Aperiodic Servers

Real-Time Operative Systems Course

Paulo Pedreiras, DETI/UA/IT

November 24, 2021

Last lecture

- Priority inversion as a consequence of blocking
- Basic techniques to enforce exclusive access to shared resources:
  - Priority Inheritance Protocol – PIP
  - Priority Ceiling Protocol – PCP
  - Stack Resource Protocol- SRP

Agenda for today

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

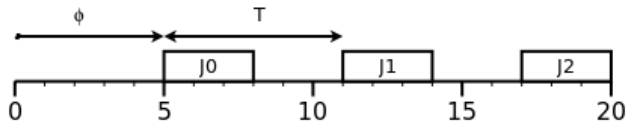
Joint scheduling of periodic and aperiodic tasks

Classification of tasks found in real-life systems

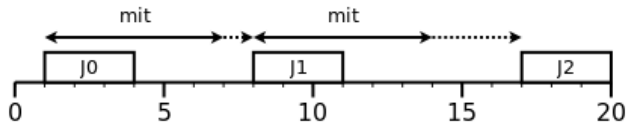
- **Periodic tasks**
  - Suitable e.g. to applications where it is required sampling regularly a given physical entity (e.g. acquire an image, a temperature, pressure, torque, speed), actuate regularly on the system, etc.
- **Aperiodic and Sporadic tasks**
  - Suitable to scenarios where the event activation 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

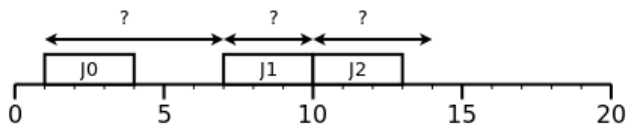
- Periodic tasks
  - k_{th} task instance activated at $a_k = k \cdot T_k + \Phi_k$
    Worst-case is well defined



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



- Aperiodic tasks

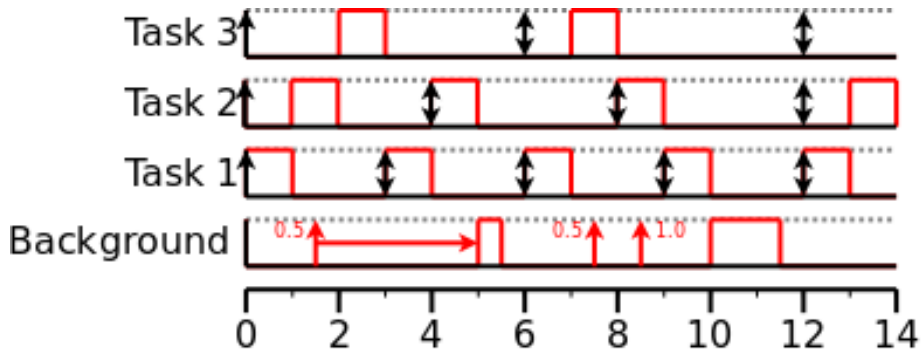


  - Only characterizable via probabilistic methods
  - How to **bound the interference** on periodic tasks?
  - How to guarantee an acceptable/best possible **quality of service** (QoS)?

Background execution

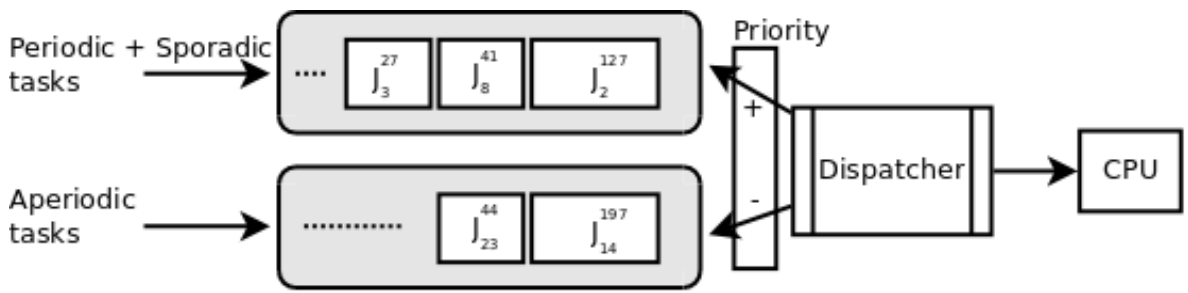
- A simple way of combining both task types is giving higher priority to the periodic tasks than to the aperiodic/sporadic ones.
- Thus the aperiodic/sporadic tasks only execute when **there are no ready periodic tasks** .
- Aperiodic/sporadic tasks are executed in background with respect to the periodic ones – background execution.

