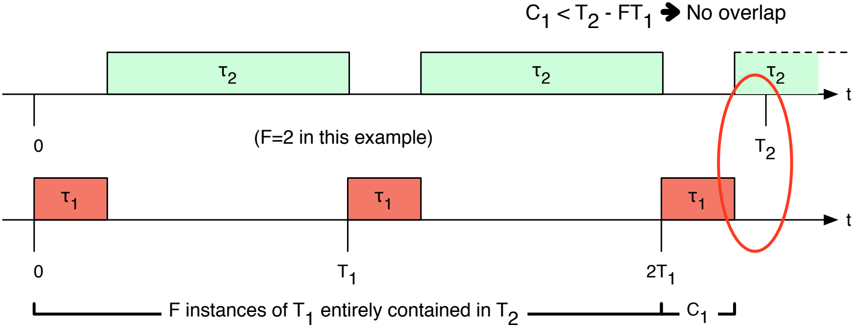
As before, let F be the number of periods of τ_1 entirely contained within T_2:

$$F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$$

Without loss of generality, we will adjust C_2 to fully utilize the processor. Again, we must consider two cases:

1. The execution time C_1 is "short enough" so that the all the instances of τ_1 within the critical zone of τ_2 are completed before the next release of τ_2.
2. The execution of the last instance of τ_1 that starts within the critical zone of τ_2 overlaps the next release of τ_2.

First Case



The largest possible value of C_2 is:

$$C_2 = T_2 - (F + 1)C_1$$

If we compute U for this value of C_2, we will obtain U_{ub}.

U_{ub} in the First Case

- By definition of U we have:

$$
\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{T_2 - (F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_1} - \frac{(F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right]
\end{aligned}
$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $FT_1 \leq T_2 < (F+1)T_1$ and the quantity between square brackets will be strictly negative.

U_{ub} is monotonically decreasing with respect to C_1.

U_{ub} in the First Case

- By definition of U we have:

$$
\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{T_2 - (F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_1} - \frac{(F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right]
\end{aligned}
$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $FT_1 \leq T_2 < (F+1)T_1$ and the quantity between square brackets will be strictly negative.

U_{ub} is monotonically decreasing with respect to C_1.

U_{ub} in the First Case

- By definition of U we have:

$$\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{T_2 - (F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_1} - \frac{(F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right]
\end{aligned}$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $FT_1 \leq T_2 < (F+1)T_1$ and the quantity between square brackets will be strictly negative.

U_{ub} is monotonically decreasing with respect to C_1.

U_{ub} in the First Case

- By definition of U we have:

$$
\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{T_2 - (F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_1} - \frac{(F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right]
\end{aligned}
$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $FT_1 \leq T_2 < (F+1)T_1$ and the quantity between square brackets will be strictly negative.

U_{ub} is monotonically decreasing with respect to C_1.

U_{ub} in the First Case

- By definition of U we have:

$$
\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{T_2 - (F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_1} - \frac{(F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right]
\end{aligned}
$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $FT_1 \le T_2 < (F+1)T_1$ and the quantity between square brackets will be strictly negative.

U_{ub} is monotonically decreasing with respect to C_1.

U_{ub} in the First Case

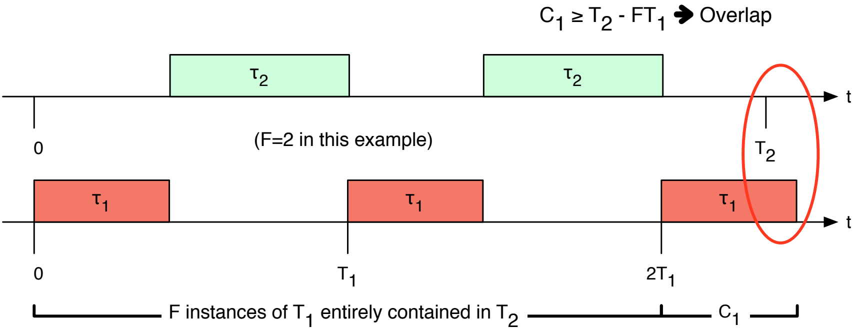
- By definition of U we have:

$$
\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{T_2 - (F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_1} - \frac{(F+1)C_1}{T_2} \\
&= 1 + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right]
\end{aligned}
$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $FT_1 \leq T_2 < (F+1)T_1$ and the quantity between square brackets will be strictly negative.

U_{ub} is monotonically decreasing with respect to C_1.

Second Case



The largest possible value of C_2 is:

$$C_2 = FT_1 - FC_1$$

Again, if we compute U for this value of C_2, we will obtain U_{ub}.

U_{ub} in the Second Case

- By definition of U we have:

$$
\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{FT_1 - FC_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_1} - F\frac{C_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - F\right]
\end{aligned}
$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $F \leq \frac{T_2}{T_1}$ and the quantity between square brackets will be either positive or zero.

U_{ub} is monotonically nondecreasing with respect to C_1.

U_{ub} in the Second Case

- By definition of U we have:

$$
\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{FT_1 - FC_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_1} - F\frac{C_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - F\right]
\end{aligned}
$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $F \leq \frac{T_2}{T_1}$ and the quantity between square brackets will be either positive or zero.

U_{ub} is monotonically nondecreasing with respect to C_1.

U_{ub} in the Second Case

- By definition of U we have:

$$\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{FT_1 - FC_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_1} - F\frac{C_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - F\right]
\end{aligned}$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $F \leq \frac{T_2}{T_1}$ and the quantity between square brackets will be either positive or zero.

U_{ub} is monotonically nondecreasing with respect to C_1.

U_{ub} in the Second Case

- By definition of U we have:

$$
\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{FT_1 - FC_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_1} - F\frac{C_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - F\right]
\end{aligned}
$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $F \leq \frac{T_2}{T_1}$ and the quantity between square brackets will be either positive or zero.

U_{ub} is monotonically nondecreasing with respect to C_1.

U_{ub} in the Second Case

- By definition of U we have:

$$
\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{FT_1 - FC_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_1} - F\frac{C_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - F\right]
\end{aligned}
$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $F \leq \frac{T_2}{T_1}$ and the quantity between square brackets will be either positive or zero.

U_{ub} is monotonically nondecreasing with respect to C_1.

U_{ub} in the Second Case

- By definition of U we have:

$$
\begin{aligned}
U_{\text{ub}} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} \\
&= \frac{C_1}{T_1} + \frac{FT_1 - FC_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_1} - F\frac{C_1}{T_2} \\
&= F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - F\right]
\end{aligned}
$$

- But, since $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, then $F \leq \frac{T_2}{T_1}$ and the quantity between square brackets will be either positive or zero.

U_{ub} is monotonically nondecreasing with respect to C_1.

Where is the Minimum?

1. In the first case, since U_{ub} is monotonically decreasing with respect to C_1, its value will be at its minimum when C_1 assumes its maximum allowed value.
2. In the second case, since U_{ub} is monotonically nondecreasing with respect to C_1, its value will be at its minimum when C_1 assumes its minimum allowed value.

Observation

Being $C_1 < T_2 - FT_1$ by hypothesis in the first case, and $C_1 \geq T_2 - FT_1$ in the second case, U_{ub} is at its minimum at the boundary between the two cases, that is, when:

$$C_1 = T_2 - FT_1$$

Where is the Minimum?

1. In the first case, since U_{ub} is monotonically decreasing with respect to C_1, its value will be at its minimum when C_1 assumes its maximum allowed value.
2. In the second case, since U_{ub} is monotonically nondecreasing with respect to C_1, its value will be at its minimum when C_1 assumes its minimum allowed value.

