Real-Time Systems

Lecture 8

Aperiodic task scheduling

Joint execution of periodic and sporadic tasks
Use of aperiodic task servers
Fixed-priority aperiodic task servers
Dynamic-priority aperiodic task servers

Last lecture (7)

- Access to **shared resources**: blocking
- The **priority inversion**: need to bound and analyze
- **Basic techniques** to allow exclusive access to shared resources
  - Disable interrupts, preemption
- **Advanced techniques** to allow exclusive access to shared resources
  - ***The Priority Inheritance Protocol – PIP***
  - ***The Priority Ceiling Protocol – PCP***
  - ***The Stack Resource Protocol - SRP***

Joint scheduling of periodic and aperiodic tasks

Periodic tasks

Suitable e.g. to applications where it is required sampling regularly a given physical entity (e.g. a temperature, pressure, torque, speed), or actuate regularly on the system via an actuator.

Aperiodic+sporadic tasks

Suitable to scenarios where the event activation instants cannot be forecast, e.g. alarms, human-machine interfaces, external asynchronous interrupts.

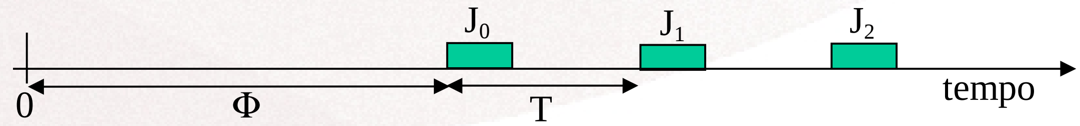
Hybrid systems

Applications which contain both types of tasks.

Many (most?) real systems contain naturally both periodic and aperiodic events/tasks

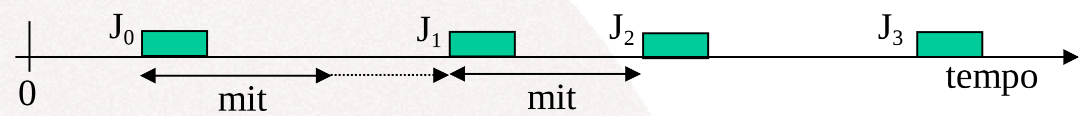
Joint scheduling of periodic and aperiodic tasks

Periodic tasks



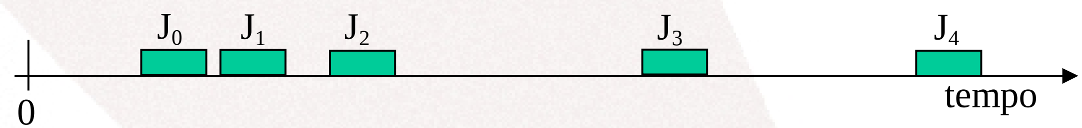
n[th] task instance activated at $a_n = n*T+\Phi$ (worst-case is well defined)

Sporadic tasks



In worst-case it behaves as a periodic task with period = *mit*

Aperiodic tasks



- Only characterizable via probabilistic methods
- How to **bound the interference** on periodic tasks?
- How to guarantee an **acceptable/best possible quality of service (QoS)**?

