Embedded Systems - Real-Time Scheduling

<u>part 7</u>

Scheduling aperiodic tasks

Running together periodic and aperiodic tasks
Using servers to run aperiodic tasks
Servers with fixed priorities
Servers with dynamic priorities

Putting together periodic and aperiodic tasks

Periodic tasks

Adequate, for example, to situations in which it is necessary to keep track of physical entities, normally continuous, or produce regularly a certain value or actuation.

Aperiodic tasks

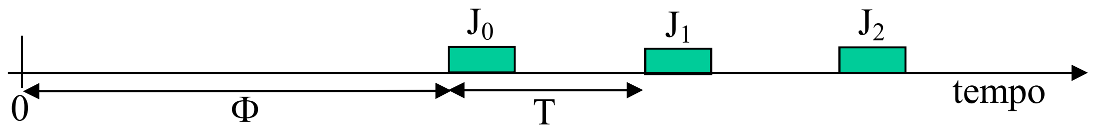
Conversely, adequate to situations in which we cannot pre determine the next activation instant, such as alarms, operator interfaces, or other asynchronous events.

Hybrid systems

Applications composed by a mix of periodic and aperiodic tasks. It is the most common case in real applications.

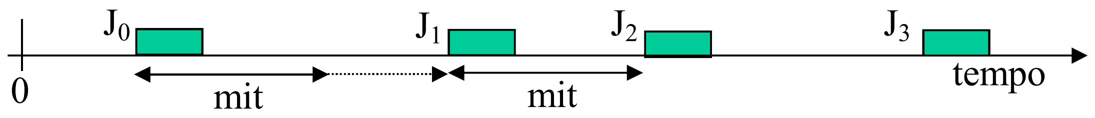
Putting together periodic and aperiodic tasks

- **Periodic tasks**



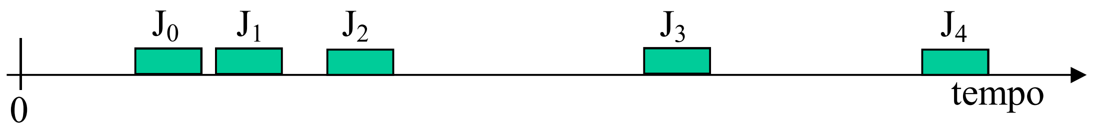
nth *instance* activated at $a_n=n*T+\Phi$ (well known worst-case)

- **Sporadic tasks**



In the worst-case, reverts to a periodic task with períod = *mit*

- **Aperiodic tasks**



Characterized stocastically, only

- How to **limit interference** over periodic tasks?
- How to provide the **best Quality-of-Service** possible?

