05_Scheduling of Periodic Tasks

05_Scheduling of Periodic Tasks

1. Basic Concepts

- 1.1. Task Model
- 1.2. Target
- 1.3. Hyperperiod

2. Schedulability Tests

- 2.1. Introduction
  - Necessary Test:
  - Sufficient Test:
  - Exact
- 2.2. Utilization
  - Definition
  - Important Bounds
  - Judgement
- 2.3. Critical Instant

3. Some intuitive yet not efficient Algorithms

- 3.1. Proportional Share Algorithm
- 3.2. Work-and-Sleep Algorithm

4. Timeline Scheduling (Smart Round Robin)

  - Main Idea
  - Property:
  - Important Issues
  - Implementation

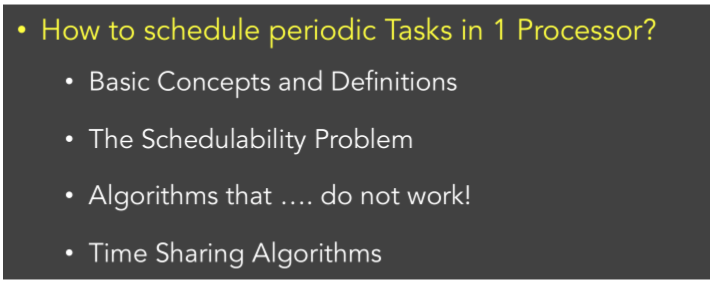
5. Priority-based Scheduling

- 5.1. Basic Idea
- 5.2. Rate Monotonic
- 5.3. Deadline Monotonic
- 5.4. Earliest Deadline First

6. Guarantee Tests

- 6.1. LUB
- 6.2. Guarantee Tests for RM
  - 6.2.1. Test 1 (sufficient)
  - 6.2.2. Proof of Test 1
  - 6.2.3. RM Test with Hyperbolic Bound
  - 6.2.4. Properties
  - 6.2.5. RM for Harmonic Periods
- 6.3. Guarantee Check of DM

A **too pessimistic** method
Response Time Analysis (RTA)
6.4. Guarantee Tests for EDF
An exact test (for $D_i = T_i$)
Test for $D_i < T_i$
7. Summary
7.1. Summary For Tests
7.2. RM VS EDF
When Permanent Overload Happens



1. Basic Concepts

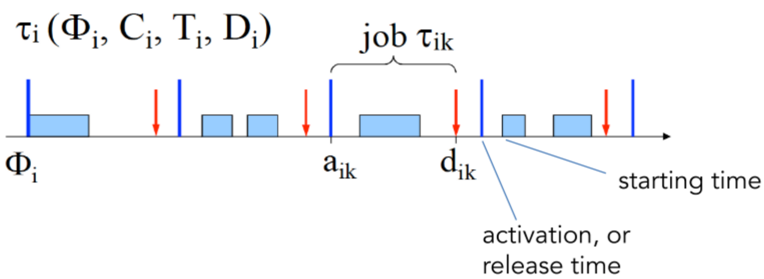
1.1. Task Model

Consider a computing system that needs to execute a set Γ of n periodic real-time tasks:

$$\Gamma = \{\tau_1, \tau_2, \dots \tau_n\}$$

Each task τ_i is characterized by:

$$\begin{gathered} \mathrm{C_i} : \text{worst-case computation time} \\ \mathrm{T_i} : \text{ activation period} \\ \mathrm{D_i} : \text{relative deadline} \\ \Phi_i : \text{initial arrival time (phase)} \end{gathered}$$



1.2. Target

We want to ensure that for each task i:

- Each job k will be activated at:

$$a_{ik} = \Phi_i + (k-1)T_i$$

- Each job k will be completed before its deadline

$$\mathrm{d_{ik} = a_{ik} + D_i}$$

1.3. Hyperperiod

- The minimum time interval after which the schedule repeats
- If the tasks are activated at t=0, then it is given by the least common multiple (lcm) of their periods

$$H = \operatorname{lcm}(T_1, T_2, \ldots, T_n)$$

2. Schedulability Tests

2.1. Introduction

A task set Γ is **feasible** if each task i=1,...,n in Γ can be executed for C_i time units in every interval $[a_{ik}, d_{ik}]$, k=1, 2,....

Necessary Test:

If the task set **does not pass** the test, then it is **certainly not schedulable** by this algorithm.

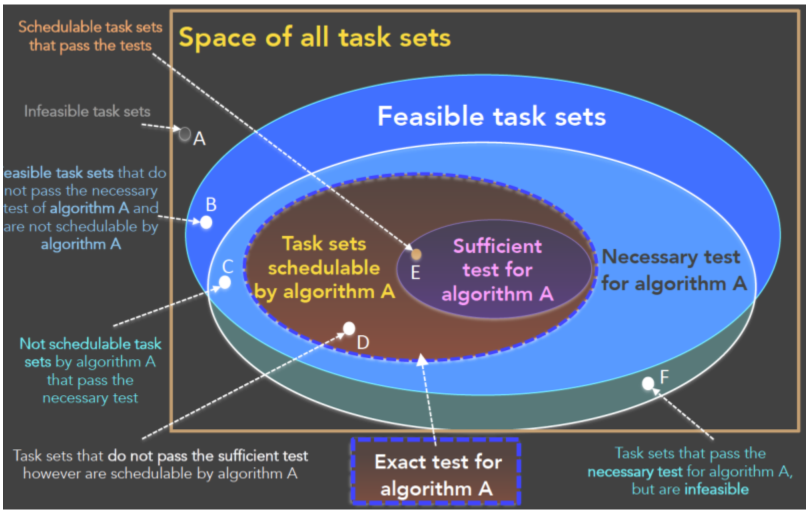
Sufficient Test：

If the task set **passes the test**, then it is **certainly schedulable** by this algorithm.