τ_i	C_i	T_i
1	1	3
2	1	4
3	1	6



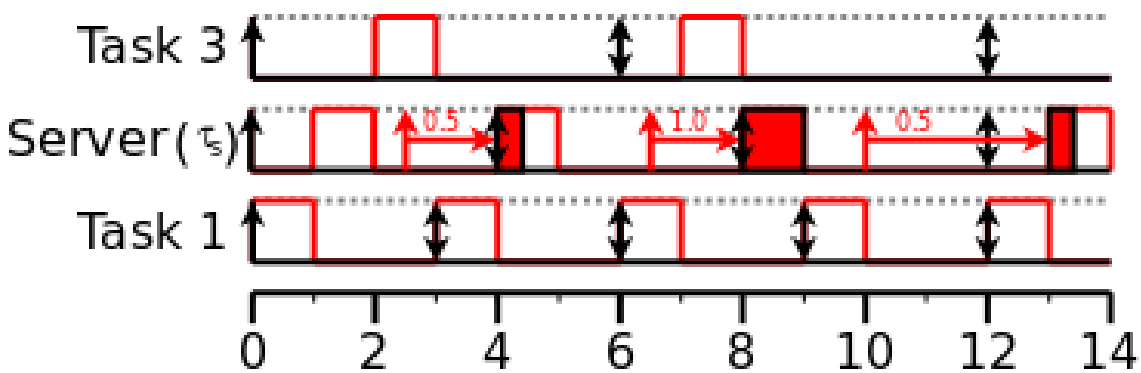
Background execution

- 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 ISR, non-preemptive system calls, 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 the lowest priority task.
- The performance is poor for real-time aperiodic 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** TS and a **capacity** CS.
- 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

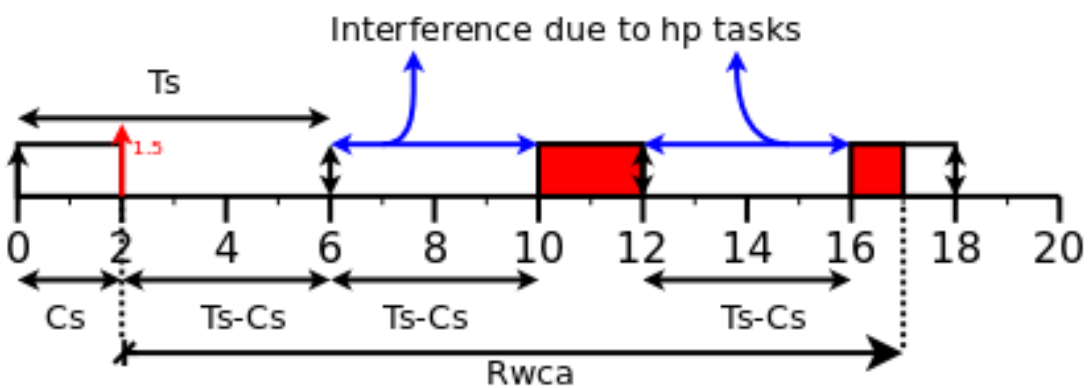
Worst-case response time upper bound:

- 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 $\tau_s(C_S, T_S)$
  - Suffers maximum jitter on the instant of the aperiodic request
  - Suffers maximum delay in all successive instances

WCRT of an aperiodic request of task τ_i

$$Rwc_i^a = Ca_i + (T_s - C_s) \cdot (1 + \lceil \frac{Ca_i}{C_s} \rceil)$$

where Ca_i is the worst-case execution time of aperiodic task i



Worst-case response time to aperiodic requests

Considering several aperiodic requests:

- If there are several aperiodic requests queued for the same server i (Na_i), sorted by a suitable criteria, the worst-case response time is:

WCRT of a set of aperiodic requests Ca_k to a server

$$\forall i = 1..Na, Rwc_i^a = \sum_{k=1}^{i}(Ca_k) + (T_s - C_s) \cdot (1 + \lceil \frac{\sum_{k=1}^{i} Ca_k}{C_s} \rceil)$$

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

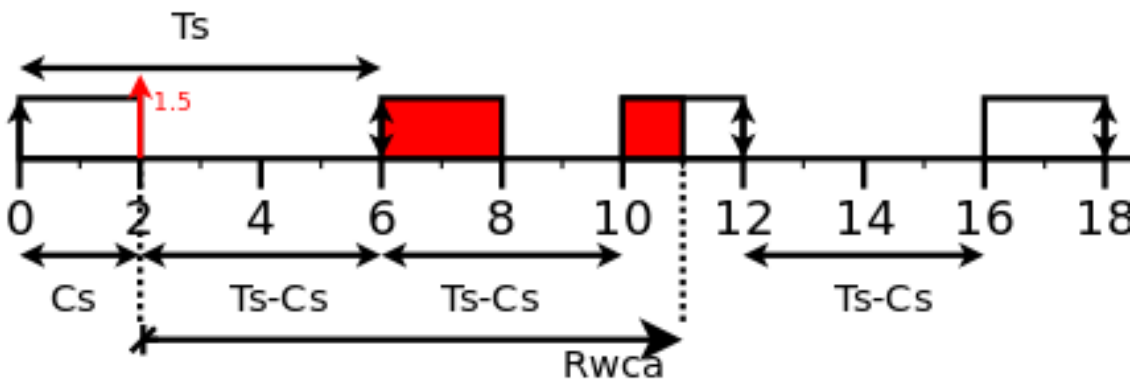
Worst-case response time to aperiodic requests

If the server has the **highest priority** :

- 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:

WCRT of a set of aperiodic requests Ca_k to a server

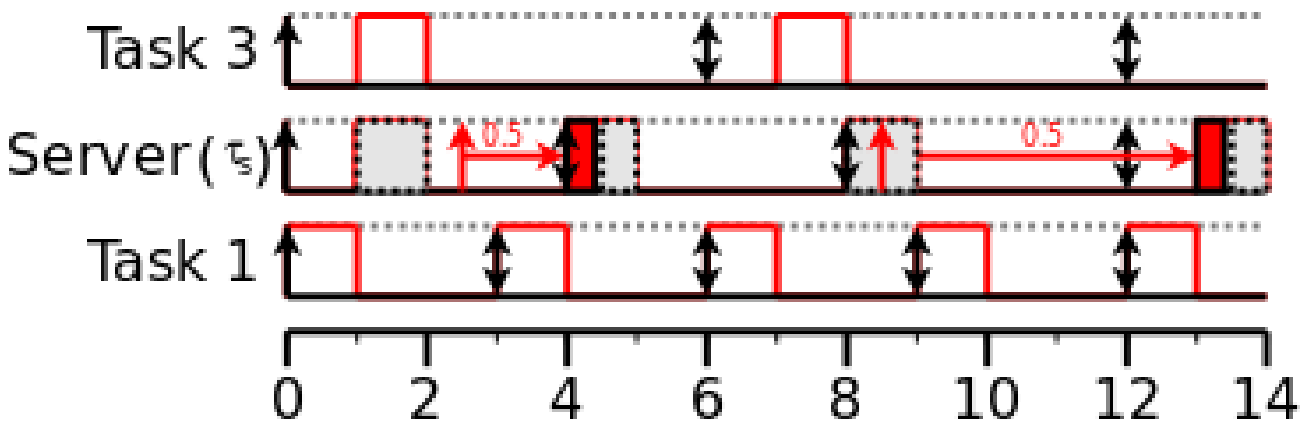
$$Rwc_i^a = Ca_i + (T_s - C_s) \cdot \lceil \frac{Ca_i}{T_s} \rceil$$



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.

E.g. Polling server with $(C, T) = (1, 4)$



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.

Utilization test for RM + PS

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

Up: utilization of n periodic tasks

Polling server (PS)

Tighter tests

- 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).
- Let U_p and U_s be the utilization factors of the periodic task set and polling server, resp.