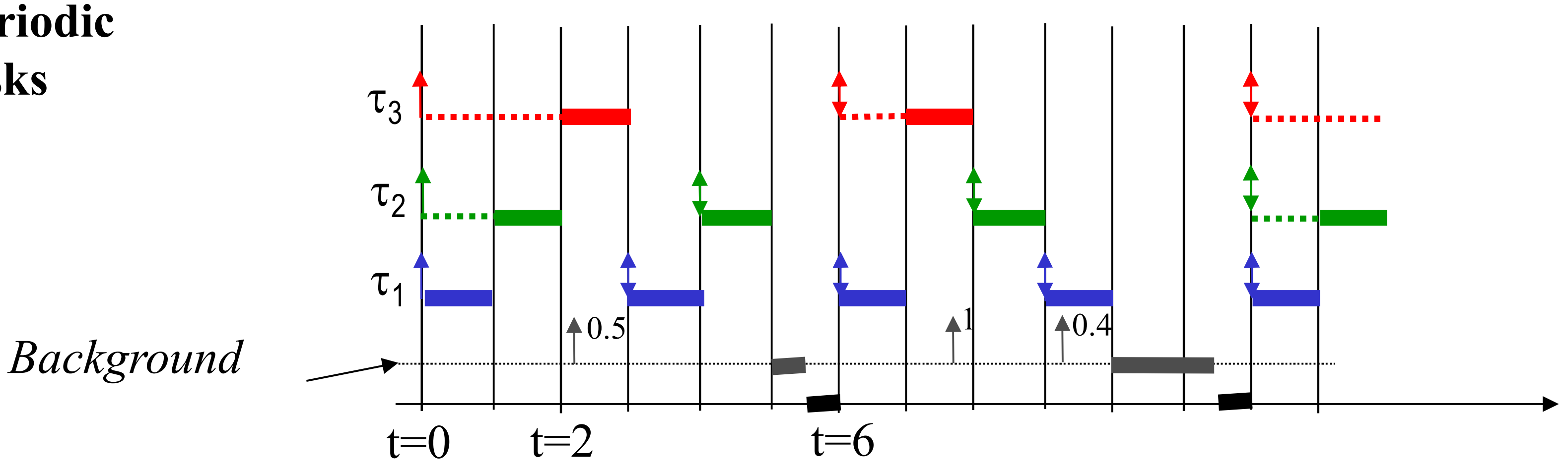
Execution in the background

A common and simple way to combine both types of tasks consists in giving **absolute priority** to **periodic tasks** and execute **aperiodic ones** in the intervals of CPU time **left free by the periodic subsystem**.

We say that the aperiodic tasks are **executed in the *background*** (lowest priority level in the system)

τ_i	T_i	C_i
1	3	1
2	4	1
3	6	1

Periodic tasks

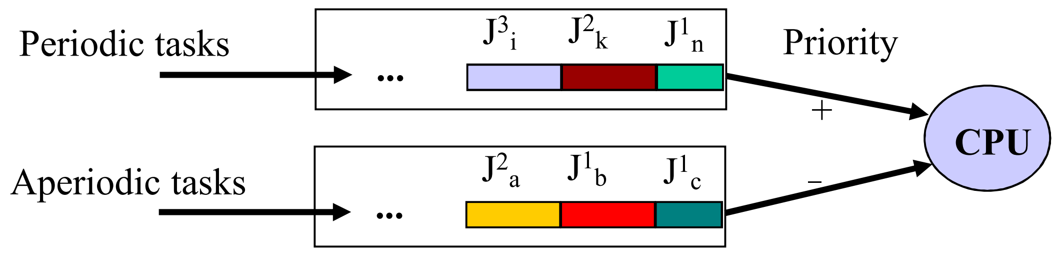


Execution in the background

Executing aperiodics in the background is **easy to implement** and **does not interfere with the periodic subsystem** (might cause small interference due to interrupt service routines)

Conversely, aperiodic tasks can suffer **large service delays** depending on the periodic load (can be computed considering aperiodic tasks as the lowest priority ones)

Poor performance for **real-time aperiodic tasks (alarms)** but **suitable for non real-time aperiodic tasks (file transfers)**.

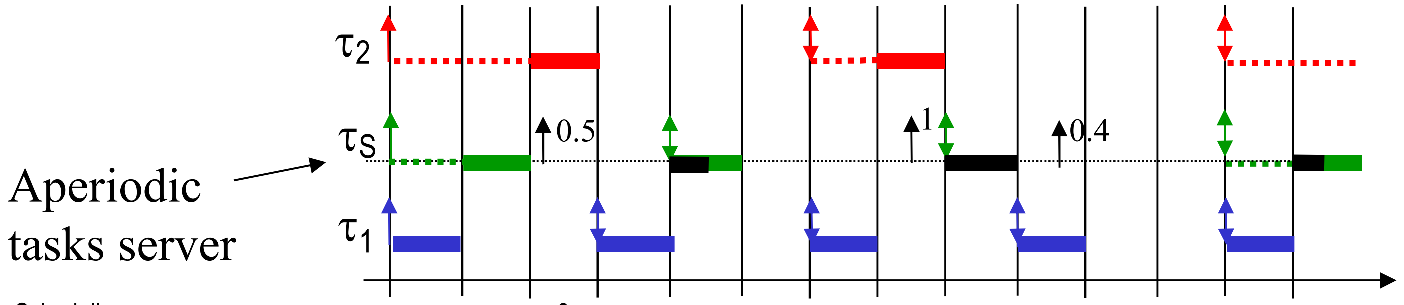


Using servers to run aperiodic tasks

One way to **improve the service** to aperiodic tasks (when background execution is not enough to meet the real-time constraints) consists in using a special **periodic task** whose purpose is just to **execute active aperiodic tasks**.

