Scheduling hybrid task sets

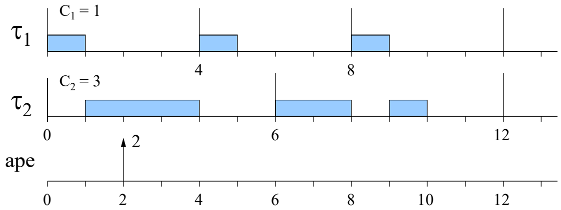
Periodic tasks
+
Aperiodic tasks

Handling aperiodic tasks

- Aperiodic tasks are typically activated by the arrival of external events (notified by interrupts).
- From one hand, one objective of the kernel is to reduce the response time of aperiodic tasks (interrupt latency).
- On the other hand, aperiodic task execution should not jeopardize schedulability.

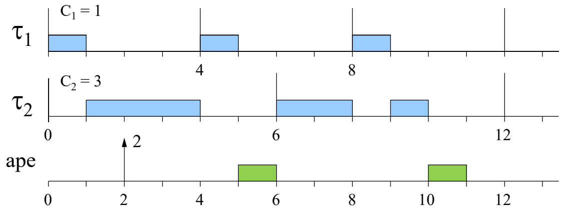
Aperiodic Scheduling

Consider a simple example with 2 periodic tasks (scheduled by RM) and a single aperiodic job with $C_a = 2$ arriving at time t = 2:



Background service

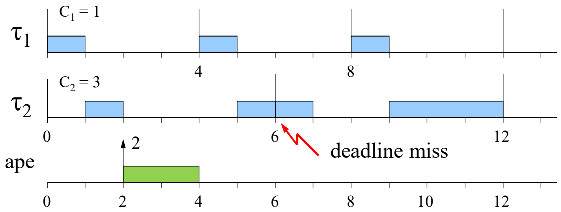
If aperiodic jobs are scheduled in background (i.e., during idle times left by periodic tasks) their response times are too long:



Response Time = 9

Immediate service

On the other hand, if interrupts service routines are scheduled at the highest priority, the other tasks can miss their deadlines:



Response Time = 2

HARD aperiodic tasks

- Aperiodic tasks with **HARD** deadlines must be guaranteed under worst-case conditions.
- Off-line guarantee is only possible if we can bound interarrival times (**sporadic tasks**).
- Hence **sporadic tasks** can be guaranteed as periodic tasks with $C_i = WCET_i$ and $T_i = MIT_i$

WCET = Worst-Case Execution Time
MIT = Minimum Interarrival Time

SOFT aperiodic tasks

- Aperiodic tasks with **SOFT** deadlines should be executed as soon as possible, but without jeopardizing HARD tasks.
- We may be interested in
  - → minimizing the response time of each aperiodic request
  - → performing an on-line guarantee

How can we achieve these goals?

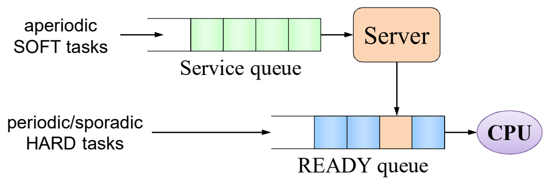
Aperiodic Servers

- A server is a kernel activity aimed at controlling the execution of aperiodic tasks.
- Normally, a server is a periodic task having two parameters:

C_s capacity (or budget)
T_s server period

To preserve periodic tasks, no more than C_s units must be executed every period T_s

Aperiodic service queue



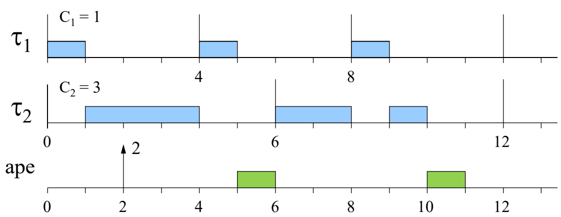
- The server is scheduled as any periodic task.
- Priority ties are broken in favor of the server.
- Aperiodic tasks can be selected using an arbitrary queueing discipline.

Polling Server (PS)

- At the beginning of each period, the budget is recharged at its maximum value.
- Budget is consumed during job execution.
- When the server becomes active and there are no pending jobs, C_s is discharged to zero.
- When the server becomes active and there are pending jobs, they are served until $C_s > 0$.