Observation

Being $C_1 < T_2 - FT_1$ by hypothesis in the first case, and $C_1 \geq T_2 - FT_1$ in the second case, U_{ub} is at its minimum at the boundary between the two cases, that is, when:

$$C_1 = T_2 - FT_1$$

Where is the Minimum?

1. In the first case, since U_{ub} is monotonically decreasing with respect to C_1, its value will be at its minimum when C_1 assumes its maximum allowed value.
2. In the second case, since U_{ub} is monotonically nondecreasing with respect to C_1, its value will be at its minimum when C_1 assumes its minimum allowed value.

Observation

Being $C_1 < T_2 - FT_1$ by hypothesis in the first case, and $C_1 \geq T_2 - FT_1$ in the second case, U_{ub} is at its minimum at the boundary between the two cases, that is, when:

$$C_1 = T_2 - FT_1$$

Computing the Minimum

- At this point, we can take either one of the expressions we derived for U_{ub} and substitute $C_1 = T_2 - FT_1$ into it.
- We can take either one, for example the second one, because both refer to the same situation from the scheduling point of view, hence they must both give the same result.
- It should be noted that the resulting expression for U_{ub} will still depend on the task periods T_1 and T_2 through F, hence we will have to minimize it with respect to F in order to find U_{lub}.

Computing the Minimum

- At this point, we can take either one of the expressions we derived for U_{ub} and substitute $C_1 = T_2 - FT_1$ into it.
- We can take either one, for example the second one, because both refer to the same situation from the scheduling point of view, hence they must both give the same result.
- It should be noted that the resulting expression for U_{ub} will still depend on the task periods T_1 and T_2 through F, hence we will have to minimize it with respect to F in order to find U_{lub}.

Computing the Minimum

- At this point, we can take either one of the expressions we derived for U_{ub} and substitute $C_1 = T_2 - FT_1$ into it.
- We can take either one, for example the second one, because both refer to the same situation from the scheduling point of view, hence they must both give the same result.
- It should be noted that the resulting expression for U_{ub} will still depend on the task periods T_1 and T_2 through F, hence we will have to minimize it with respect to F in order to find U_{lub}.

Eliminating C_1

By substituting $C_1 = T_2 - FT_1$ into:

$$U = F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right)$$

we obtain:

$$\begin{aligned}
U &= F\frac{T_1}{T_2} + \frac{T_2 - FT_1}{T_2}\left(\frac{T_2}{T_1} - F\right) \\
&= F\frac{T_1}{T_2} + \left(1 - F\frac{T_1}{T_2}\right)\left(\frac{T_2}{T_1} - F\right) \\
&= F\frac{T_1}{T_2} + \frac{T_1}{T_2}\left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right) \\
&= \frac{T_1}{T_2}\left[F + \left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right)\right]
\end{aligned}$$

Eliminating C_1

By substituting $C_1 = T_2 - FT_1$ into:

$$U = F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right)$$

we obtain:

$$\begin{aligned}
U &= F\frac{T_1}{T_2} + \frac{T_2 - FT_1}{T_2}\left(\frac{T_2}{T_1} - F\right) \\
&= F\frac{T_1}{T_2} + \left(1 - F\frac{T_1}{T_2}\right)\left(\frac{T_2}{T_1} - F\right) \\
&= F\frac{T_1}{T_2} + \frac{T_1}{T_2}\left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right) \\
&= \frac{T_1}{T_2}\left[F + \left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right)\right]
\end{aligned}$$

Eliminating C_1

By substituting $C_1 = T_2 - FT_1$ into:

$$U = F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right)$$

we obtain:

$$\begin{aligned}
U &= F\frac{T_1}{T_2} + \frac{T_2 - FT_1}{T_2}\left(\frac{T_2}{T_1} - F\right) \\
&= F\frac{T_1}{T_2} + \left(1 - F\frac{T_1}{T_2}\right)\left(\frac{T_2}{T_1} - F\right) \\
&= F\frac{T_1}{T_2} + \frac{T_1}{T_2}\left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right) \\
&= \frac{T_1}{T_2}\left[F + \left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right)\right]
\end{aligned}$$

Eliminating C_1

By substituting $C_1 = T_2 - FT_1$ into:

$$U = F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right)$$

we obtain:

$$\begin{aligned}
U &= F\frac{T_1}{T_2} + \frac{T_2 - FT_1}{T_2}\left(\frac{T_2}{T_1} - F\right) \\
&= F\frac{T_1}{T_2} + \left(1 - F\frac{T_1}{T_2}\right)\left(\frac{T_2}{T_1} - F\right) \\
&= F\frac{T_1}{T_2} + \frac{T_1}{T_2}\left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right) \\
&= \frac{T_1}{T_2}\left[F + \left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right)\right]
\end{aligned}$$

Eliminating C_1

By substituting $C_1 = T_2 - FT_1$ into:

$$U = F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right)$$

we obtain:

$$\begin{aligned} U &= F\frac{T_1}{T_2} + \frac{T_2 - FT_1}{T_2}\left(\frac{T_2}{T_1} - F\right) \\ &= F\frac{T_1}{T_2} + \left(1 - F\frac{T_1}{T_2}\right)\left(\frac{T_2}{T_1} - F\right) \\ &= F\frac{T_1}{T_2} + \frac{T_1}{T_2}\left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right) \\ &= \frac{T_1}{T_2}\left[F + \left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right)\right] \end{aligned}$$

Eliminating C_1

By substituting $C_1 = T_2 - FT_1$ into:

$$U = F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right)$$

we obtain:

$$\begin{aligned}
U &= F\frac{T_1}{T_2} + \frac{T_2 - FT_1}{T_2}\left(\frac{T_2}{T_1} - F\right) \\
&= F\frac{T_1}{T_2} + \left(1 - F\frac{T_1}{T_2}\right)\left(\frac{T_2}{T_1} - F\right) \\
&= F\frac{T_1}{T_2} + \frac{T_1}{T_2}\left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right) \\
&= \frac{T_1}{T_2}\left[F + \left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right)\right]
\end{aligned}$$

Introducing G (I)

We define G as:

$$G = \frac{T_2}{T_1} - F$$

- Since, by definition, $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, $0 \leq G < 1$.
- It will be $G = 0$ when T_2 is an integer multiple of T_1.
- By back substitution, we obtain:

$$\begin{aligned} U &= \frac{T_1}{T_2}(F + G^2) \\ &= \frac{F + G^2}{T_2/T_1} \\ &= \frac{F + G^2}{(T_2/T_1 - F) + F} \\ &= \frac{F + G^2}{F + G} \end{aligned}$$

Introducing G (I)

We define G as:

$$G = \frac{T_2}{T_1} - F$$

- Since, by definition, $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, $0 \leq G < 1$.
- It will be $G = 0$ when T_2 is an integer multiple of T_1.
- By back substitution, we obtain:

$$\begin{aligned} U &= \frac{T_1}{T_2}(F + G^2) \\ &= \frac{F + G^2}{T_2/T_1} \\ &= \frac{F + G^2}{(T_2/T_1 - F) + F} \\ &= \frac{F + G^2}{F + G} \end{aligned}$$

Introducing G (I)

We define G as:

$$G = \frac{T_2}{T_1} - F$$

- Since, by definition, $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, $0 \leq G < 1$.
- It will be $G = 0$ when T_2 is an integer multiple of T_1.
- By back substitution, we obtain:

$$\begin{aligned} U &= \frac{T_1}{T_2}(F + G^2) \\ &= \frac{F + G^2}{T_2/T_1} \\ &= \frac{F + G^2}{(T_2/T_1 - F) + F} \\ &= \frac{F + G^2}{F + G} \end{aligned}$$