This task is called **aperiodic tasks server** and it is characterized by a period T_S and a capacity C_S

This way, it is possible to insert the server in the periodic subsystem with the **required priority level** to obtain the desired service level



Using servers to run aperiodic tasks

There are **several types** of servers to run aperiodic tasks, either with **fixed or dynamic priorities**, that vary in terms of:

- **Impact** on the schedulability of the periodic subsystem
- **Average response time** to aperiodic execution requests
- Computing and memory **overhead** and implementation cost.

- **Fixed priorities:** Polling server, deferrable server, priority exchange server, sporadic server,...
- **Dynamic priorities:** adapted versions of the fixed priority servers, total bandwidth server (TBS) and constant BW server (CBS),...

Worst-case response time to aperiodic requests

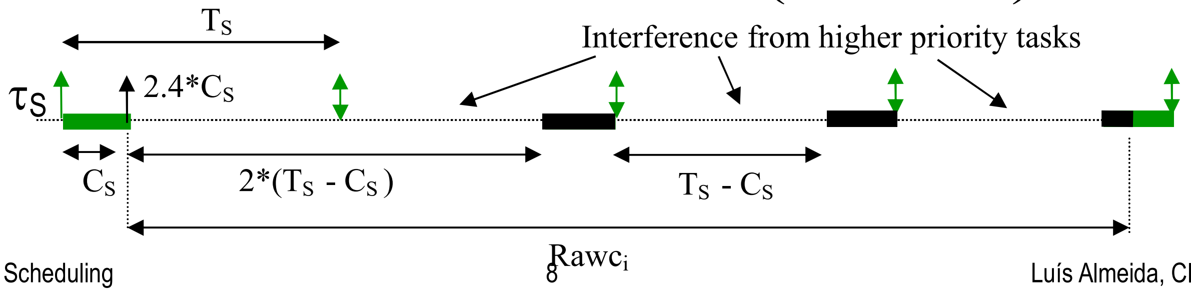
Worst-case response time

(similar for all servers that can be modeled as a periodic task under full load)

Consider that:

- The server is a periodic task τ_S (C_S,T_S)
- Suffers maximum jitter at the instant of the aperiodic request
- Suffers maximum interference delay in all subsequent instances

$$Rawc_i = Ca_i + (T_S - C_S) * \left(1 + \left\lceil \frac{Ca_i}{C_S} \right\rceil\right)$$



Worst-case response time to aperiodic requests

Worst-case response time

(cont.)

If for the same server there are *Na* aperiodic requests queued waiting for service (scheduled according to a certain criterion – consistent with index *k*), the schedulability test for aperiodic task i is:

(consider that all aperiodic requests are issued at the same instant → critical instant, worst-case situation, and there might be recurrent arrivals of the same request)

$$\forall_{i=1..Na}\ Rawc_i = Ca_i + (T_S - C_S) * \left(1 + \left\lceil \frac{\sum_{k=1}^{i} Ca_k}{C_S} \right\rceil\right) < Da_i$$