Exact

Both Necessary and Sufficient

- If the task set passes the test, then it is certainly schedulable by this algorithm.
- If the task set does not pass the test, then it is certainly not schedulable



2.2. Utilization

Definition

The **task utilization** factor U_i is the fraction of processor time spent in the execution of task i:

$$U_i = \frac{C_i}{T_i}$$

The **processor utilization** factor U is the fraction of processor time spent in the execution of the given task set:

$$U = \sum_i U_i$$

Important Bounds

Utilization depends on task set Γ and the algorithm A:

$$U_{ub}(\Gamma, A)$$

Upper bound of processor utilization for task set Γ under a given algorithm A

- if we increase further the computation time of any task, it becomes infeasible

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

For a given algorithm A, it is the **minimum of the utilization factors** over **all** task sets that fully utilize the processor

- any task set whose utilization is less than or equal to this bound, is schedulable by A

Judgement

- U > 1: no algorithm can schedule the task set: (H for hyperperiod)

$$\mathrm{U} > 1 \Rightarrow \mathrm{HU} > \mathrm{H} \Rightarrow \sum_{i=1}^{n} \frac{C_i}{T_i} H = \sum_{i=1}^{n} \frac{H}{T_i} C_i = \sum_{i=1}^{n} m_i C_i > H$$

- $\mathrm{U}(\Gamma, \mathrm{A}) \leq \mathrm{U}_{\mathrm{lub}}(\mathrm{A})$: set Γ can be scheduled with Algorithm A
- $\mathrm{U}_{\mathrm{lub}}(\mathrm{A}) < \mathrm{U}(\Gamma, \mathrm{A}) \leq 1$: cannot really tell! Depends on the relation of the task periods, computation times, etc

2.3. Critical Instant

- **Critical instant** of a task = arrival time inducing the **largest response time** R.
- This occurs when the task arrives concurrently with all higher priority tasks

3. Some intuitive yet not efficient Algorithms

3.1. Proportional Share Algorithm

Rules:

- Divide the time into slots of length: $\mathit{\Delta} = G.C.D.(T_1, T_2, \dots T_n)$
- In each slot serve each task for time proportional to its utilization

$$\delta_i = U_i \cdot \Delta$$

Property:

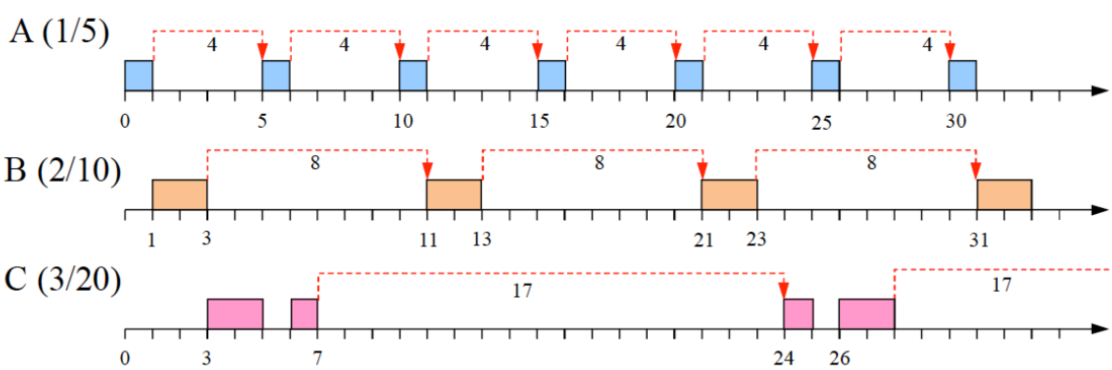
- If $\sum_i U_i \leq 1$, then it will be feasible
- But if Δ is very small**, overhead can be very high** (too many context switch)
- If switching indueces delay, it may infeasible

3.2. Work-and-Sleep Algorithm

Rules:

- A task is executed for C_i units and suspends for $T_i – C_i$ units
- **Preemption** is used when a higher-priority task wakes

Task	C_i	T_i	Sleep time
A	1	5	4
B	2	10	8
C	3	20	17



Property:

- easy to implement
- **starves** the low priority tasks.

4. Timeline Scheduling (Smart Round Robin)

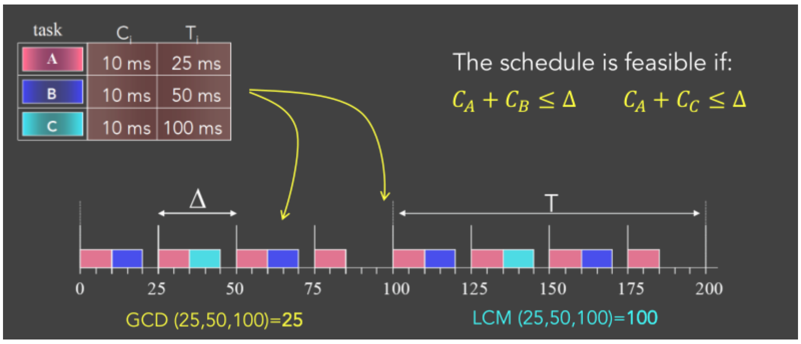
Main Idea

- Time is divided into slots of equal length. for the task set Γ, we define:
    - the small cycle (Δ) : $\Delta = GCD(Periods)$
    - the large cycle (T): $T = LCM(Periods)$
- Each task is statically allocated to a slot, the algorithm does not care about how to map the tasks to slots

task	C_i	T_i
A	10 ms	25 ms
B	10 ms	50 ms
C	10 ms	100 ms

GCD (25,50,100)=**25**

LCM (25,50,100)=**100**



The schedule is feasible if: $C_A + C_B \le \Delta \quad C_A + C_C \le \Delta$

Property:

- Advantages:
    - Simple to implement
    - consistently low jitter
- Disadvantages:

- Difficulties handling overloading
- Sensitivity to application changes

Important Issues

1. If any task does not finish on time, then we
    - either terminate it, endangering inconsistent system state
    - wait for it to finish, endangering a **domino effect**
2. If the compute time of a task changes, we need to reschedule
3. If the frequency of a task changes, the impact is even worse