Introducing G (I)

We define G as:

$$G = \frac{T_2}{T_1} - F$$

- Since, by definition, $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, $0 \leq G < 1$.
- It will be $G = 0$ when T_2 is an integer multiple of T_1.
- By back substitution, we obtain:

$$\begin{aligned} U &= \frac{T_1}{T_2}(F + G^2) \\ &= \frac{F + G^2}{T_2/T_1} \\ &= \frac{F + G^2}{(T_2/T_1 - F) + F} \\ &= \frac{F + G^2}{F + G} \end{aligned}$$

Introducing G (I)

We define G as:

$$G = \frac{T_2}{T_1} - F$$

- Since, by definition, $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, $0 \leq G < 1$.
- It will be $G = 0$ when T_2 is an integer multiple of T_1.
- By back substitution, we obtain:

$$\begin{aligned} U &= \frac{T_1}{T_2}(F + G^2) \\ &= \frac{F + G^2}{T_2/T_1} \\ &= \frac{F + G^2}{(T_2/T_1 - F) + F} \\ &= \frac{F + G^2}{F + G} \end{aligned}$$

Introducing G (I)

We define G as:

$$G = \frac{T_2}{T_1} - F$$

- Since, by definition, $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, $0 \le G < 1$.
- It will be $G = 0$ when T_2 is an integer multiple of T_1.
- By back substitution, we obtain:

$$\begin{aligned} U &= \frac{T_1}{T_2}(F + G^2) \\ &= \frac{F + G^2}{T_2/T_1} \\ &= \frac{F + G^2}{(T_2/T_1 - F) + F} \\ &= \frac{F + G^2}{F + G} \end{aligned}$$

Introducing G (I)

We define G as:

$$G = \frac{T_2}{T_1} - F$$

- Since, by definition, $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, $0 \leq G < 1$.
- It will be $G = 0$ when T_2 is an integer multiple of T_1.
- By back substitution, we obtain:

$$\begin{aligned} U &= \frac{T_1}{T_2}(F + G^2) \\ &= \frac{F + G^2}{T_2/T_1} \\ &= \frac{F + G^2}{(T_2/T_1 - F) + F} \\ &= \frac{F + G^2}{F + G} \end{aligned}$$

Introducing G (I)

We define G as:

$$G = \frac{T_2}{T_1} - F$$

- Since, by definition, $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, $0 \leq G < 1$.
- It will be $G = 0$ when T_2 is an integer multiple of T_1.
- By back substitution, we obtain:

$$\begin{aligned}
U &= \frac{T_1}{T_2}(F + G^2) \\
&= \frac{F + G^2}{T_2/T_1} \\
&= \frac{F + G^2}{(T_2/T_1 - F) + F} \\
&= \frac{F + G^2}{F + G}
\end{aligned}$$

Introducing G (II)

Then, we have:

$$U = \frac{(F+G) - (G - G^2)}{F+G} = 1 - \frac{G(1-G)}{F+G}$$

- Since $0 \leq G < 1$, then $0 < (1-G) \leq 1$ and $0 \leq G(1-G) \leq 1$.
- As a consequence, U is monotonically nondecreasing with respect to F, and will be minimum when F is minimum, that is, when $F = 1$.
- Therefore, we can substitute $F = 1$ in the previous equation to obtain:

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

Introducing G (II)

Then, we have:

$$U = \frac{(F + G) - (G - G^2)}{F + G} = 1 - \frac{G(1 - G)}{F + G}$$

- Since $0 \leq G < 1$, then $0 < (1 - G) \leq 1$ and $0 \leq G(1 - G) \leq 1$.
- As a consequence, U is monotonically nondecreasing with respect to F, and will be minimum when F is minimum, that is, when $F = 1$.
- Therefore, we can substitute $F = 1$ in the previous equation to obtain:

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

Introducing G (II)

Then, we have:

$$U = \frac{(F+G)-(G-G^2)}{F+G} = 1 - \frac{G(1-G)}{F+G}$$

- Since $0 \leq G < 1$, then $0 < (1-G) \leq 1$ and $0 \leq G(1-G) \leq 1$.
- As a consequence, U is monotonically nondecreasing with respect to F, and will be minimum when F is minimum, that is, when $F = 1$.
- Therefore, we can substitute $F = 1$ in the previous equation to obtain:

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

Introducing G (II)

Then, we have:

$$U = \frac{(F+G) - (G - G^2)}{F+G} = 1 - \frac{G(1-G)}{F+G}$$

- Since $0 \leq G < 1$, then $0 < (1-G) \leq 1$ and $0 \leq G(1-G) \leq 1$.
- As a consequence, U is monotonically nondecreasing with respect to F, and will be minimum when F is minimum, that is, when $F = 1$.
- Therefore, we can substitute $F = 1$ in the previous equation to obtain:

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

Introducing G (II)

Then, we have:

$$U = \frac{(F+G) - (G - G^2)}{F+G} = 1 - \frac{G(1-G)}{F+G}$$

- Since $0 \leq G < 1$, then $0 < (1-G) \leq 1$ and $0 \leq G(1-G) \leq 1$.
- As a consequence, U is monotonically nondecreasing with respect to F, and will be minimum when F is minimum, that is, when $F = 1$.
- Therefore, we can substitute $F = 1$ in the previous equation to obtain:

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

Introducing G (II)

Then, we have:

$$U = \frac{(F+G)-(G-G^2)}{F+G} = 1 - \frac{G(1-G)}{F+G}$$

- Since $0 \leq G < 1$, then $0 < (1-G) \leq 1$ and $0 \leq G(1-G) \leq 1$.
- As a consequence, U is monotonically nondecreasing with respect to F, and will be minimum when F is minimum, that is, when $F = 1$.
- Therefore, we can substitute $F = 1$ in the previous equation to obtain:

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

Introducing G (II)

Then, we have:

$$U = \frac{(F+G) - (G - G^2)}{F+G} = 1 - \frac{G(1-G)}{F+G}$$

- Since $0 \leq G < 1$, then $0 < (1-G) \leq 1$ and $0 \leq G(1-G) \leq 1$.
- As a consequence, U is monotonically nondecreasing with respect to F, and will be minimum when F is minimum, that is, when $F = 1$.
- Therefore, we can substitute $F = 1$ in the previous equation to obtain:

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

Minimizing with Respect to G

- We have:

$$\frac{\mathrm{d}U}{\mathrm{d}G} = \frac{2G(1+G) - (1+G^2)}{(1+G)^2}$$
$$= \frac{G^2 + 2G - 1}{(1+G)^2}$$

- It will be $\frac{\mathrm{d}U}{\mathrm{d}G} = 0$ when $G^2 + 2G - 1 = 0$, that is, when:

$$G_1 = -1 - \sqrt{2} \text{ or,}$$
$$G_2 = -1 + \sqrt{2}$$

- Of these solutions, only G_2 is acceptable, because $G_1 < 0$.

Minimizing with Respect to G

- We have:

$$\frac{\mathrm{d}U}{\mathrm{d}G} = \frac{2G(1+G) - (1+G^2)}{(1+G)^2} = \frac{G^2+2G-1}{(1+G)^2}$$

- It will be $\frac{\mathrm{d}U}{\mathrm{d}G} = 0$ when $G^2 + 2G - 1 = 0$, that is, when:

$$G_1 = -1 - \sqrt{2} \text{ or, } \quad G_2 = -1 + \sqrt{2}$$

- Of these solutions, only G_2 is acceptable, because $G_1 < 0$.