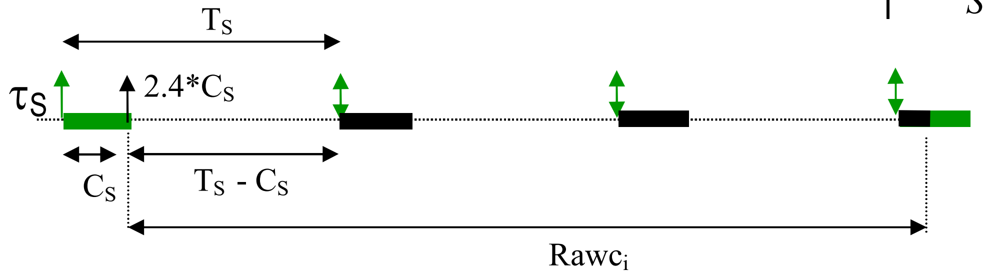
Worst-case response time to aperiodic requests

Worst-case response time

(cont.)

If the server has the highest priority in a fixed priorities system (or in time-triggered slotted systems such as TDMA) there is no more the interference delay that affects the subsequent instances and the worst-case response time is given by:

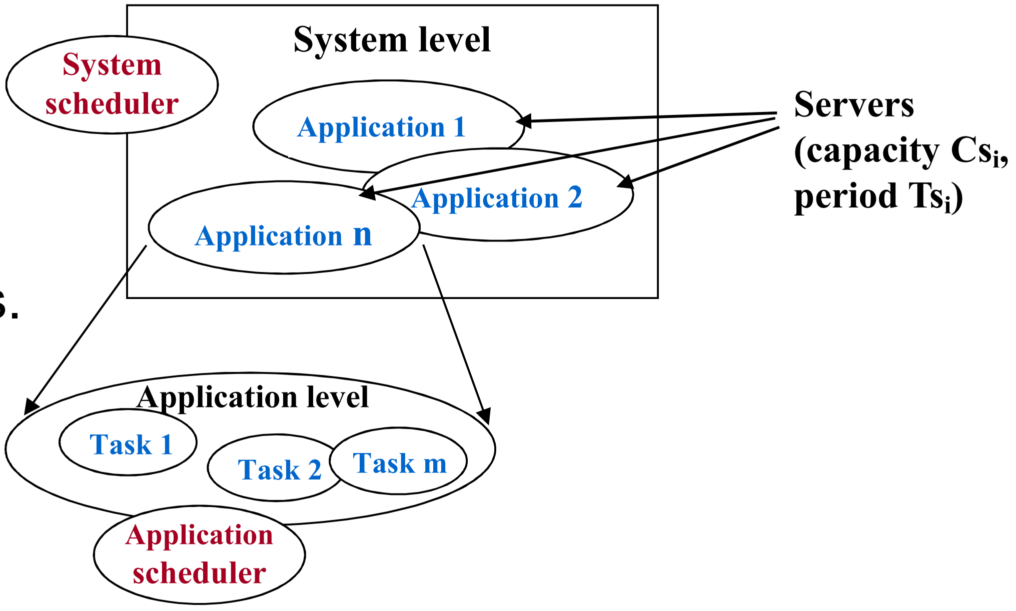
$$Rawc_i = Ca_i + (T_S - C_S) * \left\lceil \frac{Ca_i}{C_S} \right\rceil$$



Hierarchical scheduling

Taking a more general perspective, we can think of a system that runs **different applications** that must be guaranteed independently. One way to **bound the mutual interference** across applications is to **run each application within a server**.

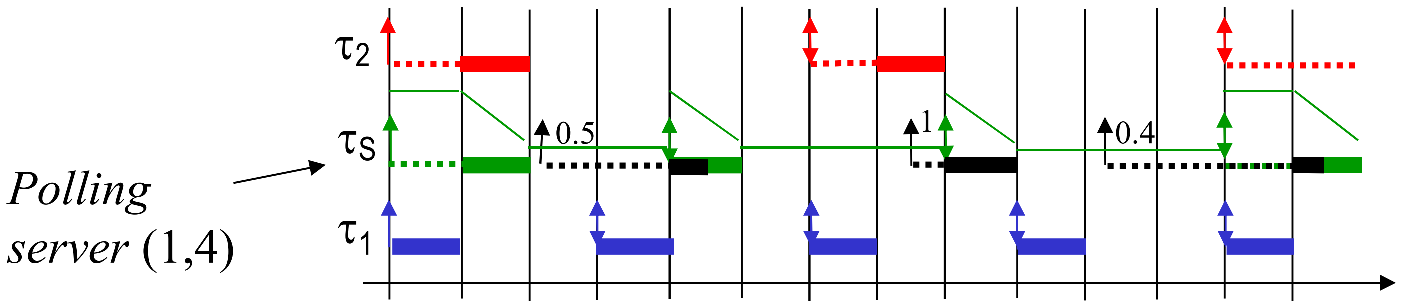
Thus we get two levels, the **system level** that manages the servers and the **server level** that manages the application specific tasks.



