Scheduling of Aperiodic and Sporadic Jobs in Priority-Driven Systems
- Chapter 7 –

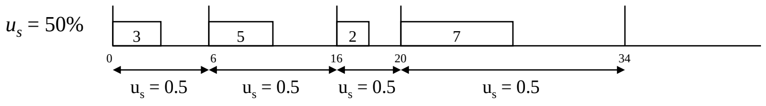
(Dynamic Priority Framework)

Dynamic priority servers

- We focused on fixed priority servers so far. Such servers can be used along with RM.
- Now, we will focus on dynamic priority servers. They are used along with EDF

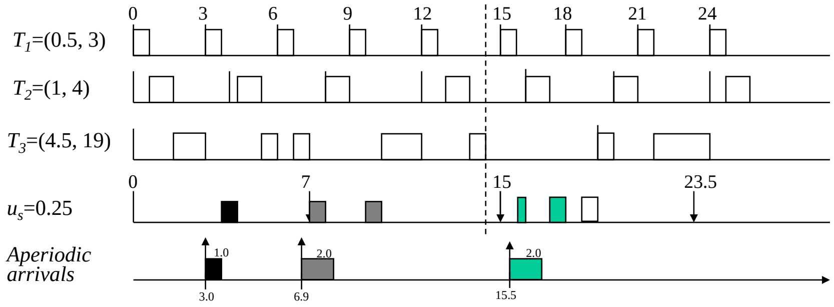
Constant Utilization Server - 1

- The key idea: a server is given budget u_s. Upon the aperiodic request of e, set the server budget $e_s = e$ and adjust the relative deadline D_s such that $e_s/D_s = u_s$, i.e., $D_s = e_s/u_s$.



- Rule 1: initialization: $e_s = 0$; and $d_s = 0$;
- Rule 2: when an aperiodic request e arrives at time t,
  - A) If the queue is not empty, join the queue
  - B) else if $t < d_s$ (d_s is the current server deadline), do nothing (just join the queue)
  - else $d_s = t + e/u_s$; $e_s = e$;
- Rule 3: when t becomes equal to d_s
  - A) If the queue is not empty, serve the head of the queue: $d_s = d_s + e/u_s$; $e_s = e$.
  - B) else do nothing.

Constant Utilization Server - 2



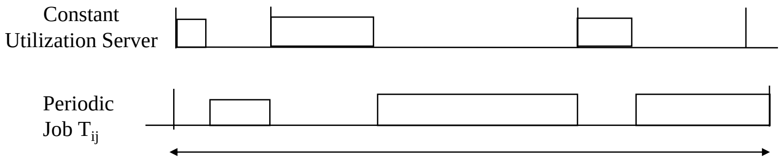
- At $t = 3$, A_1 arrives, e_s = ___ , d = _____
- At $t = 3$, A_1 arrives, $e_s = 1$, $d = 3+1.0/0.25 = 7$

- At $t = 6.9$, A_2 arrives, what should we do?
- At t = ___, e_s = _____, d = ______
- At t = 7, e_s = 2.0, d = 7 + 2.0/0.25= 15

- At t = 15, do nothing. Why? At t= 15.5, e_s = 2.0, d = $\underline{15.5}$ + 2.0/0.25 = 23.5

Constant Utilization Server - 3

- Giving a set of periodic tasks and a set of CUSs, if the total utilization is no more than 1, the system is schedulable.
- How can we prove the above?
  - What is the closest result that we know? How did we do it? Can we reuse the method directly? Can we convert this new problem into the old form by transformations?



Which Part of this can be Reused?

What needs to be modified in this figure?…

Proof by contradiction: We assume that the total utilization is no more than 1 but J_{22} misses its deadline.