Task	T_{old}	T_{new}
A	25 ms	25 ms
B	50 ms	**40 ms**
C	100 ms	100 ms

minor cycle: $\Delta = 25$ $\Delta = 5$

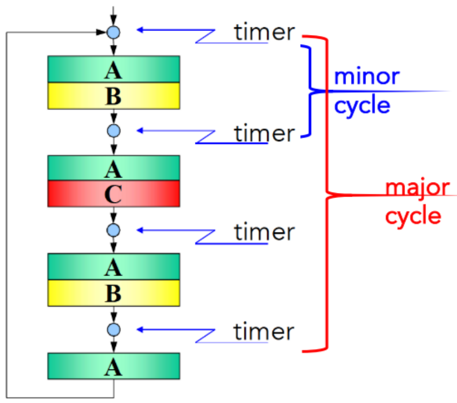
major cycle: $T = 100$ $T = 200$

(40 sync. per cycle!)

Implementation

- program a timer to interrupt with period equal to minor cycle
- call the tasks in the order given in the major cycle by inserting a synch point at the start of each minor cycle

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



5. Priority-based Scheduling

5.1. Basic Idea

1. **Assign priorities** to tasks based on their timing constraints
2. **Verify** the schedule feasibility using analytical techniques.
3. **Execute** tasks on a priority-based kernel.

5.2. Rate Monotonic

Assumption

- C_i and T_i **are constant** for every task i.
- **The relative deadline is equal to task period**: $D_i = T_i$
- Tasks **are preemptable**;
- Context switching and preemption induce zero overheads;
- Tasks are independent:
    - No precedence relations, no resource constraints or blocking on I/O.

Rule:

Each task is assigned a fixed priority proportional to its rate (=inverse of period)

Properties

RM is **optimal** among **all fixed priority** algorithms (w.r.t. feasibility):

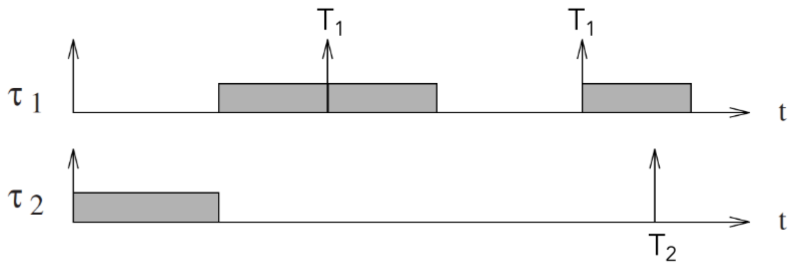
- If there is a fixed priority assignment which leads to a feasible schedule, then the RM schedule is also feasible.
- If a task set is not schedulable by RM, then it cannot be scheduled by any fixed priority assignment

RM Optimality

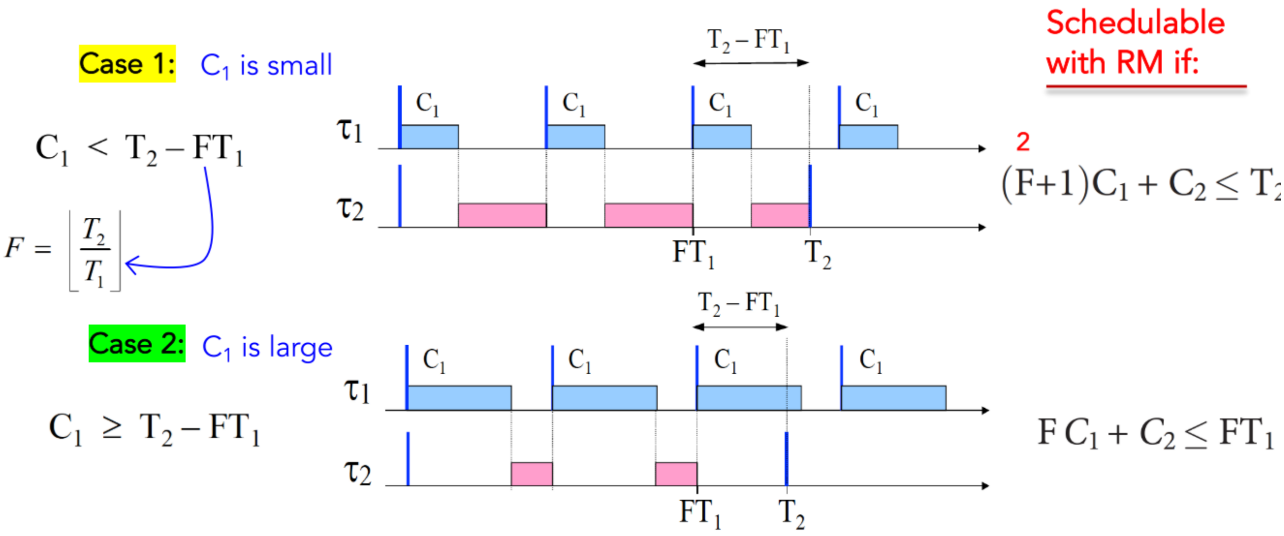
We try to prove: If a set is schedulable by some priority assignment, then it is also schedulable by RM