Polling server - PS

This **fixed priorities server** is completely equivalent to executing a periodic task. The **aperiodic requests** are executed **during the intervals of time assigned to the server** by the periodic tasks scheduler.

Note that, even if there are no requests to execute, the server capacity is used up anyway. The capacity is replenished every period.



Polling server - PS

Implementing a polling server is **rather simple**, requiring a single queue for the aperiodic requests and a control of the capacity used thus far.

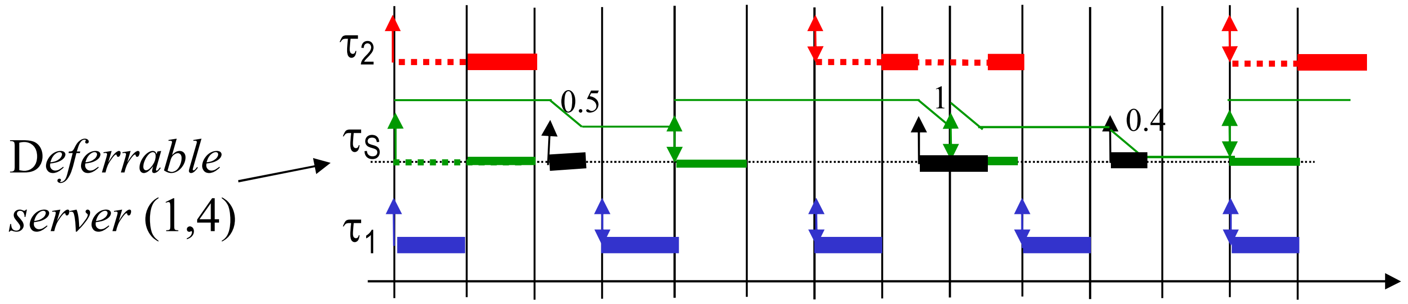
The **average response time** to the aperiodic requests is **improved** with respect to background execution because it is possible to execute the aperiodic tasks at a higher priority level. However, there are still intervals of unavailability corresponding to the server period.

The **impact** on the schedulability of the periodic subsystems is equivalent to that of the corresponding virtual periodic task.

Deferrable server - DS

This is a **fixed priorities server** that keeps its capacity when not used by the aperiodic requests. Therefore, an aperiodic request can get immediate service if the server is active and still has capacity in that period.

The server capacity is fully replenished every server period.



Deferrable server - DS

The **complexity** to implement a deferrable server is similar to that of a PS.