Background execution

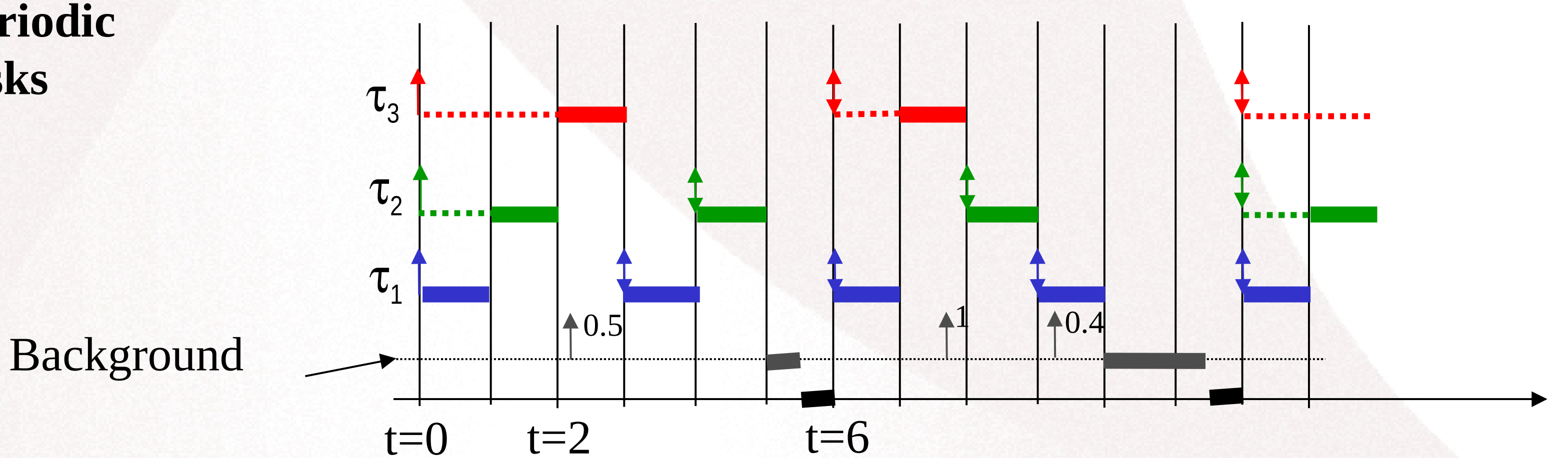
A simple way of combining both task types is giving **higher priority** to the **periodic tasks** than to the sporadic ones.

Thus the **sporadic tasks** only **execute** when there are **no ready periodic tasks**.

In this case the aperiodic tasks are executed in **background** with respect to the periodic ones – **background execution**.

τ_i	T_i	C_i
1	3	1
2	4	1
3	6	1

Periodic tasks



Background execution

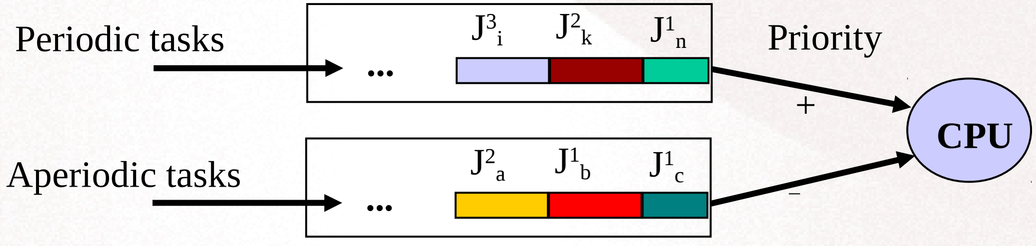
The background execution is very **easy to implement** and does **not interfere** directly with the periodic system/tasks.

- However, **interference may still occur indirectly**, via interrupt service routines, non-preemptive system calls, shared resources, etc.

On the other hand, aperiodic tasks may suffer **big delays**, depending on the periodic load.

- This delay may be upper-bounded considering the aperiodic tasks as a lowest priority task.

The performance is **poor for real-time tasks**, tough it can be **acceptable to non real-time ones.**

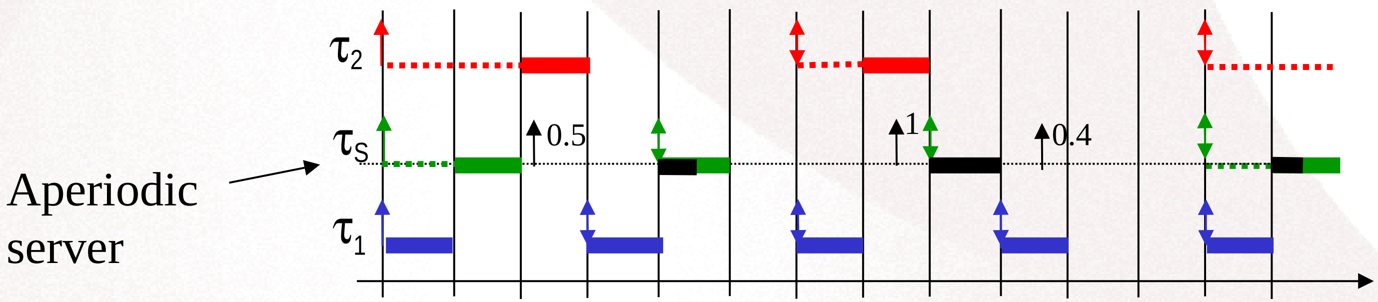


Aperiodic servers

When the background execution service does not allow meeting the real-time constraints of aperiodic tasks, the **response time** of these can be **improved** by using a **pseudo-periodic task** whose only function is to execute the active aperiodic tasks.