Computational demand of T_1 in [0, t]

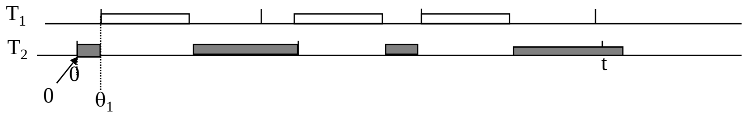
We sum up J_{22} and all the jobs completed before J_{22}. If J_{22} misses its deadline t, the sum must be greater than t

$$\left\lfloor \frac{t-\theta_1}{p_1} \right\rfloor e_1 + \frac{t}{p_2} e_2 > t$$

$$\left\lfloor \frac{t}{p_1} \right\rfloor e_1 + \frac{t}{p_2} e_2 > t$$

$$\frac{t}{p_1} e_1 + \frac{t}{p_2} e_2 > t$$

$$\frac{e_1}{p_1} + \frac{e_2}{p_2} > 1$$



CUS: Counting the Demands

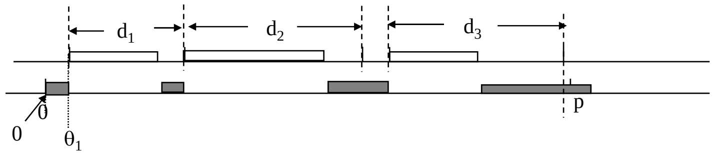
Quiz: what is the assumption to be contradicted? can you explain what is going on here?

$$d_1 \times u_s + d_2 \times u_s + d_3 \times u_s + e > p$$

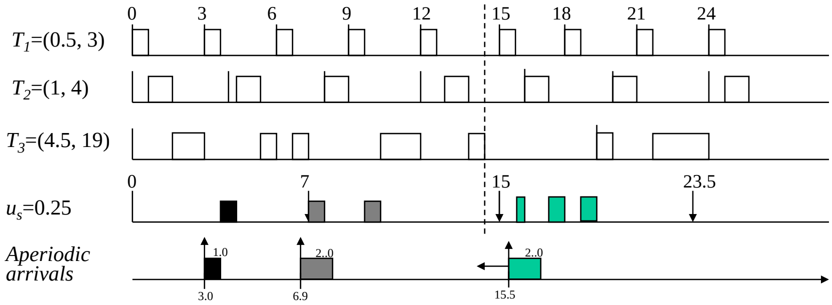
$$(d_1 + d_2 + d_3) \times u_s + e > p$$

$$\frac{(d_1 + d_2 + d_3)}{p} u_s + \frac{e}{p} > 1$$

$$u_s + \frac{e}{p} > 1$$



TBS: Get it done earlier



- There is an idle time between 14 and 15. Suppose that A_3 arrives at 14 instead. CUS will not do anything until 15 and the CPU still idles, since the CUS already used its share within [7, 15].

- One would like to use such idle time to execute earlier and thus achieving a better response time.

Total Bandwidth Server - 1

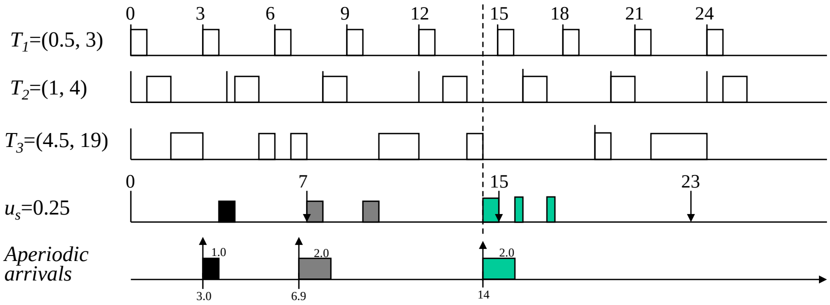
- The key idea: same to CUS except it is possible to have an earlier replenishment of the budget. The sequence of deadlines is the same.

- Rule 1: initialization: e_s = 0; and d_s = 0;
- Rule 2: when an aperiodic request *e* arrives at time *t*,
  - A) If the queue is not empty, join the queue
  - B) else $d_s = max(t, d_s) + e/u_s$; $e_s = e$ *and the aperiodic request is ready to execute*;
- Rule 3: at the completion of executing aperiodic request
  - A) If the queue is not empty, serve the head of the queue: $d_s = d_s + e/u_s$; $e_s = e$.
  - B) else do nothing.