Consider this example where priorities are: Task 1 > Task 2

A non-RM algorithm schedules first Task 2



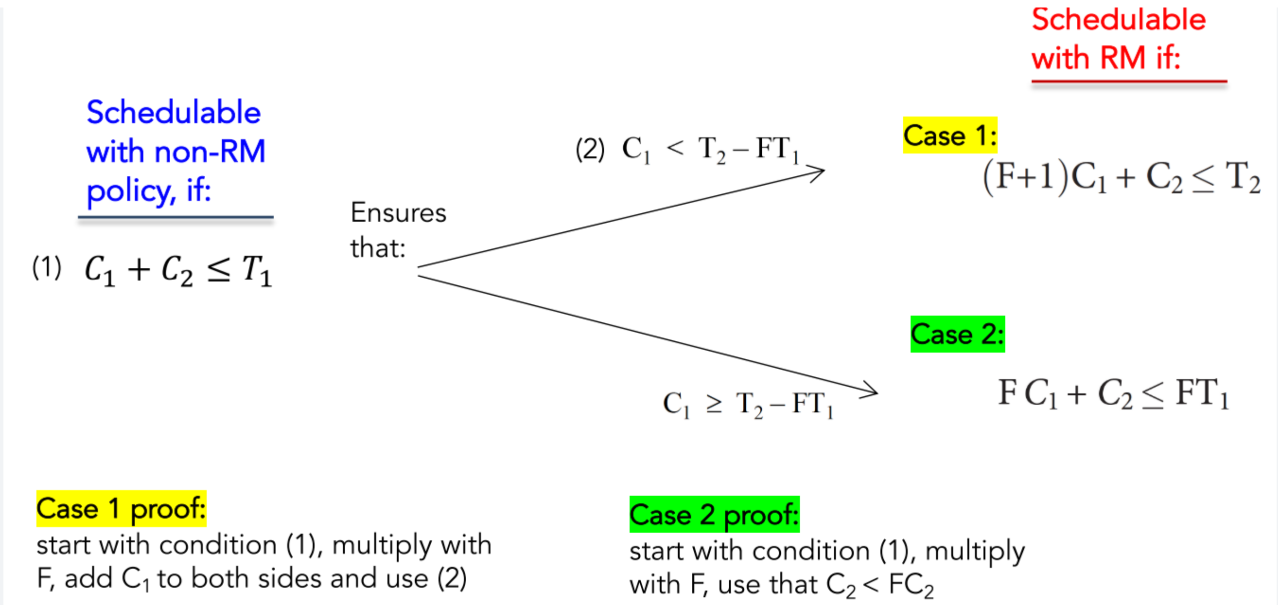
- In this scenario, the schedule is feasible if and only $C_1 + C_2 \leq T_1$
- We will show that this condition is sufficient for the feasibility with RM



$$\mathrm{FC_1 + C_2 \leq max\{FT_1, T_2 - C_1\}}$$

One point to be explained here: in general, it should be $FC_1 + C_2 < \min(.., ..)$, However, it can be easily proved that (by the relations between C_1 and $T_2 - FT_1$):

- when case 1: $T_2 - C_1 > FT_1$,
- when case 2: $T_2 - C_1 < FT_1$



- Case 1 means in the less in T1 part, can complete 1 more C1
- Case 2 means in the less in T1 part, can not complete 1 more C1
- Then we can from the graph found right two inequation, then we need to prove it

5.3. Deadline Monotonic

Assumption

- Extension to Tasks with D<T
- We drop the assumption that D = T

Rule

Similar with Rate Monotonic, but for relative deadlines smaller than tasks period

- At any time-instant, execute the task with **the shortest relative deadline**.

It is a **preemptive** algorithm.

Property

- **Static** method
- DM is **optimal (wrt feasibility)** among all **fixed priority algorithms**.

5.4. Earliest Deadline First

Rule

- Each new job k of each task i, gets priority inversely proportional to its absolute deadline (so it is dynamic)

$$P_{ik} = \frac{1}{d_{ik}} \quad d_{ik} = r_{ik} + D_i$$

- A **preemptive** policy, where at each time we execute the task with the smaller absolute deadline (sooner-to-expire)
- Works equally well for aperiodic and for periodic tasks.
- EDF assigns priorities to each job as it is generated.

Optimality of EDF

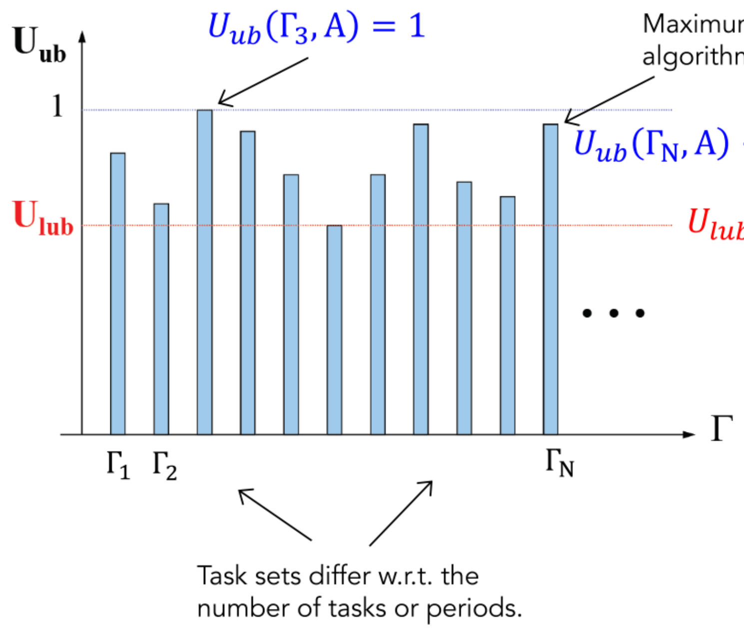
- EDF is **optimal among all** scheduling algorithms.

- If there exists a feasible schedule for a task set Γ, then EDF will generate a feasible schedule.
- If Γ is not schedulable by EDF, then it cannot be scheduled by any algorithm.

6. Guarantee Tests

6.1. LUB

- Different task set yields a different upper bound: if we increase any C, we will have an infeasible schedule



- Any task set with U less than U_{lub} is schedulable;
- Any task set with U>1 is not schedulable;
- If $U_{lub} < U \leq 1$, we need to perform schedulability analysis

U_{lub} is a value on **"all" Test sets**. But we cannot enumerate all test sets, we will introduce some methods to calculate U_{lub} with some analyze ways.