This pseudo-task is designated **aperiodic server** and is characterized by a period T_s and a capacity C_s.

It is now possible to insert the aperiodic server in the set of periodic tasks and assign it **sufficient priority** to provide the **required QoS**.



Aperiodic servers

- There are many types of aperiodic servers, both based on fixed and dynamic priorities, which vary in terms of:
  - **Impact** on the schedulability of the periodic tasks
  - **Average response time** to aperiodic requests
  - **Computational cost/overhead**, memory and implementation complexity.
- **Fixed priority:** ***Polling Server***, ***Deferable Server***, *Priority Exchange Server*, ***Sporadic Server***,...
- **Dynamic priorities:** Adapted fixed-priority servers, **Total Bandwidth Server**, **Constant Bandwidth Server**, ...

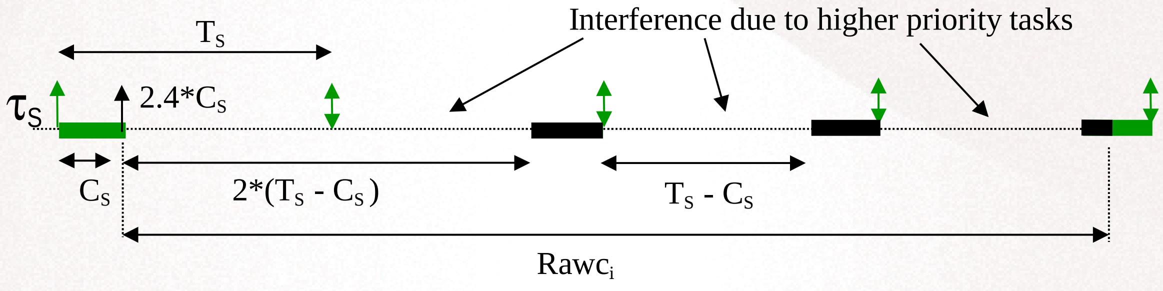
Worst-case response time to aperiodic requests

Worst-case response time:

- Equal to all servers that can be modeled by a periodic task
- It is assumed that (worst-case scenario):
  - The server is a **periodic** task τ_S (C_S, T_S)
  - Suffers **maximum jitter** on the instant of the aperiodic request
  - Suffers **maximum delay** in all successive instances

$$Ra_{wc_i} = Ca_i + (T_s - C_s) * \left(1 + \left\lceil \frac{Ca_i}{Cs} \right\rceil\right)$$

Ca_i = execution time of aperiodic task i



Worst-case response time to aperiodic requests

Worst-case response time (cont):

- If there are **several aperiodic requests** queued for the same server *i* (Na_i), sorted by a suitable criteria, the schedulability test for aperiodic requests is:

$$\forall i = 1..\mathrm{Na}, Ra_{wc_i} = \left(\sum_{k=1}^{i} Ca_k \right) + (T_s - C_s) * \left(1 + \left\lceil \frac{\sum_{k=1}^{i} Ca_k}{Cs} \right\rceil \right) \leq Da_i$$

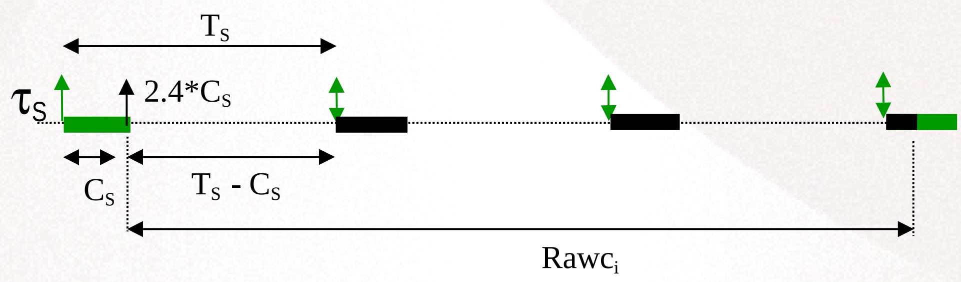
- It is assumed that **all requests** are issued at the **same instant**, which corresponds to the worst-case scenario.