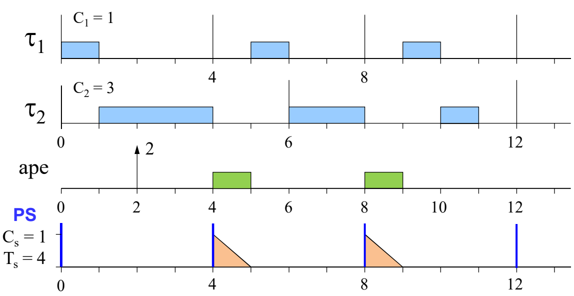
Background service

Let's take the previous example:



Response Time = 9

RM + Polling Server



Response Time = 7

PS properties

- In the worst-case, the PS behaves as a periodic task with utilization $U_s = C_s/T_s$
- Aperiodic tasks execute at the highest priority if $T_s = \min(T_1, \ldots, T_n)$.
- Liu & Layland analysis gives that:

$$U_{\text{lub}}^{RM+PS}(n) = U_s + n\left[\left(\frac{2}{U_s+1}\right)^{1/n} - 1\right]$$

Analysis with Hyperbolic Bound

A set of periodic tasks is schedulable by Rate Monotonic in the presence of a Polling Server with utilization U_s if

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

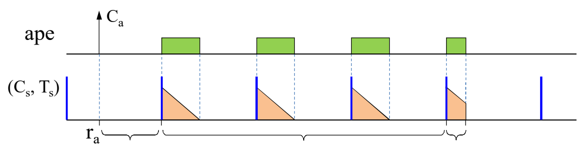
Defining $P = \prod_{i=1}^{n}(U_i+1)$

the maximum server utilization that guarantees the schedulability of the periodic task set is

$$U_s^{\max} = \frac{2}{P} - 1$$

Response time under PS

Consider a PS running at the highest priority and an aperiodic job arriving when the server is idle:



initial delay: $\Delta_a = \left\lceil \frac{r_a}{T_s} \right\rceil T_s - r_a$

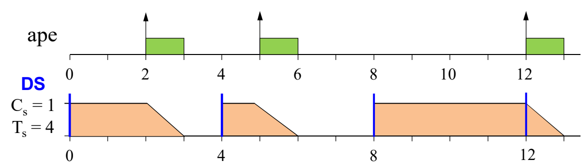
full service periods: $F_a = \left\lceil \frac{C_a}{C_s} \right\rceil - 1$

final chunk: $\delta_a = C_a - F_a C_s$

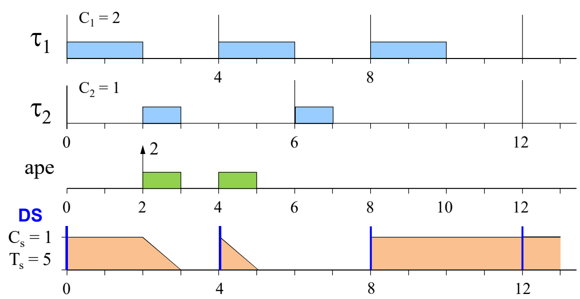
$$R_a = \Delta_a + C_a + F_a(T_s - C_s)$$

Deferrable Server (DS)

- Is similar to the PS, but the budget is not discharged if there are no pending requests.
- Keeping the budget improves responsiveness, since jobs can be served within a period.



RM + Deferrable Server

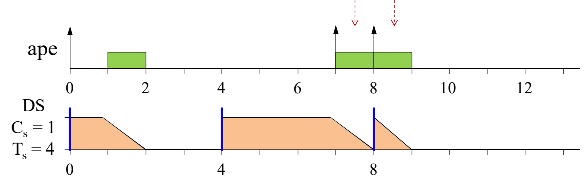


Response Time = 3

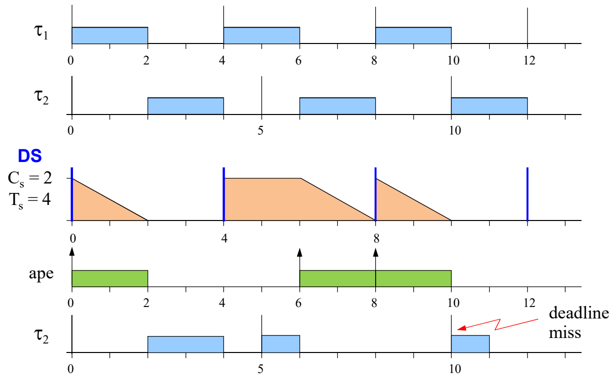
Deferrable Server (DS)

- However, DS does not behaves like a periodic task and it is more invasive than PS.
- Keeping the budget decreases the utilization bound.