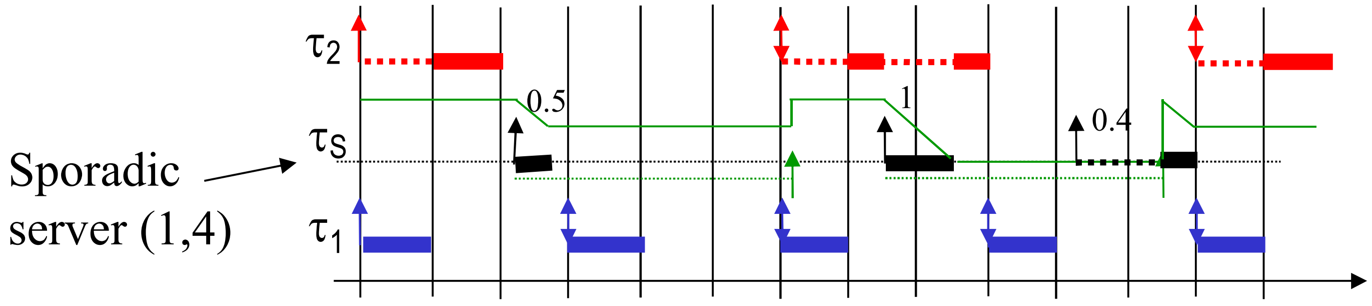
The **average response time** to the aperiodic requests is **improved** with respect to the PS because it is possible to use the server capacity even after the intervals assigned by the periodic scheduler, thus eliminating the unavailability periods of the PS between successive replenishments.

Nevertheless, this server has a **negative impact** (potential concentration of interference) on the schedulability of the periodic subsystem due to the possible deferred execution.

Sporadic server - SS

This **fixed priorities server** also **keeps its capacity when not used** by the aperiodic requests but **without penalizing the schedulability** of the periodic subsystem.

In this case, if the **execution** of the server **is deferred so is the replenishment** to enforce the **server bandwidth**. Basically, the amount consumed is always replenished a period later.



Sporadic server - SS

The complexity of implementing a sporadic server **is higher** than for a DS given the need to maintain a structure with the replenishment instants and the amounts to replenish.

The **average response time** to the aperiodic requests is **similar** to that of a DS.

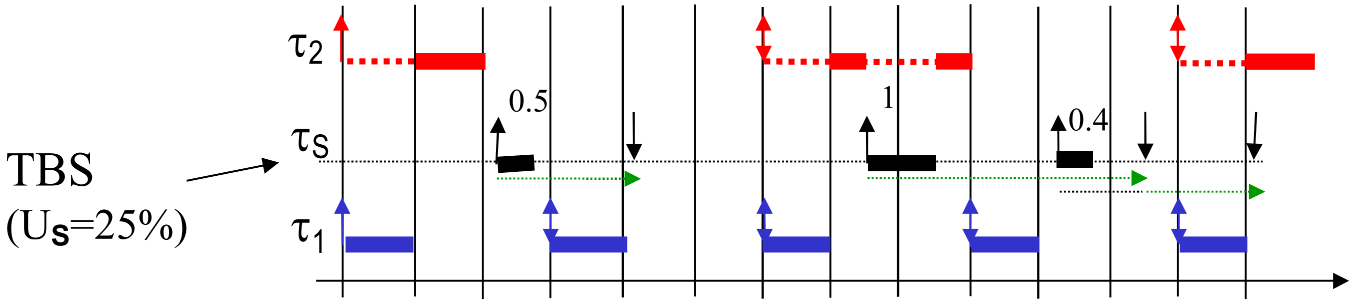
The **impact** on the schedulability of the periodic subsystem is similar to that of a normal periodic task, similarly to the PS, due to the bandwidth enforcement with the deferred replenishments, as opposed to the DS.

Total bandwidth server - TBS

The Total Bandwidth Server (TBS) is a **dynamic priorities server** which purpose is to handle aperiodic requests in an EDF system, as early as possible but maintaining a given bandwidth so as to limit the impact on the periodic subsystem.

When a new requests arrives at instant r_k, a deadline d_k, is assigned

$$\mathbf{d_k = max\ (r_k, d_{k-1}) + C_k/U_S}$$



Total bandwidth server - TBS

The **complexity** of implementing a TBS is relatively small since it is only necessary to compute a deadline every time a request arrives and then it is inserted in the ready tasks queue, together with the periodic tasks.

The **average response time** to aperiodic requests is **shorter** than with dynamic priorities versions of the previous servers.

The **impact** on the schedulability of the periodic subsystem is similar to that of a periodic task with the same bandwidth as that of the server. Using EDF+TBS

$$U_p + U_S \leq 1$$

The TBS is **vulnerable to overruns** since there is **no control over the actual execution time** (no capacity control, for example).