Worst-case response time to aperiodic requests

Worst-case response time (cont):

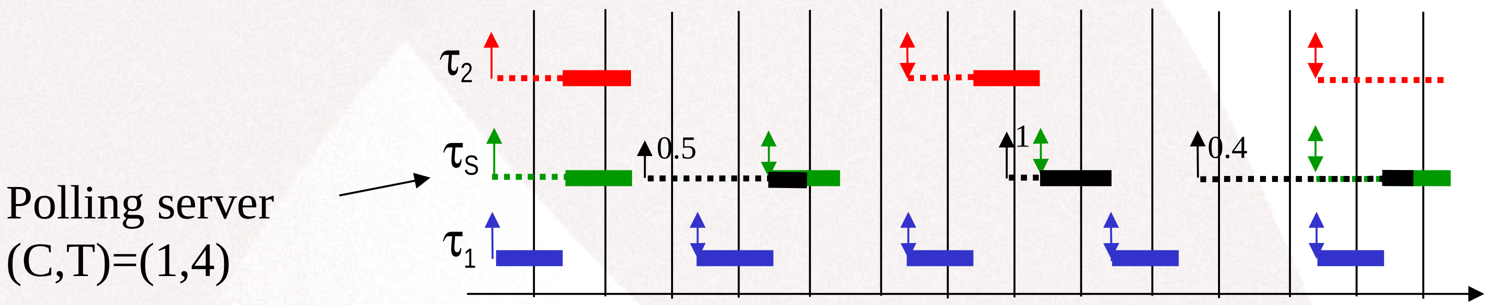
- If, in a **fixed priority system**, the aperiodic **server** has the **highest priority**, the interference term, due to higher priority tasks, disappears, and the worst-case response time is:

$$Ra_{wc_i} = Ca_i + (T_s - C_s) * \left\lceil \frac{Ca_i}{Cs} \right\rceil$$



Polling server (PS)

- This **fixed priority** server is completely equivalent to the execution of a periodic task. The **aperiodic requests** are **served** only during the **execution intervals** granted to the **server** by the periodic task scheduler.



Polling server (PS)

- The **implementation** of a polling server is **relatively simple**. It only requires a queue for the aperiodic requests and control of the capacity used.
- The **average response time** to aperiodic requests is **better** than the one obtained with **background execution**, since it is possible to elevate the priority of the server. However it has relatively **long unavailability periods**.
- The **impact** on the periodic task set is exactly the **same** as the one of a **periodic task**. So, e.g., using RM + PS

(U_p: utilization of n periodic tasks)

$$U_p + U_s \le (n+1)\left(2^{\frac{1}{n+1}} - 1\right)$$

Polling server (PS)

- The previous test (Liu & Layland bound) is independent of the utilization of each task. It is possible to improve the test (i.e. obtain tighter bounds) for particular scenarios.
- Giving the **highest priority to the server** (corresponding utilization rate **$U_S=C_S/T_S$**), the L&L least upper bound becomes:

$$U_p+U_s \le U_s+n\left(\left(\frac{2}{U_s+1}\right)^{\frac{1}{n}}-1\right)$$

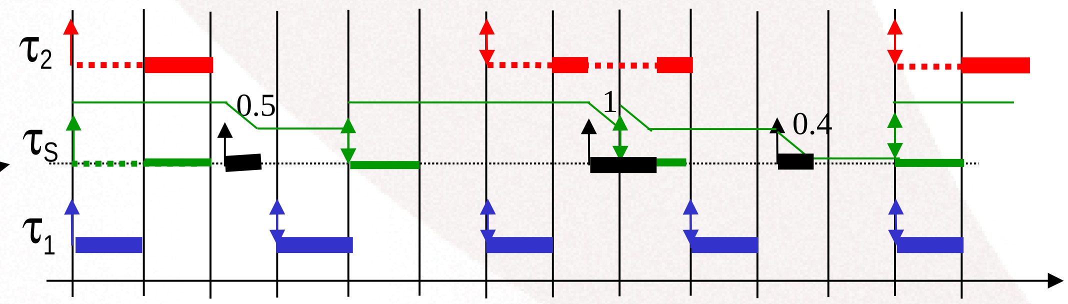
- And when **n->∞,**

$$U_p+U_s \le U_s+\ln\left(\frac{2}{U_s+1}\right)$$

Deferrable server (DS)

- The basic idea of this **fixed-priority server** is to handle aperiodic requests from the **beginning** of its execution until:
  - **End of its period** (T_S) or
  - Its **capacity** (C_S) gets **exhausted**
- The capacity is **replenished** at the beginning of each period.

Deferrable server
(C,T)=(1,4)



Deferrable server (DS)

- The **complexity** of the implementation of a DS is similar to the one of a PS.
- The **average response time** to aperiodic requests is **improved** with respect to the PS, since it is possible to use the capacity of the DS during the whole period, provided that its capacity is not exhausted.
- However, there is a **negative impact** on the schedulability of the periodic tasks. The reason for this impact is that the **delayed executions** increase the **load** on the **future**. E.g., it is possible having two consecutive executions (back-to-back execution).
  - Using RM+DS and a server with highest priority:

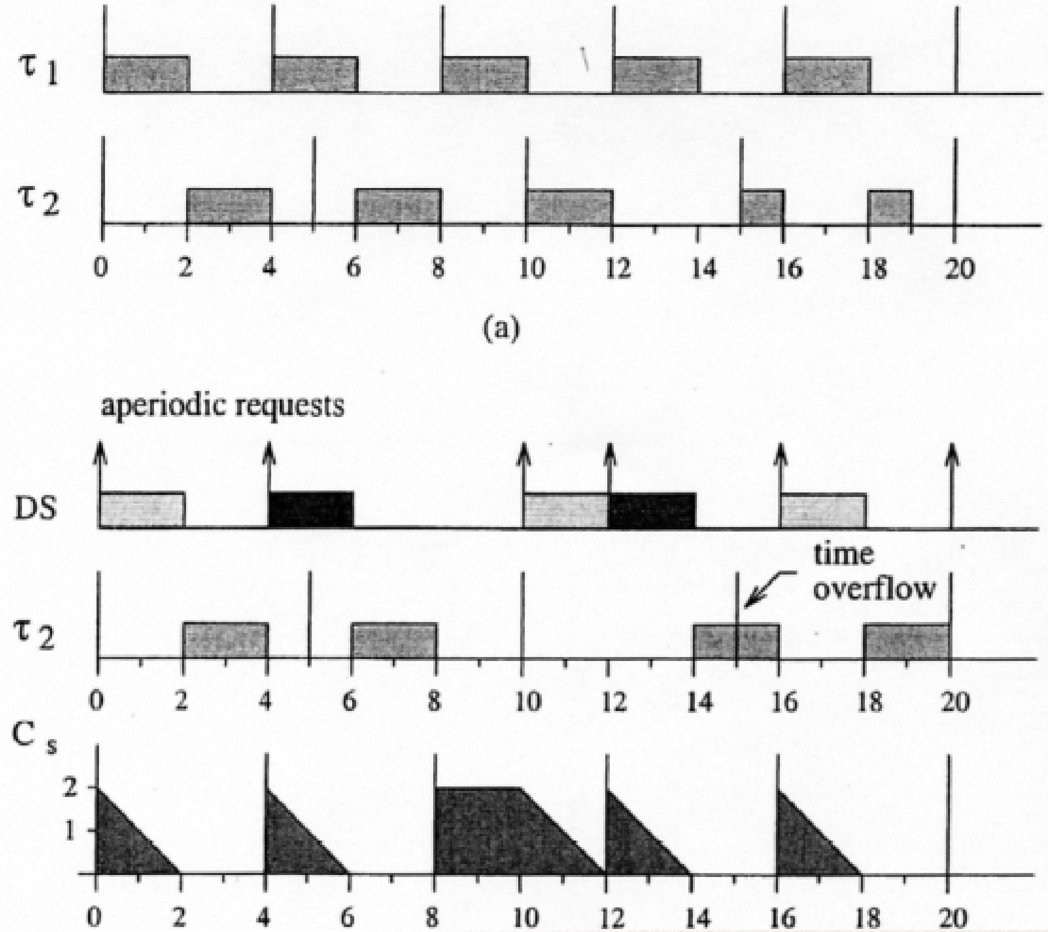
$$U_p + U_s \leq U_s + n\left(\left(\frac{U_s + 2}{2.\mathrm{U}_s + 1}\right)^{\frac{1}{n}} - 1\right)$$