Minimizing with Respect to G

- We have:

$$\frac{\mathrm{d}U}{\mathrm{d}G} = \frac{2G(1+G)-(1+G^2)}{(1+G)^2} = \frac{G^2+2G-1}{(1+G)^2}$$

- It will be $\frac{\mathrm{d}U}{\mathrm{d}G} = 0$ when $G^2 + 2G - 1 = 0$, that is, when:

$$G_1 = -1-\sqrt{2} \text{ or,}$$
$$G_2 = -1+\sqrt{2}$$

- Of these solutions, only G_2 is acceptable, because $G_1 < 0$.

Minimizing with Respect to G

- We have:

$$\frac{\mathrm{d}U}{\mathrm{d}G} = \frac{2G(1+G) - (1+G^2)}{(1+G)^2} = \frac{G^2 + 2G - 1}{(1+G)^2}$$

- It will be $\frac{\mathrm{d}U}{\mathrm{d}G} = 0$ when $G^2 + 2G - 1 = 0$, that is, when:

$$\begin{aligned} G_1 &= -1 - \sqrt{2} \text{ or,} \\ G_2 &= -1 + \sqrt{2} \end{aligned}$$

- Of these solutions, only G_2 is acceptable, because $G_1 < 0$.

The Final Result

The least upper bound of U is given for $G = G_2$:

$$
\begin{aligned}
U_{\text{lub}} &= U|_{G=\sqrt{2}-1} \\
&= \frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} \\
&= \frac{4-2\sqrt{2}}{\sqrt{2}} \\
&= 2(\sqrt{2}-1)
\end{aligned}
$$

For two tasks, $U_{\text{lub}} = 2(\sqrt{2}-1)$.

The Final Result

The least upper bound of U is given for $G = G_2$:

$$
\begin{aligned}
U_{\text{lub}} &= U|_{G=\sqrt{2}-1} \\
&= \frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} \\
&= \frac{4-2\sqrt{2}}{\sqrt{2}} \\
&= 2(\sqrt{2}-1)
\end{aligned}
$$

For two tasks, $U_{\text{lub}} = 2(\sqrt{2}-1)$.

The Final Result

The least upper bound of U is given for $G = G_2$:

$$
\begin{aligned}
U_{\text{lub}} &= U|_{G=\sqrt{2}-1} \\
&= \frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} \\
&= \frac{4-2\sqrt{2}}{\sqrt{2}} \\
&= 2(\sqrt{2}-1)
\end{aligned}
$$

For two tasks, $U_{\text{lub}} = 2(\sqrt{2}-1)$.

The Final Result

The least upper bound of U is given for $G = G_2$:

$$\begin{aligned} U_{\text{lub}} &= U|_{G=\sqrt{2}-1} \\ &= \frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} \\ &= \frac{4-2\sqrt{2}}{\sqrt{2}} \\ &= 2(\sqrt{2}-1) \end{aligned}$$

For two tasks, $U_{\text{lub}} = 2(\sqrt{2}-1)$.

The Final Result

The least upper bound of U is given for $G = G_2$:

$$\begin{aligned} U_{\text{lub}} &= U|_{G=\sqrt{2}-1} \\ &= \frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} \\ &= \frac{4-2\sqrt{2}}{\sqrt{2}} \\ &= 2(\sqrt{2}-1) \end{aligned}$$

For two tasks, $U_{\text{lub}} = 2(\sqrt{2}-1)$.

The Final Result

The least upper bound of U is given for $G = G_2$:

$$\begin{aligned} U_{\text{lub}} &= U|_{G=\sqrt{2}-1} \\ &= \frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} \\ &= \frac{4-2\sqrt{2}}{\sqrt{2}} \\ &= 2(\sqrt{2}-1) \end{aligned}$$

For two tasks, $U_{\text{lub}} = 2(\sqrt{2}-1)$.

Extension to N Tasks

- The result just obtained can be extended to an arbitrary set of N tasks.
- The original proof by Liu and Layland was not completely convincing; it was refined by Devillers and Goossens in 2000.

Theorem (Liu and Layland, 1973)

For a set of N periodic tasks scheduled by the Rate Monotonic algorithm, the least upper bound of the processor utilization factor U_{lub} is:

$$U_{lub} = N\left(2^{1/N} - 1\right)$$

Extension to N Tasks

- The result just obtained can be extended to an arbitrary set of N tasks.
- The original proof by Liu and Layland was not completely convincing; it was refined by Devillers and Goossens in 2000.

Theorem (Liu and Layland, 1973)

For a set of N periodic tasks scheduled by the Rate Monotonic algorithm, the least upper bound of the processor utilization factor U_{lub} is:

$$U_{lub} = N\left(2^{1/N} - 1\right)$$

Extension to N Tasks

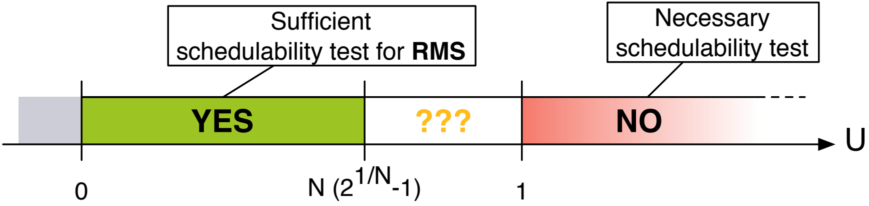
- The result just obtained can be extended to an arbitrary set of N tasks.
- The original proof by Liu and Layland was not completely convincing; it was refined by Devillers and Goossens in 2000.

Theorem (Liu and Layland, 1973)

For a set of N periodic tasks scheduled by the Rate Monotonic algorithm, the least upper bound of the processor utilization factor U_{lub} is:

$$U_{lub} = N\left(2^{1/N} - 1\right)$$

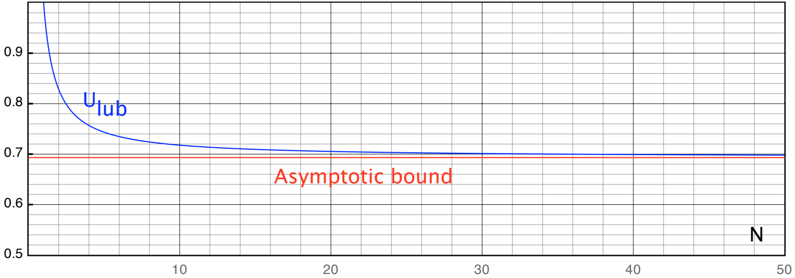
RM Schedulability Summary



- This theorem gives us a sufficient schedulability test for the Rate Monotonic algorithm: a set of N periodic tasks will be schedulable by the Rate Monotonic algorithm if:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq N\left(2^{1/N} - 1\right)$$

One Step Further



- U_{lub} is monotonically decreasing with respect to N. For large values of N, it asymptotically approaches $\ln 2 \approx 0.693$.
- From this observation a simpler — but more pessimistic — sufficient test can be stated: regardless of N, any task set with a combined utilization factor of less than $\ln 2$ will always be schedulable by the Rate Monotonic algorithm.

Examples (I)

We want to check whether the following task set is schedulable by the Rate Monotonic algorithm:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	80	32	Low	
τ_2	40	5	Medium	
τ_3	16	4	High	

- In this and in the following examples, we will assume that both T_i and C_i are measured with the same, arbitrary time unit.
- The combined processor utilization factor is $U = 0.775$.
- For three tasks, we have $U_{\text{lub}} = 3(2^{1/3} - 1) \approx 0.779$.
- Since $U < U_{\text{lub}}$ we conclude, from the sufficient schedulability test, that the task set is schedulable by RM.