Constant bandwidth server - CBS

The Constant Bandwidth Server (CBS) is a **dynamic priorities server** designed to solve the robustness problem of the TBS enforcing **bandwidth isolation**.

This is achieved using a **capacity management scheme** (Q_S ,T_S).

When a request arrives at instant r_k, the deadline d_S is computed as:

if $r_k + c_S/U_S < d_S^{current}$ **then** **keep** d_S

or else $d_S = r_k + T_S$ and $c_S = Q_S$

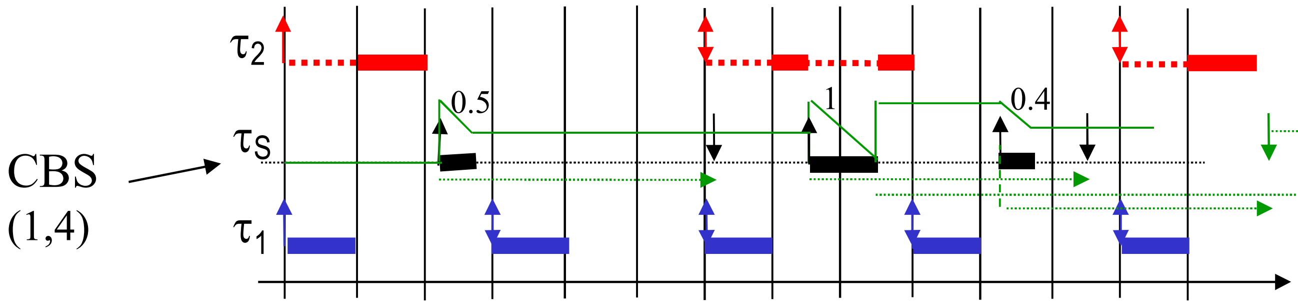
When the server capacity is exhausted (c_S=0) it is immediately replenished but d_S is postponed to enforce the server bandwidth

$$d_S = d_S + T_S \quad \text{and} \quad c_S = Q_S$$

Constant bandwidth server - CBS

The deadline assignment scheme used by CBS **enforces the server bandwidth** independently of the aperiodic load that the server actually executes.

If a task executing inside a CBS **executes for longer** than expected, its **deadline is automatically postponed**, which in practice corresponds to lowering the task priority



Constant bandwidth server - CBS

The **complexity** of implementing a CBS is higher than that of the TBS due to the required capacity management. Anyway, there is still **one single ready queue**, for periodic and aperiodic tasks, managed by deadlines.

The **average response time** to aperiodic requests is **similar** to that of the TBS.

The **impact** on the schedulability of the periodic subsystem is equal to that of a task with the same parameters than the server.

Using EDF+CBS

$$U_p + U_S \leq 1$$

Constant bandwidth server - CBS

The main motivation to use a CBS resides in the **bandwidth isolation that it provides**.

If a task is served by a CBS with bandwidth U_S, in any interval Δt multiple of its period, such task (and server) will never require more than $\Delta t * U_S$ of CPU.

Any task τ_i (C_i, T_i) schedulable under EDF is equally schedulable running within a CBS with $\mathbf{Q_S = C_i}$ and $\mathbf{T_S = T_i}$

Therefore, a CBS can be used for:

- Protecting a system of tasks ***overruns***
- Guaranteeing a **minimal service** to soft real-time tasks
- **Reserve bandwidth** for any activity.

Wrapping up

- Executing together **periodic** and **aperiodic tasks**
  - Basic technique
    - Executing aperiodic tasks in the **background**
  - Using **servers** to run **aperiodic tasks**
    - **Fixed priority** servers
      - Polling Server - PS
      - Deferrable Server - DS
      - Sporadic Server - SS
    - **Dynamic priority** servers
      - Total Bandwidth Server – TBS
      - Constant Bandwidth Server - CBS