6.2. Guarantee Tests for RM

6.2.1. Test 1 (sufficient)

$$U_{lub}^{RM} = n\left(2^{\frac{1}{n}} - 1\right), \quad n \to \infty \Rightarrow U_{lub}^{RM} \to \ln 2 = 0.69$$

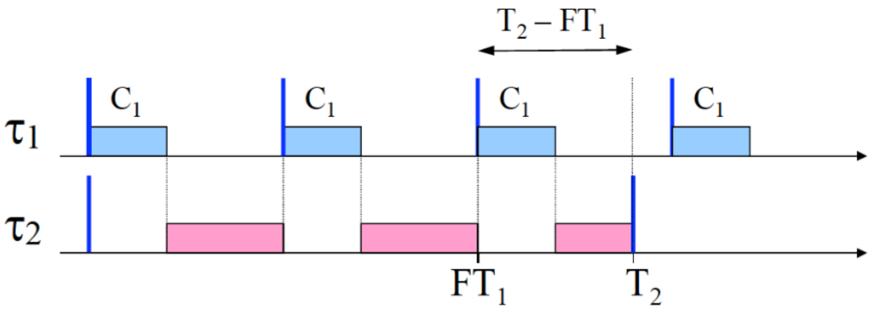
The task set is schedulable if

$$U_p = \sum_{n=1}^{n} \frac{C_i}{T_i} \leq U_{lub}^{RM} = n\left(2^{\frac{1}{n}} - 1\right)$$

6.2.2. Proof of Test 1

- Assume the **worst-case scenario** for the task set; simultaneous arrivals, critical instants of tasks;
- **Increase all C values** to fully utilize the processor;
- Compute the upper bound U_{ub};
- Minimize U_{ub} with respect to all remaining variables (yields Least U_{ub}).

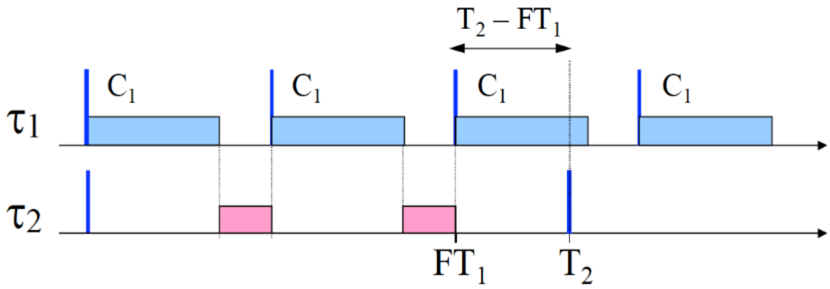
- Case 1: $C_1 < T_2 - FT_1$ where $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$



$$C_2^{\max} = T_2 - (F+1)C_1$$

$$U_{ub} = \frac{C_1}{T_1} + \frac{T_2 - (F+1)C_1}{T_2} = 1 + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right]$$

- Case 2: $C_1 \geq T_2 - FT_1$ where $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$

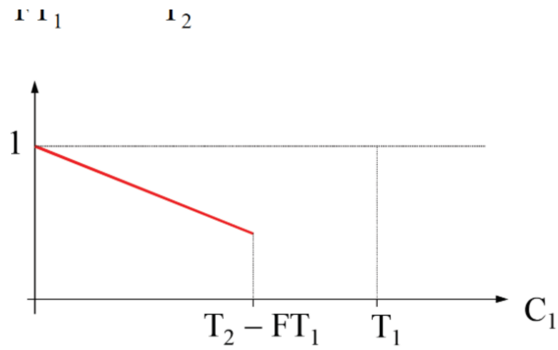


$$C_2^{\max} = F(T_1 - C_1)$$

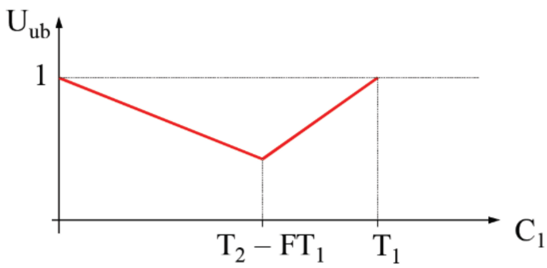
$$U_{ub} = \frac{C_1}{T_1} + \frac{F(T_1 - C_1)}{T_2} = F\frac{T_1}{T_2} + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right)$$

This can be seen as a function of C_1

$$U_{ub} = 1 + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right]$$



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



- In both cases the min bound is obtained when:

$$C_1 = T_2 - FT_1 \qquad \text{and} \qquad C_2^{\max} = T_2 - (F+1)C_1$$

- And we get:

$$U_{\text{lub}} = U_{ub}\big|_{C_1 = T_2 - FT_1} = \frac{T_1}{T_2}\left[F + \left(\frac{T_2}{T_1} - F\right)^2\right]$$

- What else can we optimize to find the LB?
  - Increases with F; hence we set the minimum possible (F=1)
  - And we can further minimize wrt $k=T_2/T_1$