Deferrable server (DS)

Illustration of a scenario in which **replacing** a **periodic task by a DS** causes **deadline misses**

Periodic tasks

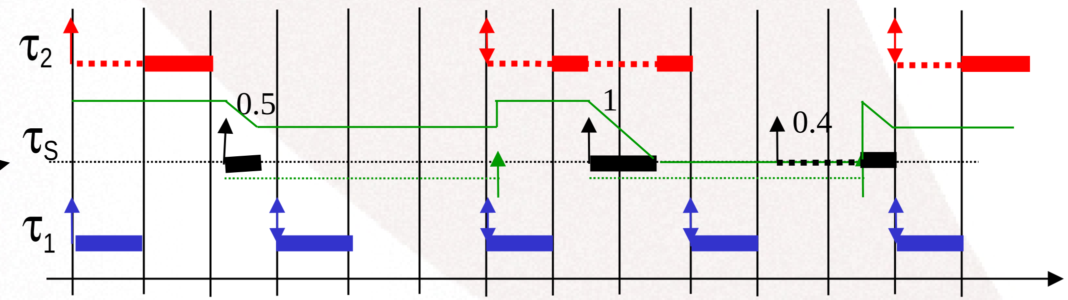
Task 1 replaced by a DS



Sporadic server (SS)

- The basic idea of this **fixed-priority** server is also allow the **execution** of the server at **any instant** (as the DS), however **without penalizing** the **schedulability** of the periodic system.
- The SS replenishes the capacity **not** at the end of the **period** but instead according with the time instants in which the **capacity** is actually **used** (consumption instants + T_S)

Sporadic Server
(C,T)=(1,4)



Sporadic server (SS)

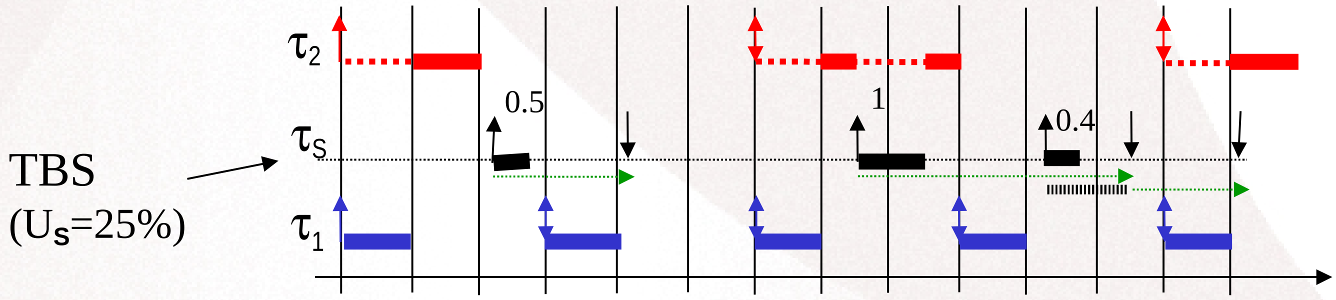
- The implementation **complexity** of a sporadic server is higher than the one of PS and DS, due to the computation of the replenishment instants and, more importantly, to the **complex timer management**
- The **average response time** to aperiodic requests is similar to the one of the DS
- The **impact** on the schedulability of the periodic tasks is exactly the same as the one of the PS,
  - The SS executes as soon as it has capacity, but the technique used to **replenish** the **capacity preserves** the **timing behavior** and bandwidth (unlike the DS).
- Using RM+SS and giving higher priority to the server:

$$U_p + U_s \leq U_s + n\left(\left(\frac{2}{U_s+1}\right)^{\frac{1}{n}} - 1\right)$$