Examples (I)

We want to check whether the following task set is schedulable by the Rate Monotonic algorithm:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	80	32	Low	
τ_2	40	5	Medium	
τ_3	16	4	High	

- In this and in the following examples, we will assume that both T_i and C_i are measured with the same, arbitrary time unit.
- The combined processor utilization factor is $U = 0.775$.
- For three tasks, we have $U_{\text{lub}} = 3(2^{1/3} - 1) \approx 0.779$.
- Since $U < U_{\text{lub}}$ we conclude, from the sufficient schedulability test, that the task set is schedulable by RM.

Examples (I)

We want to check whether the following task set is schedulable by the Rate Monotonic algorithm:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	80	32	Low	0.400
τ_2	40	5	Medium	0.125
τ_3	16	4	High	0.250

- In this and in the following examples, we will assume that both T_i and C_i are measured with the same, arbitrary time unit.
- The combined processor utilization factor is $U = 0.775$.
- For three tasks, we have $U_{\text{lub}} = 3(2^{1/3} - 1) \approx 0.779$.
- Since $U < U_{\text{lub}}$ we conclude, from the sufficient schedulability test, that the task set is schedulable by RM.

Examples (I)

We want to check whether the following task set is schedulable by the Rate Monotonic algorithm:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	80	32	Low	0.400
τ_2	40	5	Medium	0.125
τ_3	16	4	High	0.250

- In this and in the following examples, we will assume that both T_i and C_i are measured with the same, arbitrary time unit.
- The combined processor utilization factor is $U = 0.775$.
- For three tasks, we have $U_{\text{lub}} = 3(2^{1/3} - 1) \approx 0.779$.
- Since $U < U_{\text{lub}}$ we conclude, from the sufficient schedulability test, that the task set is schedulable by RM.

Examples (I)

We want to check whether the following task set is schedulable by the Rate Monotonic algorithm:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	80	32	Low	0.400
τ_2	40	5	Medium	0.125
τ_3	16	4	High	0.250

- In this and in the following examples, we will assume that both T_i and C_i are measured with the same, arbitrary time unit.
- The combined processor utilization factor is $U = 0.775$.
- For three tasks, we have $U_{\text{lub}} = 3(2^{1/3} - 1) \approx 0.779$.
- Since $U < U_{\text{lub}}$ we conclude, from the sufficient schedulability test, that the task set is schedulable by RM.

Examples (II)

Let us now consider another set of processes:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	50	12	Low	0.240
τ_2	40	10	Medium	0.250
τ_3	30	10	High	0.334

- The priority assignment does not change, because the T_i are still ordered as before.
- The combined processor utilization factor becomes $U = 0.824$.
- Since $U > U_{\text{lub}}$ the sufficient schedulability test does not tell us anything useful in this case.

Examples (II)

Let us now consider another set of processes:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	50	12	Low	0.240
τ_2	40	10	Medium	0.250
τ_3	30	10	High	0.334

- The priority assignment does not change, because the T_i are still ordered as before.
- The combined processor utilization factor becomes $U = 0.824$.
- Since $U > U_{\text{lub}}$ the sufficient schedulability test does not tell us anything useful in this case.

Examples (II)

Let us now consider another set of processes:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	50	12	Low	0.240
τ_2	40	10	Medium	0.250
τ_3	30	10	High	0.334

- The priority assignment does not change, because the T_i are still ordered as before.
- The combined processor utilization factor becomes $U = 0.824$.
- Since $U > U_{\text{lub}}$ the sufficient schedulability test does not tell us anything useful in this case.

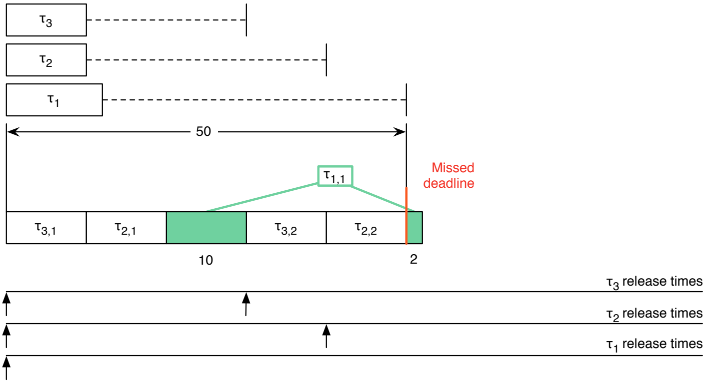
Examples (II)

Let us now consider another set of processes:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	50	12	Low	0.240
τ_2	40	10	Medium	0.250
τ_3	30	10	High	0.334

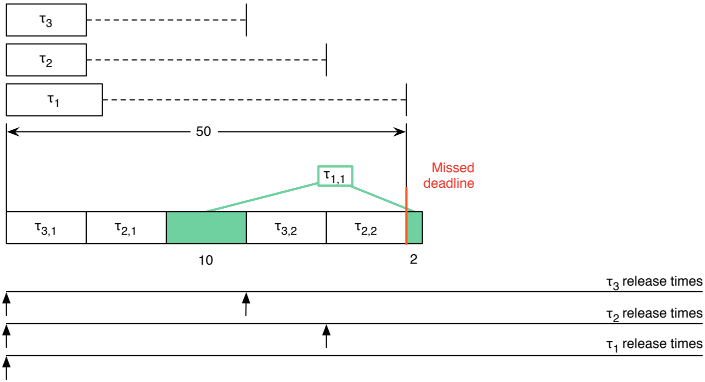
- The priority assignment does not change, because the T_i are still ordered as before.
- The combined processor utilization factor becomes $U = 0.824$.
- Since $U > U_{\text{lub}}$ the sufficient schedulability test does not tell us anything useful in this case.

Examples (III)



- We know that if all tasks fulfill their deadlines when they are released at their critical instant (simultaneously, in this case), then the RM schedule is feasible.
- However, it is easy to show that task τ_1 misses its deadline, hence the task set is not schedulable.

Examples (III)



- We know that if all tasks fulfill their deadlines when they are released at their critical instant (simultaneously, in this case), then the RM schedule is feasible.
- However, it is easy to show that task τ_1 misses its deadline, hence the task set is not schedulable.

Examples (IV)

Let us now consider yet another set of processes:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	80	40	Low	0.500
τ_2	40	10	Medium	0.250
τ_3	20	5	High	0.250

- The priority assignment does not change, because the T_i are still ordered as before.
- The combined processor utilization factor is the maximum value allowed by the necessary schedulability condition, that is, $U = 1$.
- Since $U > U_{\text{lub}}$ the sufficient schedulability test does not tell us anything useful in this case.

Examples (IV)

Let us now consider yet another set of processes:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	80	40	Low	0.500
τ_2	40	10	Medium	0.250
τ_3	20	5	High	0.250

- The priority assignment does not change, because the T_i are still ordered as before.
- The combined processor utilization factor is the maximum value allowed by the necessary schedulability condition, that is, $U = 1$.
- Since $U > U_{\text{lub}}$ the sufficient schedulability test does not tell us anything useful in this case.

Examples (IV)

Let us now consider yet another set of processes:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	80	40	Low	0.500
τ_2	40	10	Medium	0.250
τ_3	20	5	High	0.250

- The priority assignment does not change, because the T_i are still ordered as before.
- The combined processor utilization factor is the maximum value allowed by the necessary schedulability condition, that is, $U = 1$.
- Since $U > U_{\text{lub}}$ the sufficient schedulability test does not tell us anything useful in this case.