RM + PS

$$U_p \leq n \cdot [(\frac{2}{U_s+1})^{\frac{1}{n}} - 1]$$

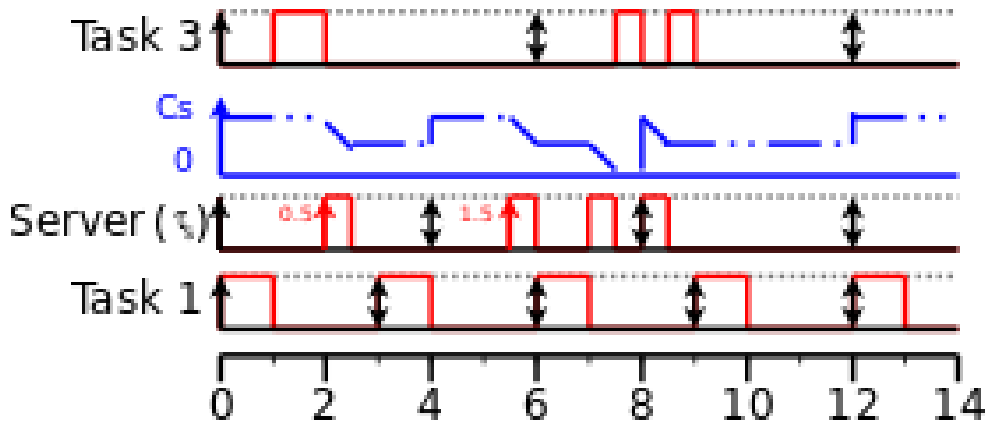
RM + PS, Hyperbolic Bound

$$\prod_{i=1}^{n} (U_i + 1) \leq \frac{2}{U_s+1}$$

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 (TS) or
  - Its capacity (CS) gets exhausted
- The capacity is replenished at the beginning of each period.

E.g. Deferrable server with $(C, T) = (1, 4)$



Deferrable server (DS)

+ Simple implementation (similar to 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).