There can be two server executions close to each other



Deferrable Server (DS)



Analysis of RM + DS

- In the worst-case, the DS is more invasive than a periodic task with utilization $U_s = C_s/T_s$
- Liu & Layland analysis gives that:

$$U_{lub}^{RM+DS}(n) = U_s + n\left[\left(\frac{U_s+2}{2U_s+1}\right)^{1/n} - 1\right]$$

Analysis with Hyperbolic Bound

A set of periodic tasks is schedulable by Rate Monotonic in the presence of a Deferrable Server with utilization U_s if

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

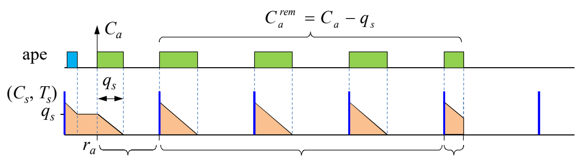
Defining $P = \prod_{i=1}^{n}(U_i+1)$

the maximum server utilization that guarantees the schedulability of the periodic task set is

$$U_s^{max} = \frac{2-P}{2P-1}$$

Response time under DS

Consider a DS running at the highest priority and an aperiodic job arriving when the server is idle:



initial delay: $\Delta_a = \left\lceil \frac{r_a}{T_s} \right\rceil T_s - r_a$

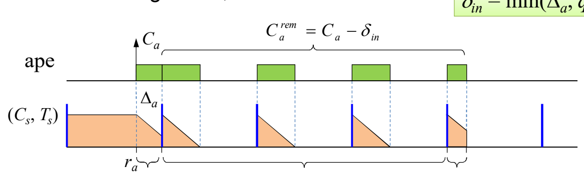
full service periods: $F_a = \left\lceil \frac{C_a^{rem}}{C_s} \right\rceil - 1$

final chunk: $\delta_a = C_a^{rem} - F_a C_s$

$$R_a = \Delta_a + F_a T_s + \delta_a$$

Response time under DS

If a task arrives close to the next server period, a value $\Delta_a < C_s$ is executed. In general, the initial execution is: $\delta_{in} = \min(\Delta_a, q_s)$



initial delay: $\Delta_a = \left\lceil \frac{r_a}{T_s} \right\rceil T_s - r_a$

full service periods: $F_a = \left\lceil \frac{C_a^{rem}}{C_s} \right\rceil - 1$

final chunk: $\delta_a = C_a^{rem} - F_a C_s$

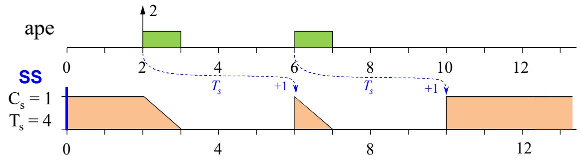
$$R_a = \Delta_a + F_a T_s + \delta_a$$

Designing server parameters

1. Determine U_s^{max} using $\prod_{i=1}^{n}(U_i+1) \le K_{server}$
2. Define $U_s \le U_s^{max}$
3. Define $T_s = \min(T_1, \ldots, T_n)$
4. Compute $C_s = U_s T_s$

Sporadic Server (SS)

- It preserves the budget like DS, but it is less aggressive than DS, since the budget is replenished only T_s units after its consumption.
- SS is not activated periodically, but from the analysis point of view it behaves like a period task with computation time C_s and period T_s.



Sporadic Server rules

Assumptions: $\begin{cases} q_s \text{ = current server budget} \\ SS \text{ has the highest priority: } T_s \le \min(T_1, \ldots, T_n) \end{cases}$

Rule 1

At time t_A, at which the following event occurs:

$(q_s > 0)$ AND ($\exists$ pending aperiodic requests)

set the replenishment time in the future at time $RT = t_A + T_s$

Rule 2

At time t_I, at which the following event occurs:

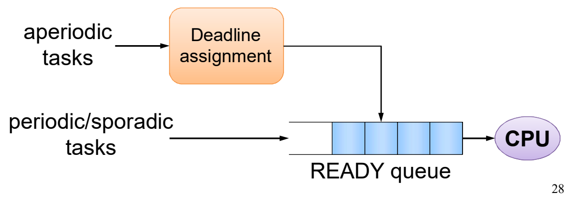
$(q_s \le 0)$ OR ($\nexists$ pending aperiodic requests)

set the replenishment amount equal to the budget $C_{ape}(t_A, t_I)$ consumed in the interval $[t_A, t_I]$.

Dynamic Priority Servers

Total Bandwidth Server (TBS)

- It is a dynamic priority server, used with EDF.
- Aperiodic jobs are assigned a deadline so that the server does not exceed a given bandwidth U_s.
- Aperiodic jobs are inserted in the ready queue and scheduled together with the HARD tasks.