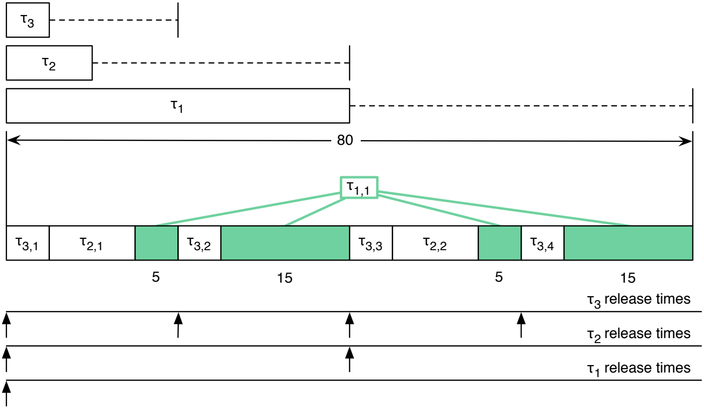
Examples (IV)

Let us now consider yet another set of processes:

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	80	40	Low	0.500
τ_2	40	10	Medium	0.250
τ_3	20	5	High	0.250

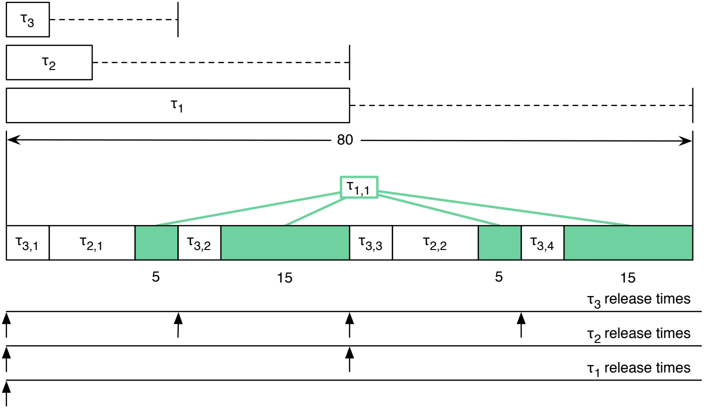
- The priority assignment does not change, because the T_i are still ordered as before.
- The combined processor utilization factor is the maximum value allowed by the necessary schedulability condition, that is, $U = 1$.
- Since $U > U_{\text{lub}}$ the sufficient schedulability test does not tell us anything useful in this case.

Examples (V)



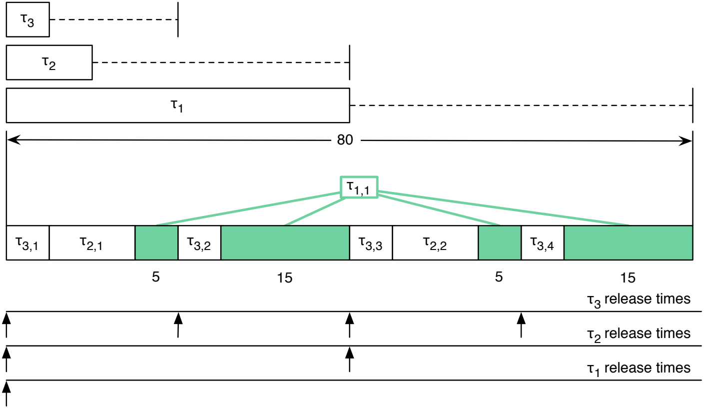
- We can check the actual behavior of the scheduler as before, and discover that all deadlines are met in this case.
- Hence, the task set is indeed schedulable, even if:
  - it does not satisfy the sufficient schedulability test, and
  - the utilization is greater than in the previous example.

Examples (V)



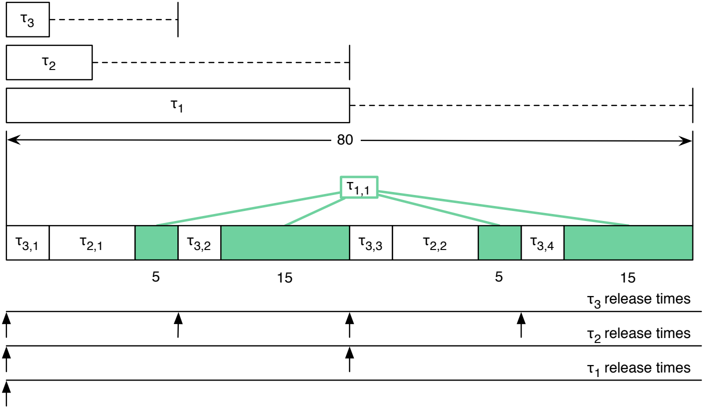
- We can check the actual behavior of the scheduler as before, and discover that all deadlines are met in this case.
- Hence, the task set is indeed schedulable, even if:
  - it does not satisfy the sufficient schedulability test, and
  - the utilization is greater than in the previous example.

Examples (V)



- We can check the actual behavior of the scheduler as before, and discover that all deadlines are met in this case.
- Hence, the task set is indeed schedulable, even if:
  - it does not satisfy the sufficient schedulability test, and
  - the utilization is greater than in the previous example.

Examples (V)



- We can check the actual behavior of the scheduler as before, and discover that all deadlines are met in this case.
- Hence, the task set is indeed schedulable, even if:
  - it does not satisfy the sufficient schedulability test, and
  - the utilization is greater than in the previous example.

Schedulability Test for EDF

Theorem (Liu and Layland, 1973)

A set of N periodic tasks is schedulable with the Earliest Deadline First algorithm if and only if:

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

Proof:

- **Only if:** This result is an immediate consequence of the necessary schedulability condition.
- **If:** We show the sufficiency by *reductio ad absurdum*, that is, we assume that the condition $U \leq 1$ is satisfied and yet the task set is not schedulable. Then, we show that starting from these hypotheses we come to an absurd.

Schedulability Test for EDF

Theorem (Liu and Layland, 1973)

A set of N periodic tasks is schedulable with the Earliest Deadline First algorithm if and only if:

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

Proof:

- Only if: This result is an immediate consequence of the necessary schedulability condition.
- If: We show the sufficiency by *reductio ad absurdum*, that is, we assume that the condition $U \leq 1$ is satisfied and yet the task set is not schedulable. Then, we show that starting from these hypotheses we come to an absurd.

Proof of Sufficiency

- Given that the task set is not schedulable, there will be at least one overflow. Let t_2 be the instant at which the first overflow occurs.
- Now, go backward in time and choose a suitable t_1 so that $[t_1, t_2]$ is the longest interval of continuous utilization before the overflow, so that only task instances $\tau_{i,j}$ with a deadline $d_{i,j} \leq t_2$ are executed within it.
- By definition, t_1 will be the release time of some task instance.