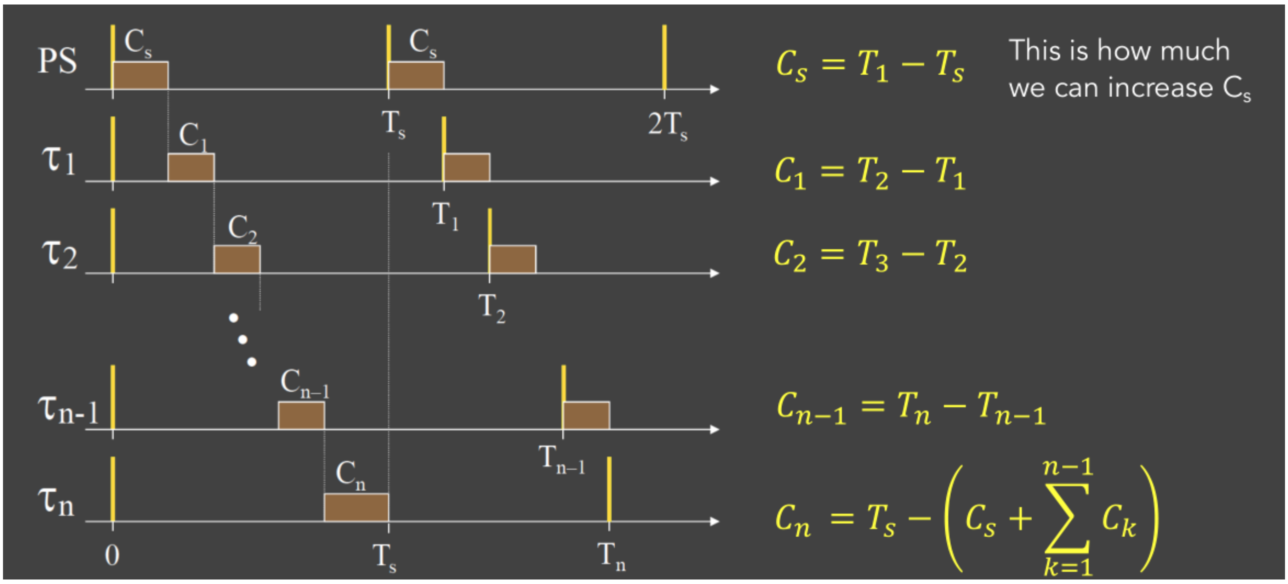
$$U_{\text{lub}} = \frac{1 + (k-1)^2}{k} \qquad \frac{dU_{\text{lub}}}{dk} = \frac{k^2 - 2}{k^2}$$

$dU/dk = 0$ for $k = \sqrt{2}$

$$U_{\text{lub}} = 2\left(\sqrt{2} - 1\right) \cong 0.83$$

- In the first line, the second equation meet the "max utility", the first equation meet "min bound"

Above case is an special case for:



6.2.3. RM Test with Hyperbolic Bound

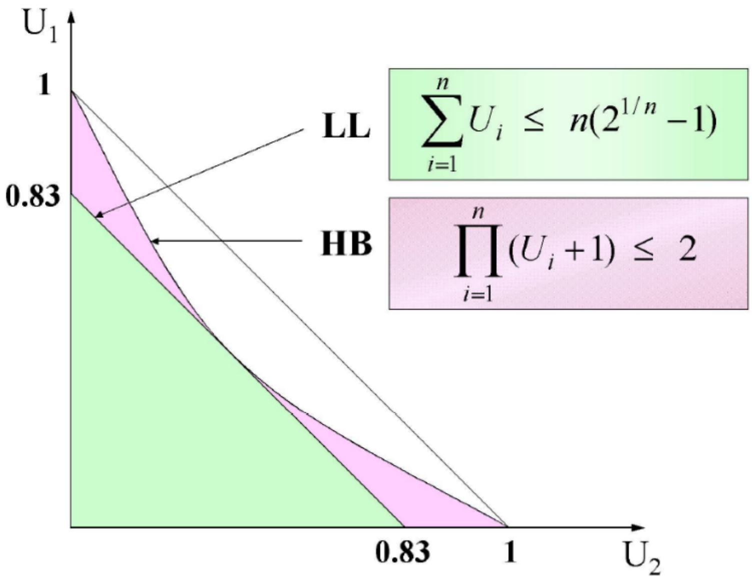
If Γ is a set of n periodic tasks, where each task τ_i, induces processor utilization U_i, Γ is schedulable with RM if:

$$\Pi_{i=1}^{n} \left(U_i + 1\right) \leq 2$$

6.2.4. Properties

All these two methods are pessimistic, that means, they are **sufficient test**. If some test set not pass, it does not mean it is not schedullable.

Comparison of the two tests using the *utilization space*



Additional Example #1

τ_i	C_i	T_i	D_i	U_i
τ_1	8	10	10	0.8
τ_2	0.9	18	18	0.05

- Is the task set feasible?
  - Yes, because $U = 0.85 \leq 1$
- Does the task set pass the LL test?
  - No, because $U = 0.85 > 2(2^{1/2} - 1) \approx 0.83$
- Does it pass the HB test?
  - Yes, because $(0.8 + 1)(0.05 + 1) = 1.89 < 2$
- Is the task set schedulable by RM?
  - Yes, because it passes the HB test (sufficient test).

Necessary Test	Sufficient LL Test	Sufficient HB Test
$\sum_{i=1}^{n} U_i \leq 1$	$\sum_{i=1}^{n} U_i \leq n(2^{\frac{1}{n}} - 1)$	$\prod_{i=1}^{n} (U_i + 1) \leq 2$

6.2.5. RM for Harmonic Periods

RM is optimal if the task periods are harmonic (i.e., each period divides exactly the larger ones).

e.g. Harmonic set: $\{T_1 = 4, T_2 = 8, T_3 = 16\}$

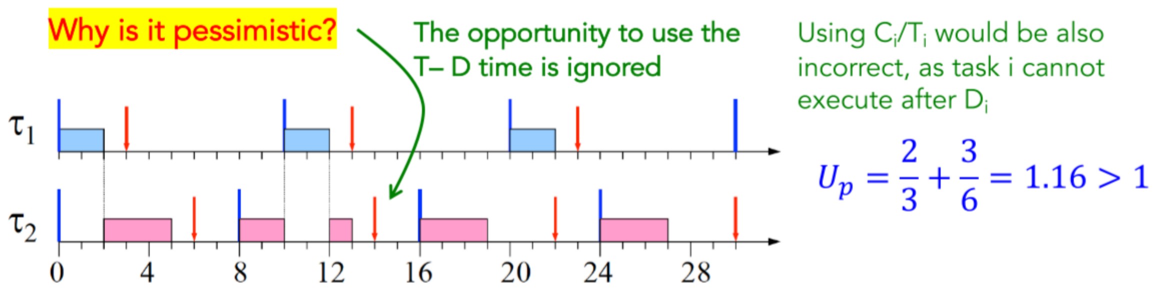
In this case the condition $U \leq 1$ is an exact test