Deadline assignment rule

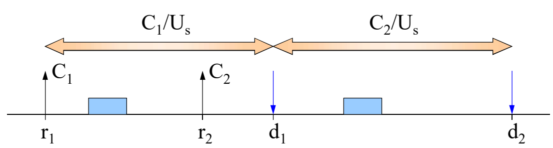
- Deadline has to be assigned not to jeopardize periodic tasks.
- A safe relative deadline is equal to the minimum period that can be assigned to a new periodic task with utilization U_s:

$$U_s = C_k / T_k \Rightarrow T_k = d_k - r_k = C_k / U_s$$

- Hence, the absolute deadline can be set as:

$$d_k = r_k + C_k / U_s$$

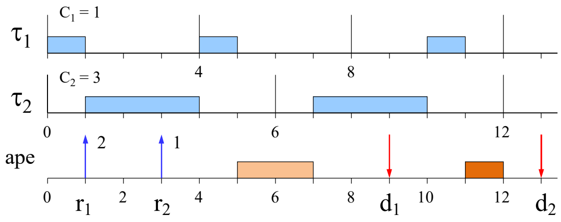
Deadline assignment rule



- To keep track of the bandwidth assigned to previous jobs, d_k must be computed as:

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

EDF + TBS schedule

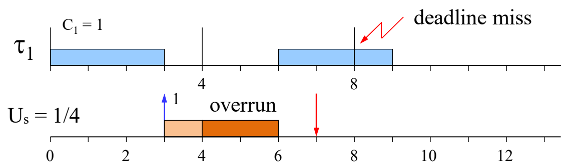


$U_s = 1 - U_p = 1/4$

$$\begin{cases} d_1 = r_1 + C_1 / U_s = 1 + 2 \cdot 4 = 9 \\ d_2 = \max(r_2, d_1) + C_2 / U_s = 9 + 1 \cdot 4 = 13 \end{cases}$$

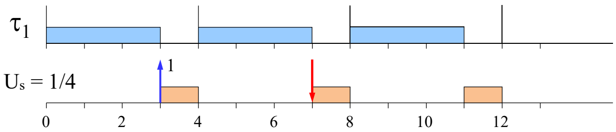
Problems with the TBS

- Without a budget management, there is no protection against execution overruns.
- If a job executes more than expected, hard tasks could miss their deadlines.



Overrun handling

- If a job executes more than expected (i.e., consumes its budget) it must be delayed by decreasing its priority or postponing its deadline.

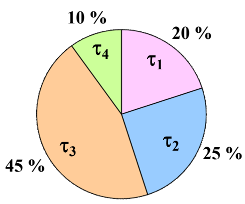


Solution: task isolation

- In the presence of overruns, only the faulty task should be delayed.
- Each task τ_i should not consume more than its declared utilization ($U_i = C_i/T_i$).
- If a task executes more than expected, its priority should be decreased (or its deadline postponed).

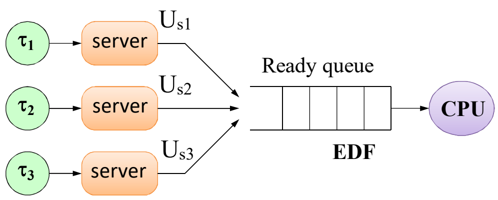
Achieving isolation

- Isolation among tasks can be achieved through a **bandwidth reservation**.
- Each task is managed by a dedicated server having bandwidth U_s



- The server assigns priorities (or deadlines) to tasks so that they do not exceed the reserved bandwidth.

Implementation



$U_{s1} + U_{s2} + U_{s3} \leq 1$

Constant Bandwidth Server

- It assigns deadlines to tasks as the TBS, but keeps track of job executions through a budget mechanism.
- When the budget is exhausted it is immediately replenished, but the deadline is postponed to keep the demand constant.

CBS parameters

Maximum budget:	Q_s		assigned by the user
Server period:	T_s		
Server bandwidth:	$U_s = Q_s/T_s$		
Current budget:	q_s	(initialized to 0)	maintained by the server
Server deadline:	d_s	(initialized to 0)	

Basic CBS rules

Arrival of job J_k at time r_k $\Rightarrow$ assign d_s

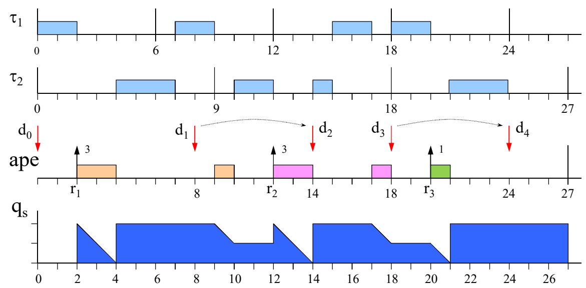
if (∃ pending ape. requests) **then** <enqueue J_k>

else if $(q_s > (d_s - r_k)U_s)$ **then** $\begin{cases} q_s = Q_s \\ d_s = r_k + T_s \end{cases}$

Budget exhausted $\Rightarrow$ postpone d_s

$$\begin{cases} q_s = Q_s \\ d_s = d_s + T_s \end{cases}$$

EDF + CBS schedule



CBS: $Q_s = 2$, $T_s = 6$