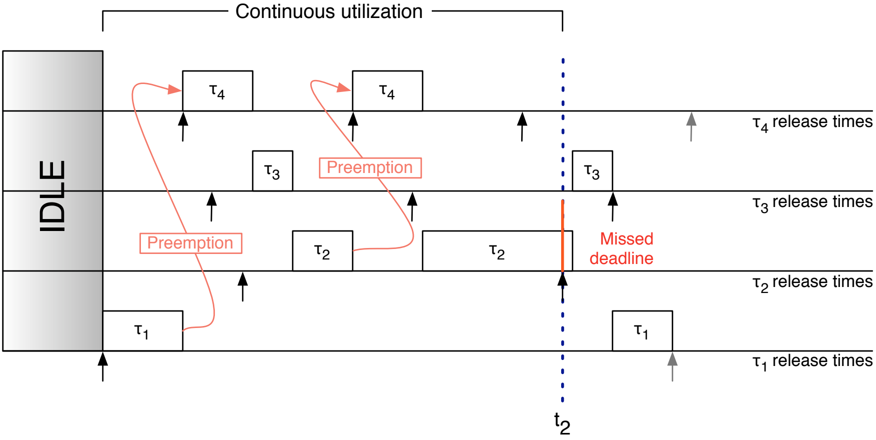
Proof of Sufficiency

- Given that the task set is not schedulable, there will be at least one overflow. Let t_2 be the instant at which the first overflow occurs.
- Now, go backward in time and choose a suitable t_1 so that $[t_1, t_2]$ is the longest interval of continuous utilization before the overflow, so that only task instances $\tau_{i,j}$ with a deadline $d_{i,j} \leq t_2$ are executed within it.
- By definition, t_1 will be the release time of some task instance.

Proof of Sufficiency

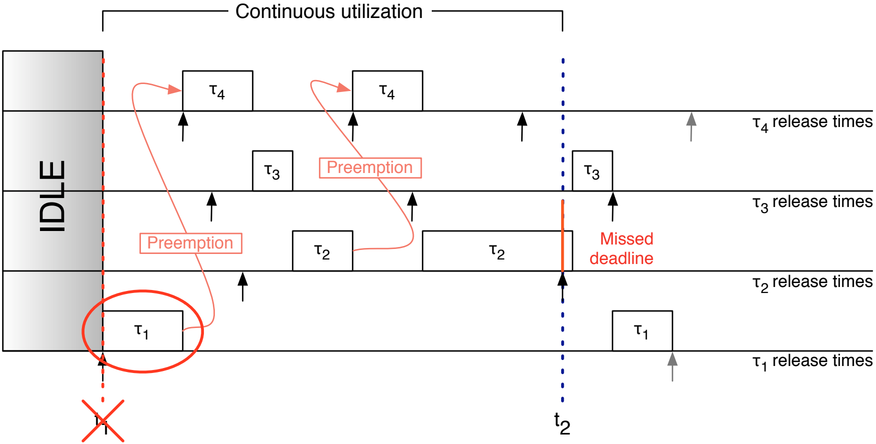
- Given that the task set is not schedulable, there will be at least one overflow. Let t_2 be the instant at which the first overflow occurs.
- Now, go backward in time and choose a suitable t_1 so that $[t_1, t_2]$ is the longest interval of continuous utilization before the overflow, so that only task instances $\tau_{i,j}$ with a deadline $d_{i,j} \leq t_2$ are executed within it.
- By definition, t_1 will be the release time of some task instance.

Determining t_1 and t_2



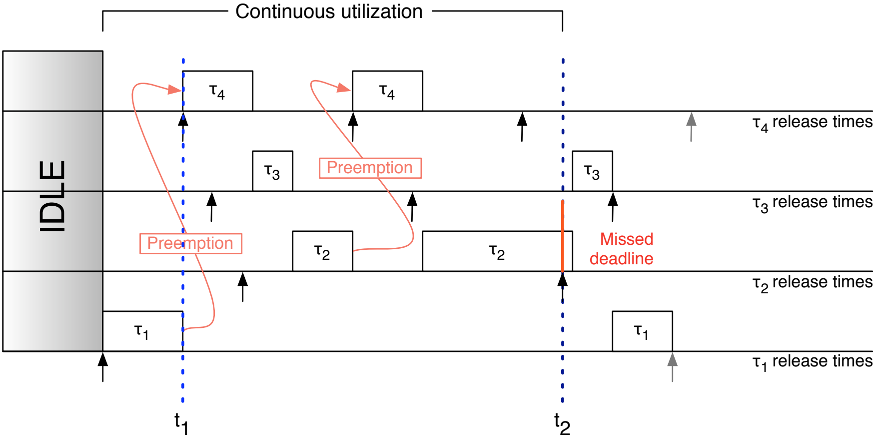
- In this schedule, τ_2 overflows at t_2, that is, it misses its deadline there.

Determining t_1 and t_2



- This is not the "right" t_1, because the highlighted fraction of τ_1 has a deadline which is outside the $[t_1, t_2]$ interval.

Determining t_1 and t_2



- The "right" t_1 is this one, because now all task instances executed within $[t_1, t_2]$ have a deadline which belongs to $[t_1, t_2]$ itself.

Computation Time Demand in $[t_1, t_2]$

- Let $C_p(t_1, t_2)$ the total computation time demand in the time interval $[t_1, t_2]$.
- It can be computed as:

$$C_p(t_1, t_2) = \sum_{i \mid r_{i,j} \geq t_1 \wedge d_{i,j} \leq t_2} C_i$$

Question...

How many instances of each τ_i must be considered in the above formula?

Computation Time Demand in $[t_1, t_2]$

- Let $C_p(t_1, t_2)$ the total computation time demand in the time interval $[t_1, t_2]$.
- It can be computed as:

$$C_p(t_1, t_2) = \sum_{i \mid r_{i,j} \geq t_1 \wedge d_{i,j} \leq t_2} C_i$$

Question...

How many instances of each τ_i must be considered in the above formula?

Computation Time Demand in $[t_1, t_2]$

- Let $C_p(t_1, t_2)$ the total computation time demand in the time interval $[t_1, t_2]$.
- It can be computed as:

$$C_p(t_1, t_2) = \sum_{i \mid r_{i,j} \geq t_1 \wedge d_{i,j} \leq t_2} C_i$$

Question...

How many instances of each τ_i must be considered in the above formula?

Computation Time Demand in $[t_1, t_2]$

. . . and answer

The maximum number of instances of τ_i to be considered is equal to the the number of periods of τ_i entirely contained within the time interval $[t_1, t_2]$, that is:

$$\left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor$$

- If we added another instance, either the first instance would have a release time before t_1, or the last one would have a deadline after t_2.
- For N tasks, we can define $C_p(t_1, t_2)$ more explicitly as:

$$C_p(t_1, t_2) = \sum_{i=1}^{N} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i$$

Computation Time Demand in $[t_1, t_2]$

... and answer

The maximum number of instances of τ_i to be considered is equal to the the number of periods of τ_i entirely contained within the time interval $[t_1, t_2]$, that is:

$$\left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor$$

- If we added another instance, either the first instance would have a release time before t_1, or the last one would have a deadline after t_2.
- For N tasks, we can define $C_p(t_1, t_2)$ more explicitly as:

$$C_p(t_1, t_2) = \sum_{i=1}^{N} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i$$

Computation Time Demand in $[t_1, t_2]$

... and answer

The maximum number of instances of τ_i to be considered is equal to the the number of periods of τ_i entirely contained within the time interval $[t_1, t_2]$, that is:

$$\left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor$$

- If we added another instance, either the first instance would have a release time before t_1, or the last one would have a deadline after t_2.
- For N tasks, we can define $C_p(t_1, t_2)$ more explicitly as:

$$C_p(t_1, t_2) = \sum_{i=1}^{N} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i$$

Coming to an Absurd (I)

- Observe that:

$$
\begin{aligned}
C_p(t_1, t_2) &= \sum_{i=1}^{N} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \\
&\leq \sum_{i=1}^{N} \frac{t_2 - t_1}{T_i} C_i \quad \text{(by definition of } \lfloor \cdot \rfloor\text{)} \\
&= (t_2 - t_1) \sum_{i=1}^{N} \frac{C_i}{T_i} \\
&= (t_2 - t_1) U \quad \text{(by definition of } U\text{)}
\end{aligned}
$$