6.3. Guarantee Check of DM

A **too pessimistic** method

$$U_p = \sum_{n=1}^{n} \frac{C_i}{D_i} \leq U_{lub}^{RM} = n\left(2^{\frac{1}{n}} - 1\right)$$

- The opportunity to use the T− D time is ignored



Response Time Analysis (RTA)

Main Idea

Focus on the critical instances (synchronous arrivals)

- Assume, w.l.o.g., that task **indexes are ordered by increasing relative deadlines**.
- **Compute the longest response time** for each task i, as

$$R_i = C_i + I_i$$

Computation time of task i

Interference from higher-priority tasks

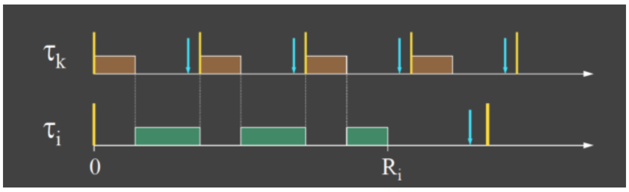
$$R_i \leq D_i$$

If the Worst Case Response Time (WCRT) is smaller from the deadline, then we are OK!

WCRT: the maximum response time among all jobs of the task

Process

- Interference on τ_i by high priority tasks



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

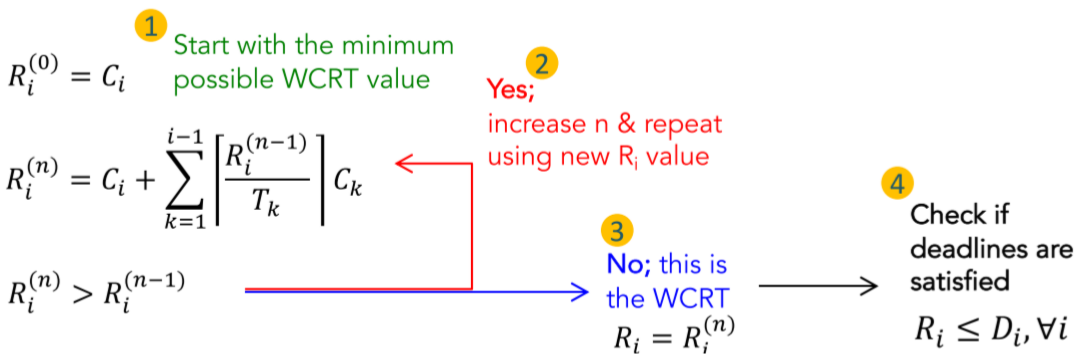
But: But when scheduling, we would not know R_i at the first time

- So we will use recursion way "fixed point iteration"

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

1. Define as X, the WCRT of Task i, which of course is **larger than** C_i;
2. Calculate the interference Y that Task i will experience in X;
3. If X is smaller than Y+ C_i, then our estimation was wrong, and we replace X with Y+ C_i. And we return to Step 2.
4. If X is larger than Y+ C_i, then our fixed iteration terminates.

   It is because, when X is larger than $Y + C_i$, in the next iteration, the value will have no change, so we can directly use $Y + C_i$ as next X



Note: I think, if we start directly from D_i, we can get result after only 1 iteration

Property

RTA is an **exact** test: **sufficient and necessary**

6.4. Guarantee Tests for EDF

An exact test (for $D_i = T_i$)

A set of periodic tasks, with $D_i = T_i$, for every i, is schedulable **if and only if** it holds:

$$U \leq U_{lub}^{EDF} = 1$$

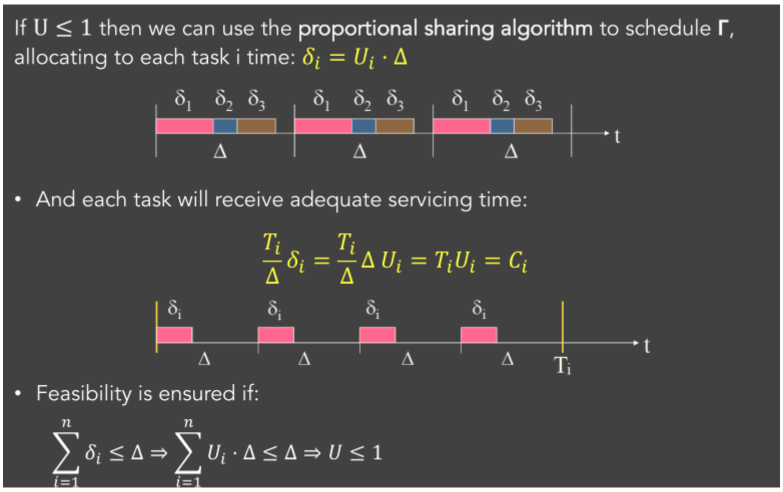
<u>Proof</u>

Necessity (only if)

that $U_\Gamma > 1$ means no algorithm can schedule task set Γ

Sufficiency (if)

With the condition that the EDF is the optimal wrt feasibility



If schedulable with proportional sharing (which is true if U≤1), then it is also schedulable with EDF (since EDF is optimal)

Test for $D_i < T_i$

In any interval $[t_1, t_2]$ the **computational demand** $g(t_1, t_2)$ of the task set must be no greater than the available time:

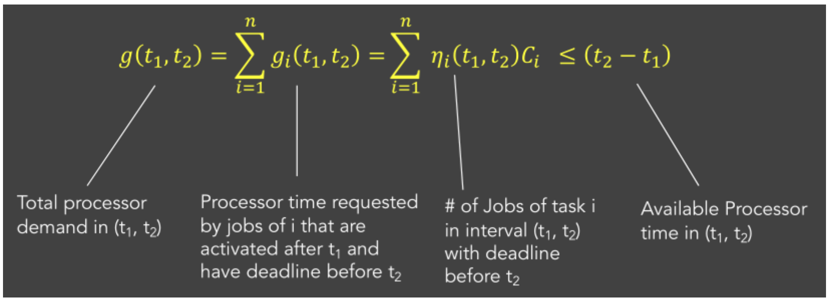
$$g(t_1, t_2) \leq (t_2 - t_1), \quad \forall t_1, t_2 > 0$$