Total Bandwidth Server (TBS)

- The ***Total Bandwidth Server*** is a **dynamic priority** server which has the objective of executing the aperiodic requests **as soon as possible** while **preserving** the **bandwidth** assigned to it, to not disturb the periodic tasks. It was developed for EDF systems.
- When an aperiodic request arrives (r_k), it receives a deadline d_k,

$$d_k = max(r_k, d_{k-1}) + \frac{C_k}{U_s}$$



Total Bandwidth Server (TBS)

- TBS is **simple** to implement and has low overhead, since it only requires a simple computation (deadline for each arrival). Then the aperiodic request is inserted in the ready queue and handled as any other task.
- The **average response time** to aperiodic requests is **smaller** than the one obtained with dynamic- priority versions of fixed-priority servers.
- The **impact** on the schedulability of the periodic task set is equal to the one of a periodic task with utilization equal to the one granted to the server. Using EDF+TBS:

$$U_P + U_S \leq 1$$

- Requires **a priori knowledge of C_k** and is **vulnerable to *overruns*.**
  - After starting executing, a task may execute more time than the one declared

Constant Bandwidth Server (CBS)

- The ***Constant Bandwidth Server*** (CBS) is a **dynamic priority** server that was crated to **solve the robustness problem of TBS**, enforcing bandwidth isolation.
- This goal is achieve by managing the execution time based and considering a budget/capacity (Q_S ,T_S).
  - When an aperiodic request r_k arrives, it is computed a server *deadline* d_S, as follows:

$$\text{if} \quad r_k + \frac{c_s}{U_s} < d_s^{actual}, \quad \text{then} \quad d_s^{actual} \text{ does not change}$$

$$\text{otherwise} \quad d_s = r_k + T_s, \wedge \quad c_s = Q_s$$

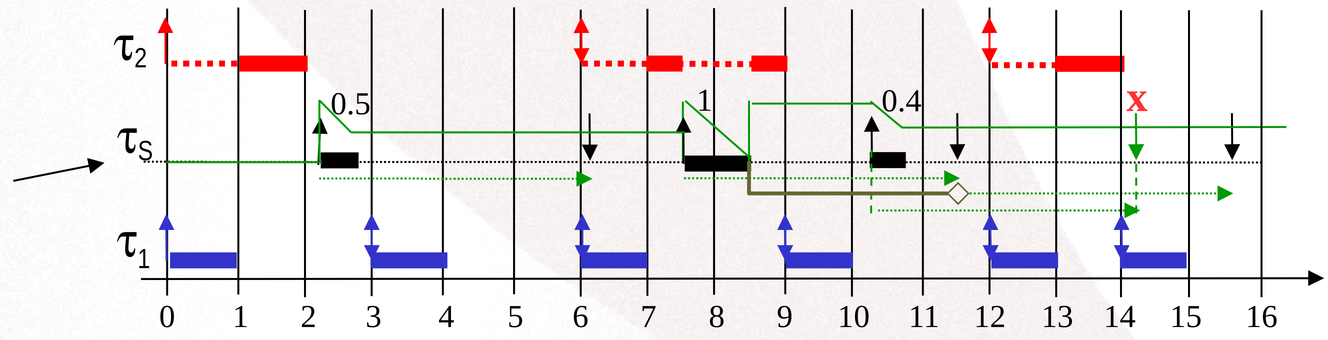
  - When the instantaneous capacity (c_s) gets exhausted, the capacity is replenished and the deadline postponed:

$$d_s = d_s + T_s, c_s = Q_s$$

Constant Bandwidth Server (CBS)

- The CBS server assigns deadlines in such a way that **prevents** the **bandwidth** given to the server **from being higher** the the one assigned to it.
- If a task **executes for longer** than expected, its **deadline is automatically postponed**, lowering the priority of the task. This can also be seen as if the task period was artificially increased, in such a way the the utilization is maintained.