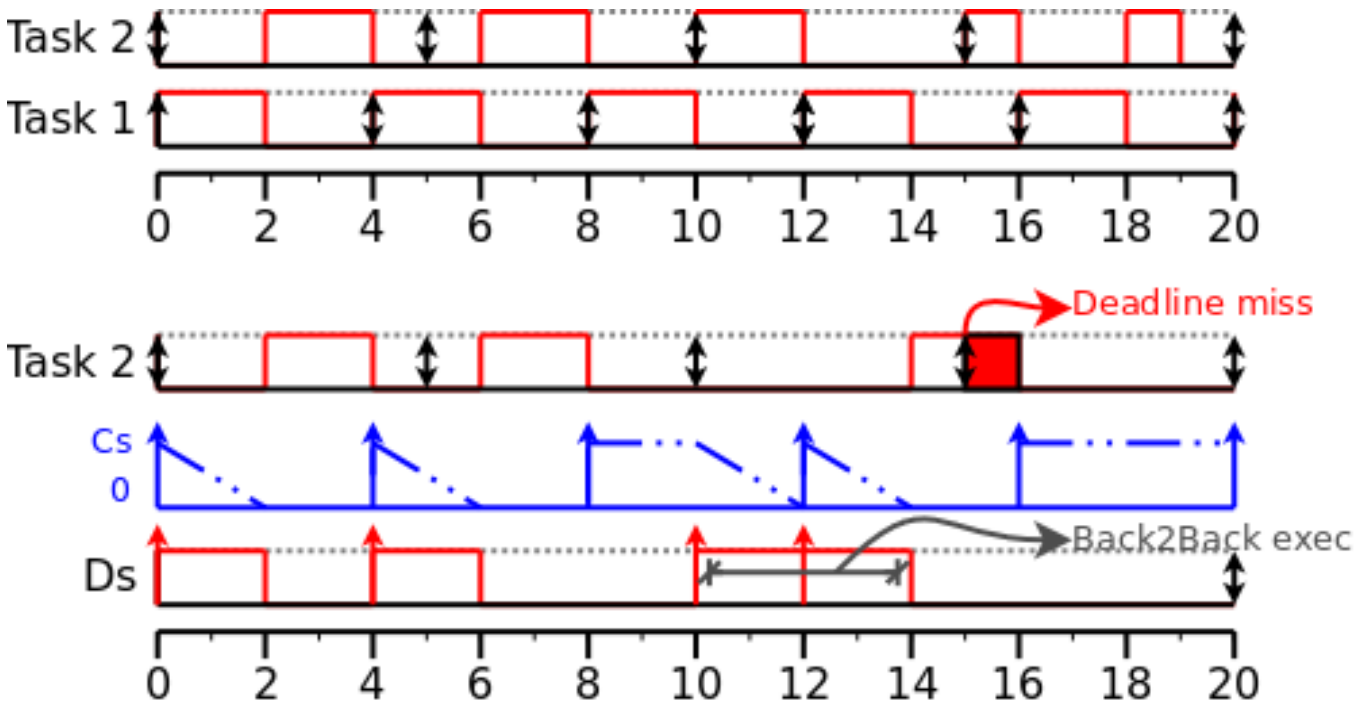
RM + DS: Least Upper Bound

$$U_{lub} = U_s + n \cdot [(\frac{U_s+2}{2 \cdot U_s+1})^{\frac{1}{n}} - 1]$$

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) [Sprunt, Sha and Lehoczky, 89]

- 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 (as the PS).
- 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** .
- Definitions

  SS active : the server or $hep(SS)$ tasks are executing
  SS idle : processor idle or $lp(SS)$ task executing
  RT/RA : replenishing time/amount
- Replenishment rules:
  - RT is set when SS becomes active and $C_s > 0$ (t_A)
  - RA is computed when the SS becomes idle or C_s is exhausted (t_I). RA is the capacity used in the interval $[t_a, t_I]$

Sporadic server (SS) - Illustration

	C_i	T_i
τ_1	1	5
τ_2	4	15

Server

$C_s = 5$

$T_s = 10$

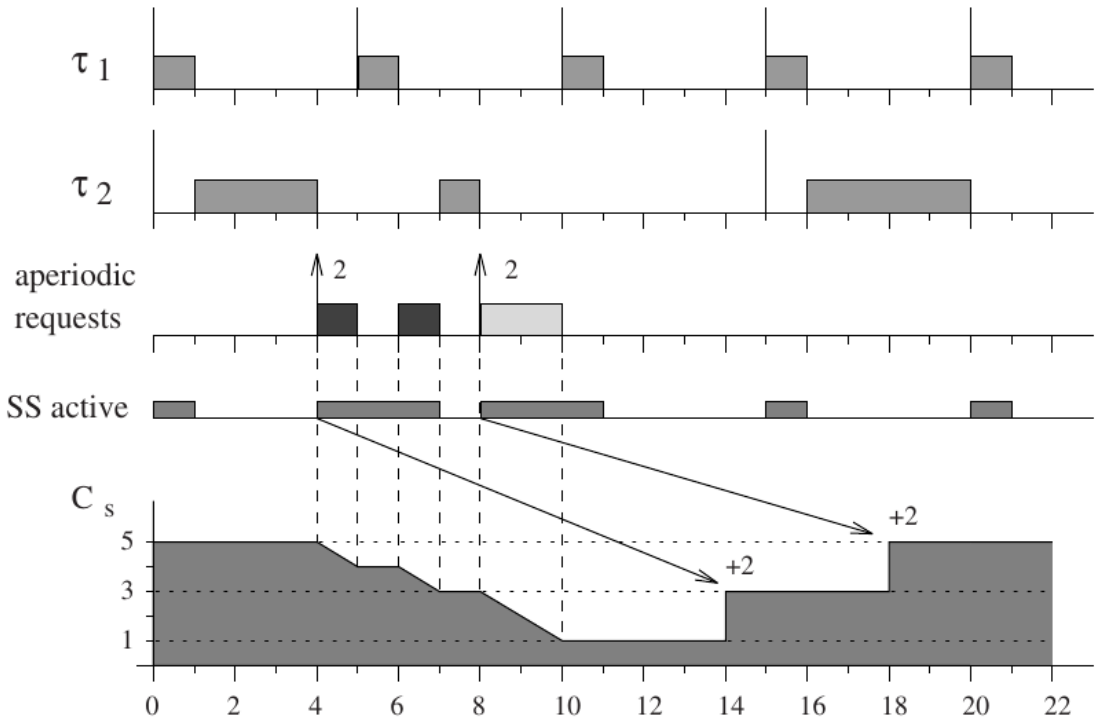


Figure taken from "HRTS, G. C. Buttazzo, 3rd edition"

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).