In summary, we have:

$$C_p(t_1, t_2) \leq (t_2 - t_1) U$$

Coming to an Absurd (I)

- Observe that:

$$
\begin{aligned}
C_p(t_1, t_2) &= \sum_{i=1}^{N} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \\
&\leq \sum_{i=1}^{N} \frac{t_2 - t_1}{T_i} C_i \quad \text{(by definition of } \lfloor \cdot \rfloor\text{)} \\
&= (t_2 - t_1) \sum_{i=1}^{N} \frac{C_i}{T_i} \\
&= (t_2 - t_1) U \quad \text{(by definition of } U\text{)}
\end{aligned}
$$

In summary, we have:

$$C_p(t_1, t_2) \leq (t_2 - t_1) U$$

Coming to an Absurd (I)

- Observe that:

$$
\begin{aligned}
C_p(t_1, t_2) &= \sum_{i=1}^{N} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \\
&\leq \sum_{i=1}^{N} \frac{t_2 - t_1}{T_i} C_i \quad \text{(by definition of } \lfloor \cdot \rfloor\text{)} \\
&= (t_2 - t_1) \sum_{i=1}^{N} \frac{C_i}{T_i} \\
&= (t_2 - t_1) U \quad \text{(by definition of } U\text{)}
\end{aligned}
$$

In summary, we have:

$$C_p(t_1, t_2) \leq (t_2 - t_1) U$$

Coming to an Absurd (I)

- Observe that:

$$
\begin{aligned}
C_p(t_1, t_2) &= \sum_{i=1}^{N} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \\
&\leq \sum_{i=1}^{N} \frac{t_2 - t_1}{T_i} C_i \quad \text{(by definition of } \lfloor \cdot \rfloor\text{)} \\
&= (t_2 - t_1) \sum_{i=1}^{N} \frac{C_i}{T_i} \\
&= (t_2 - t_1) U \quad \text{(by definition of } U\text{)}
\end{aligned}
$$

In summary, we have:

$$C_p(t_1, t_2) \leq (t_2 - t_1) U$$

Coming to an Absurd (I)

- Observe that:

$$
\begin{aligned}
C_p(t_1, t_2) &= \sum_{i=1}^{N} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \\
&\leq \sum_{i=1}^{N} \frac{t_2 - t_1}{T_i} C_i \quad \text{(by definition of } \lfloor \cdot \rfloor \text{)} \\
&= (t_2 - t_1) \sum_{i=1}^{N} \frac{C_i}{T_i} \\
&= (t_2 - t_1) U \quad \text{(by definition of } U \text{)}
\end{aligned}
$$

In summary, we have:

$$C_p(t_1, t_2) \leq (t_2 - t_1) U$$

Coming to an Absurd (II)

On the other hand…

…since there is an overflow at t_2, then $C_p(t_1, t_2)$ (the total computation time demand) must exceed $t_2 - t_1$ (the time interval in which that demand takes place), that is:

$$C_p(t_1, t_2) > t_2 - t_1$$

- By combining the two inequations just derived, we obtain:

$$(t_2 - t_1)U \geq C_p(t_1, t_2) > t_2 - t_1$$

- That is, dividing both sides by $t_2 - t_1$:

$$U > 1$$

Coming to an Absurd (II)

On the other hand...

...since there is an overflow at t_2, then $C_p(t_1, t_2)$ (the total computation time demand) must exceed $t_2 - t_1$ (the time interval in which that demand takes place), that is:

$$C_p(t_1, t_2) > t_2 - t_1$$

- By combining the two inequations just derived, we obtain:

$$(t_2 - t_1)U \geq C_p(t_1, t_2) > t_2 - t_1$$

- That is, dividing both sides by $t_2 - t_1$:

$$U > 1$$

Coming to an Absurd (II)

On the other hand…

…since there is an overflow at t_2, then $C_p(t_1, t_2)$ (the total computation time demand) must exceed $t_2 - t_1$ (the time interval in which that demand takes place), that is:

$$C_p(t_1, t_2) > t_2 - t_1$$

- By combining the two inequations just derived, we obtain:

$$(t_2 - t_1)U \geq C_p(t_1, t_2) > t_2 - t_1$$

- That is, dividing both sides by $t_2 - t_1$:

$$U > 1$$

Coming to an Absurd (III)

- This is absurd, because the conclusion contradicts one of the hypotheses, namely, $U \leq 1$.
- The contradiction comes from having supposed the task set being not schedulable.
- Hence, we can conclude that the condition

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

is both necessary and sufficient to guarantee the schedulability of a task set with the EDF algorithm.

□

Coming to an Absurd (III)

- This is absurd, because the conclusion contradicts one of the hypotheses, namely, $U \leq 1$.
- The contradiction comes from having supposed the task set being not schedulable.
- Hence, we can conclude that the condition

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

is both necessary and sufficient to guarantee the schedulability of a task set with the EDF algorithm.

□

Coming to an Absurd (III)

- This is absurd, because the conclusion contradicts one of the hypotheses, namely, $U \leq 1$.
- The contradiction comes from having supposed the task set being not schedulable.
- Hence, we can conclude that the condition

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

is both necessary and sufficient to guarantee the schedulability of a task set with the EDF algorithm.

□

Optimality of EDF

Corollary

The EDF algorithm is optimum in the sense that if any task set is schedulable by any scheduling algorithm, under the hypotheses just set out, then it is also schedulable by EDF.

The corollary is easy to prove by observing that:

- If a task set Γ is schedulable by an arbitrary algorithm A, then it must satisfy the necessary schedulability condition, that is, it must be $U \leq 1$.
- Since Γ has $U \leq 1$, then it is schedulable with the EDF algorithm, because it satisfies the sufficient schedulability test just proved.

□

Optimality of EDF

Corollary

The EDF algorithm is optimum in the sense that if any task set is schedulable by any scheduling algorithm, under the hypotheses just set out, then it is also schedulable by EDF.

The corollary is easy to prove by observing that:

- If a task set Γ is schedulable by an arbitrary algorithm A, then it must satisfy the necessary schedulability condition, that is, it must be $U \leq 1$.
- Since Γ has $U \leq 1$, then it is schedulable with the EDF algorithm, because it satisfies the sufficient schedulability test just proved.

□

Optimality of EDF

Corollary

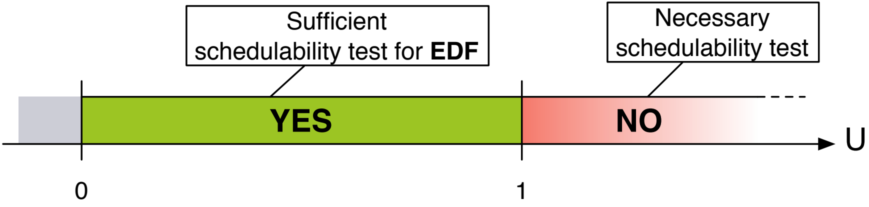
The EDF algorithm is optimum in the sense that if any task set is schedulable by any scheduling algorithm, under the hypotheses just set out, then it is also schedulable by EDF.

The corollary is easy to prove by observing that:

- If a task set Γ is schedulable by an arbitrary algorithm A, then it must satisfy the necessary schedulability condition, that is, it must be $U \leq 1$.
- Since Γ has $U \leq 1$, then it is schedulable with the EDF algorithm, because it satisfies the sufficient schedulability test just proved.

$\square$

EDF Schedulability Summary



- With respect to the sufficient schedulability test for the Rate Monotonic algorithm, the corresponding test for the Earliest Deadline First algorithm is conceptually simpler, since it hasn't any "grey area" of uncertainty.