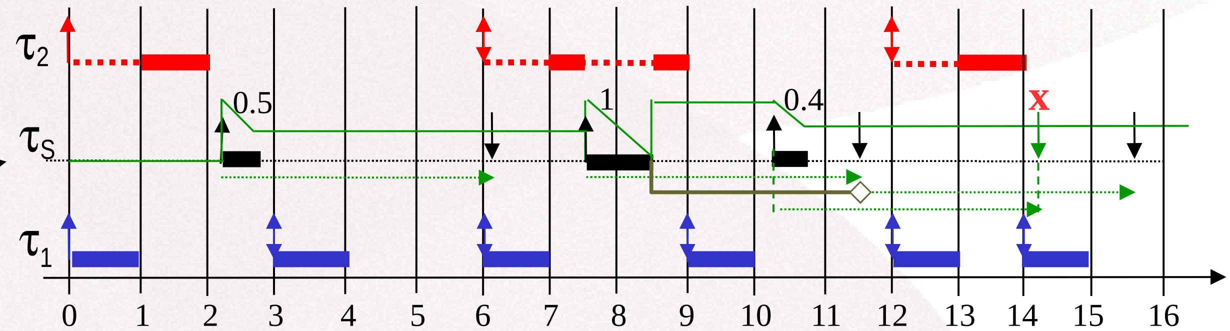
CBS
(Q,T)=(1,4)



Constant Bandwidth Server (CBS)

- Example

CBS
(Q,T)=(1,4)



Rules

- *Arrival*

If $r_k+c_s/U_s < d_s^{actual}$, d_s^{actual} doesn't change **[R1]**
Else $d_s=r_k+T_s$; $c_s=Q_s$ **[R2]**

- *C_s exhausted*

$d_s=d_s+T_s$; $c_s=Q_s$ **[R3]**

t=2.2: $d_s^{actual} < r_k$, thus **rule 2** applies
$d_s=r_k+T_s=2.2+4=6.2$; $c_s=1$

t=7.5: $d_s^{actual} < r_k$, thus **rule 2** applies
$d_s=r_k+T_s=7.5+4=11.5$; $c_s=1$

t=8.5: c_s exhausted, thus **rule 3** applies
$d_s=d_s+T_s=11.5+4=15.5$; $c_s=1$

t=10.2: $r_k+c_s/U_s=10.2+1/0.25=14.2 < d_s^{actual}$, thus **rule 1** applies
d_s^{actual} does not change; $c_s=c_s$

Constant Bandwidth Server (CBS)

- The implementation **complexity** of CBS is somehow **higher** than the one of **TBS**, due to the need to dynamically manage the capacity. Other than that, aperiodic tasks are put in the ready queue and handled as any regular periodic task.
- The **average response time** to aperiodic requests is **similar** to TBS.
- The **impact** on the schedulability of the periodic task set is equal to the one of a periodic task with an utilization equal to the one given to the server. Using EDF+CBS

$$U_P + U_S \leq 1$$

Constant Bandwidth Server (CBS)

- The big advantage of CBS is that it provides **bandwidth isolation**
- If a task is served by a CBS with bandwidth U_S, in any time interval Δt that task will **never require more than $\Delta t * U_S$** CPU time.
- Any task τ_i (C_i, T_i) schedulable with EDF is also schedulable by a CBS server with $Q_S = C_i$ e $T_S = T_i$
- **A CBS may be used to:**
  - Protect the system form possible ***overruns*** *in any task*
  - Guarantee a **minimum service** to *soft real-time tasks*
  - **Reserve bandwidth** to any activity

Summary of lecture 8

- Joint execution of **periodic and aperiodic** tasks
  - Background execution of aperiodic tasks
- Notion and characteristics of **aperiodic task servers**
  - **Fixed priority servers**
    - Polling Server - PS
    - Deferrable Server - DS
    - Sporadic Server - SS
  - **Dynamic priority servers**
    - Total Bandwidth Server – TBS
    - Constant Bandwidth Server - CBS