RM + SS utilization tests

$$U_p \leq n \cdot [(\frac{2}{U_s+1})^{\frac{1}{n}} - 1] \; ; \; \prod_{i=1}^{n}(U_i + 1) \leq \frac{2}{U_s+1}$$

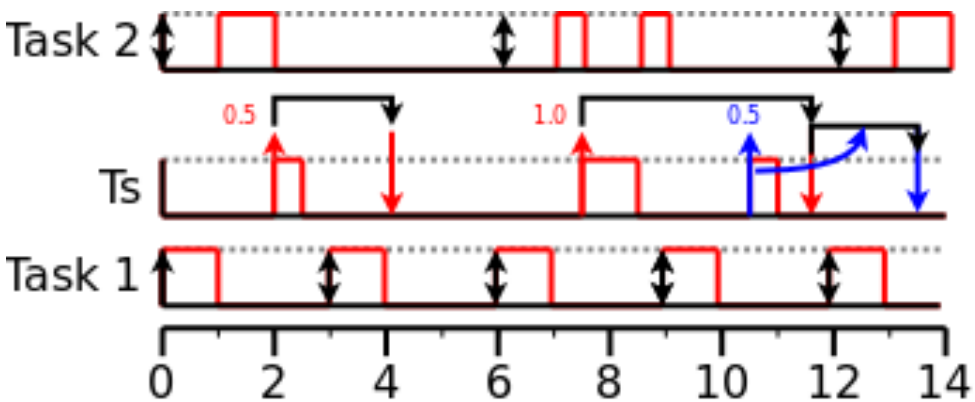
Total Bandwidth Server (TBS)

- 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,

TBS - deadline computation

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

E.g. for $U_s = 25\%$



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:
  - Simple test: $U_P + U_S \leq 1$
- **Requires a priori knowledge of Ck 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 created to **solve the robustness problem of TBS** , enforcing bandwidth isolation.
- This goal is achieved by managing the execution time based on a budget/capacity (QS ,TS) scheme.
  - When an aperiodic request r_k arrives, it is computed a server deadline d_s, as follows:

$$\boxed{\textit{If } r_k + \frac{c_s}{U_s} < d_s^{actual}, \textit{ then } d_s^{actual} \textit{ does not change.}}$$

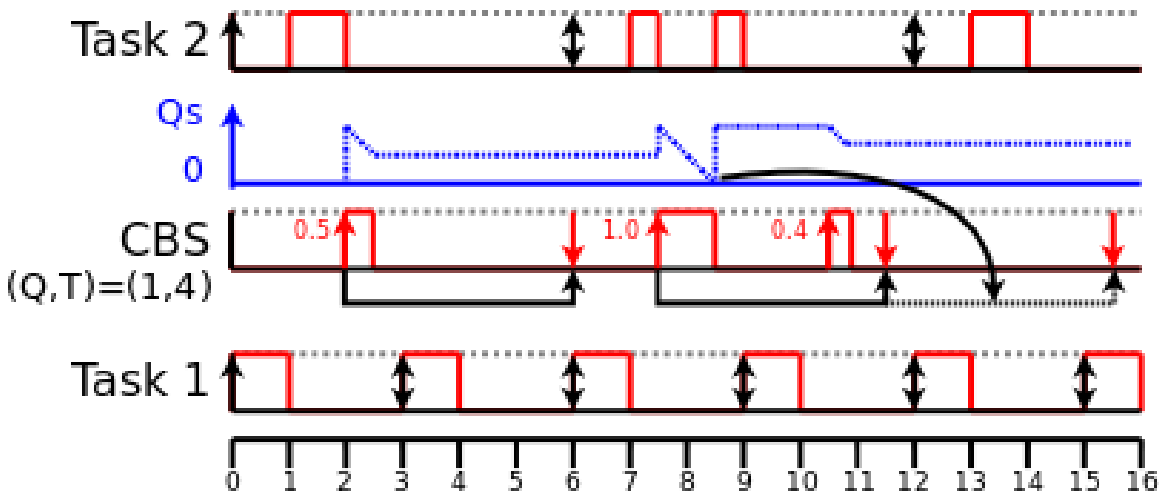
$$\boxed{\textit{Otherwise, } d_s = r_k + T_s \textit{ and } c_s = Q_s}$$