When tasks are activated **simultaneously**, we can rewrite this as

$$g(0, L) \leq L, \quad \forall L > 0$$

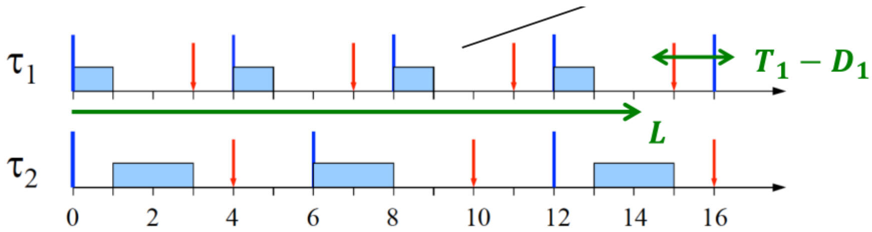
Calculation

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



When cocurrent activations:

$$\eta_i(0, L) = \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor$$



Fasten Calculation

1. Check only when we have a task deadline (g is a step function, i.e., **remains constant between task deadlines.**)

$$g(0, d_k) < d_k \Rightarrow g(0, L) < L, \quad \forall L : d_k \leq L < d_{k+1}$$

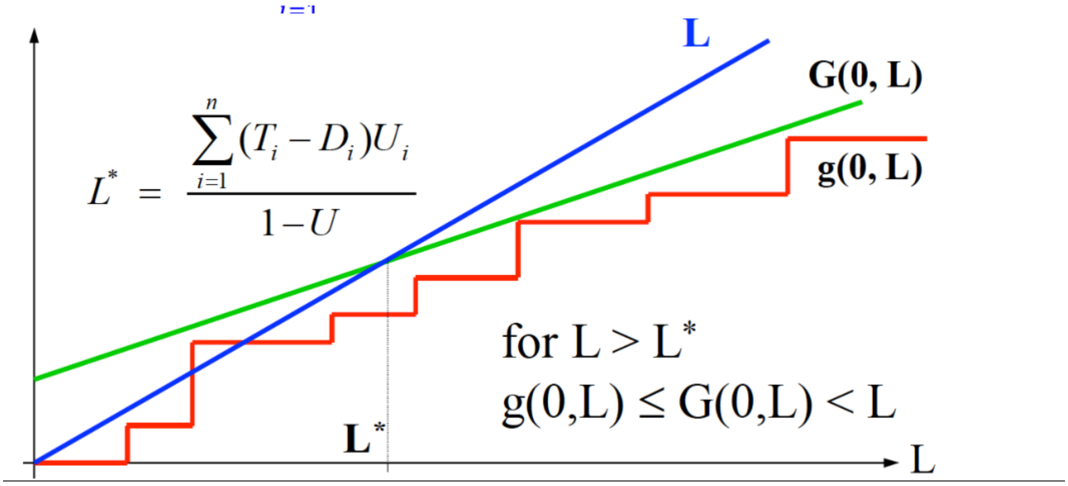
2. If all tasks are activated at t=0, we need only to check for $L \leq H$ (Hyperperiod)
3. We can further limit the checkpoints by using a refined function that **upper-bounds G**

Upper-bounds G

$$g(0,L) = \sum_{i=1}^{n} \left\lfloor \frac{L+T_i-D_i}{T_i} \right\rfloor C_i \leq \sum_{i=1}^{n} \left(\frac{L+T_i-D_i}{T_i} \right) C_i \Rightarrow$$
$$g(0,L) \leq \sum_{i=1}^{n} (T_i - D_i)U_i + L \cdot U \triangleq G(0,L)$$

If we know that G(0,L) is smaller than L (for some values of L), then we know that g(0,L) is also smaller than L.

However, G(0,L) is not always smaller than L. We can find an L^*



Synchronous Periodic Case Theorem:

A set of synchronous periodic tasks with relative deadlines less than or equal to periods can be scheduled by EDF if and only if

$$g(0,L) \leq L, \quad \forall L \in D = \{d_k \mid d_k \leq \min\{H, \max\{D_{\max}, L^*\}\}$$

where

$$H : \text{Hyperperiod}$$
$$D_{max} : \text{the maximum relative deadline in the task set}$$
$$L^* : \text{the parameter related to function G}$$

7. Summary

7.1. Summary For Tests

	$D_i = T_i$	$D_i \leq T_i$
RM	Sufficient tests; O(n) complexity LL: $\sum U_i \leq n(2^{1/n} - 1)$ HB: $\Pi(U_i + 1) \leq 2$ Exact test; pseudo-poly. RTA Exact test; O(n) compl. $\sum U_i \leq 1$ (for harmonic periods)	Exact test; pseudo-poly. Response Time Analysis (RTA) $R_i \leq D_i, \forall i$ $R_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$
EDF	Exact test; O(n) complexity $\sum U_i \leq 1$	Exact test; pseudo-poly. Processor Demand Analysis $g(0, L) \leq L, \quad \forall L > 0$ (Special criteria can be used to reduce complexity)

7.2. RM VS EDF

- RM is easier to implement as it suffices to have a kernel that can handle fixed priorities.
- EDF presumes more sophisticated priority-handling, but induces fewer preemptions and fewer context switches.
- EDF achieves higher utilization (up to 1). RM achieves smaller utilization unless special conditions hold.
- EDF is able to handle overloads in a more predictable way.

When Permanent Overload Happens

Permanent Overload: When the total utilization value increases (and stays there)

RM under permanent overload

- High priority tasks are executed at the necessary rate;
- Low priority tasks **are blocked**.

EDF under permanent overload

- All tasks are executed **at a slower rate;**
- No task is blocked.