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

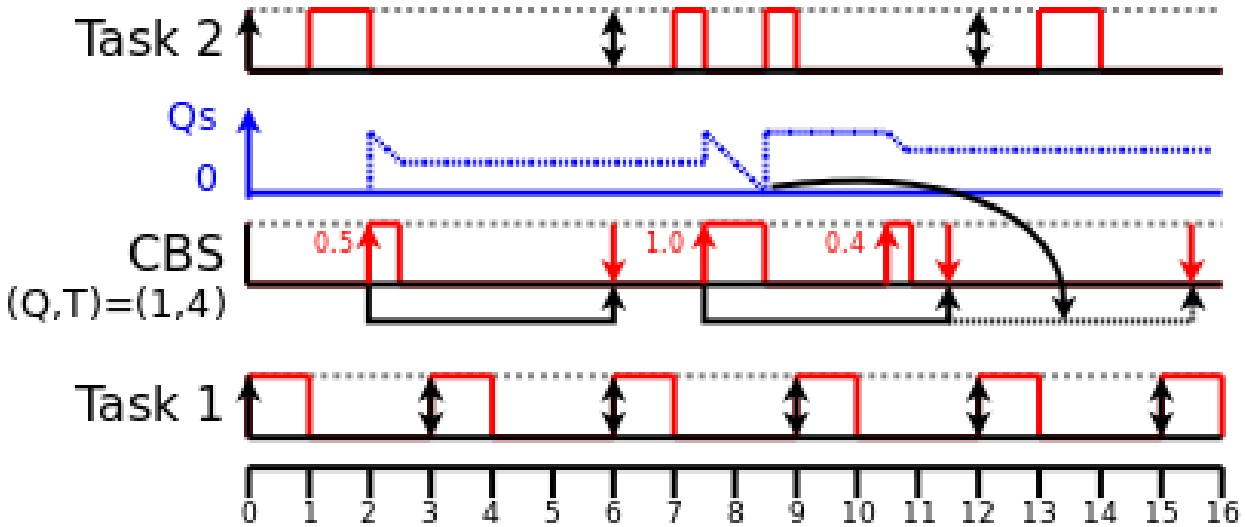
$$\boxed{d_s = d_s + T_s \textit{ and } c_s = Q_s}$$

Constant Bandwidth Server (CBS)

- The CBS assigns deadlines in such a way that prevents the **bandwidth given to the server** from being **higher than 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.



Constant Bandwidth Server (CBS)



Rules

[Arrival]

- If $r_k + \frac{c_s}{U_s} < d_s^{actual}$, then d_s^{actual} doesn't change [R1].
- Otherwise, $d_s = r_k + T_s$ and $c_s = Q_s$ [R2]

[c_s exhausted]

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

- $t = 2.0$: $d_s^{actual} < r_k$, thus R2 applies
  $d_s = r_k + T_s = 2.0 + 4 = 6.0; c_s = 1$
- $t = 7.5$: $d_s^{actual} < r_k$, thus R2 applies
  $d_s = r_k + T_s = 7.5 + 4 = 11.5; cs = 1$
- $t = 8.5$: cs exhausted, thus R3 applies
  $d_s = d_s + T_s = 11.5 + 4 = 15.5; cs = 1$
- $t = 10.5$:
  $r_k + \frac{c_s}{U_s} = 10.5 + \frac{1}{0.25} = 14.5 < d_s^{actual}$, thus R1 applies: d_s^{actual} does not change

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:

$$\boxed{U_p + U_s \leq 1}$$

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 $\Delta t \cdot U_s$ CPU time.
- Any task $\tau_i(C_i, T_i)$ schedulable with EDF is also schedulable by a CBS server with $Q_s = C_i$ and $T_s = T_i$
- A CBS may be used to:
  - Protect the system from possible overruns in any task
  - Guarantee a minimum service to soft real-time tasks
  - Reserve bandwidth to any activity

Summary

- 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