- In short, when the *k*th aperiodic request arrives at time $t = r_k$, it receives a deadline:

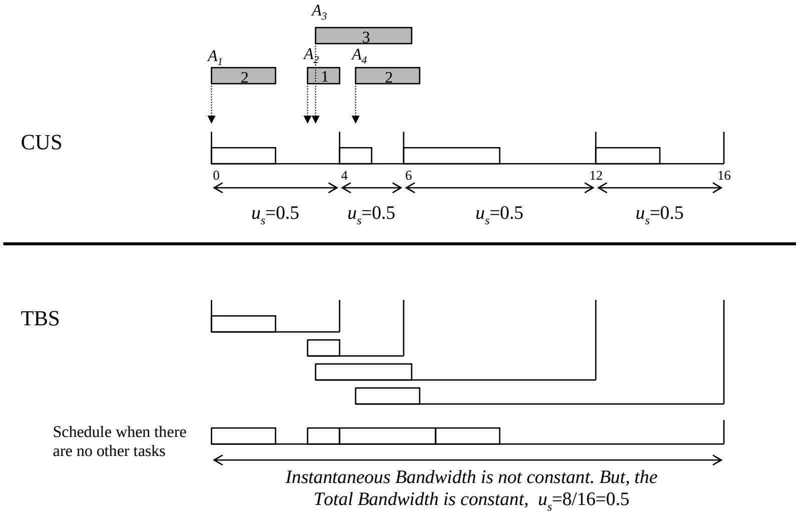
$$d_k = \max(r_k, d_{k-1}) + \frac{e_k}{u_s}$$

TBS: An Incremental Improvement



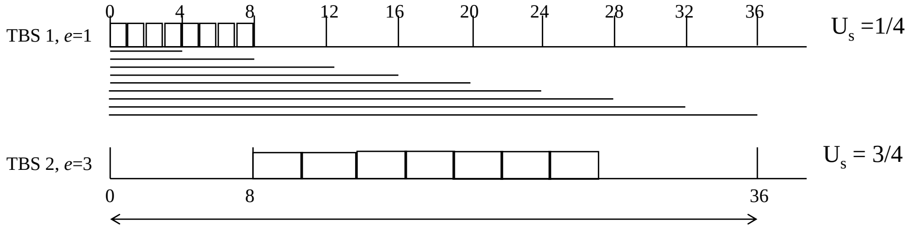
- At t = 3, A_1 arrives, e_s = 1 , d = 3+1.0/0.25 = 7
- At t = 6.9, A_2 arrives, e_s = 2.0, d = max(6.9, 7) + 2.0/0.25= 15
- At t = 14, A_3 arrives, e_s = 2.0, d = max(14,15) + 2.0/0.25 = 23
- Compared with CUS, the deadline setting is IDENTICAL. However, the budget replenishing time can be different. Consider A_3, under CUS, it will be 15, but it is moved to 14 under TBS.
- Quiz: will TBS move cause tasks missing deadlines if CUS does not? (see Textbook)

CUS vs. TBS



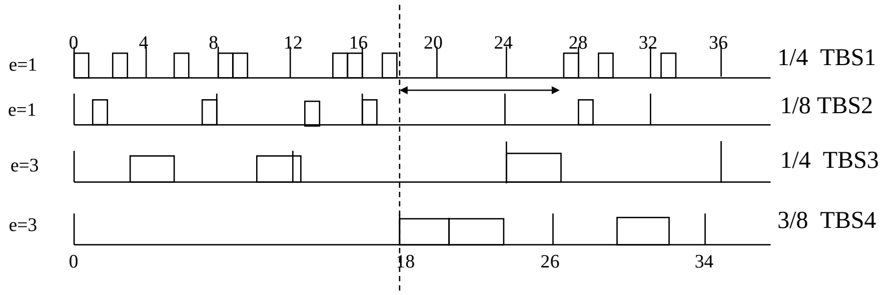
Instantaneous Bandwidth is not constant. But, the Total Bandwidth is constant, u_s=8/16=0.5

Feast and Famine Under TBS



- TBS 2 has no arrival during [0,8] and TBS 1 is backlogged since 0 and frontloads the CPU completely. At 8, the deadline of
  - 9th request to TBS 1 has deadline at 9*4 = 36.
  - 1st request of TBS 2 has deadline at 8+3/(3/4) =12 , 2nd at 16, 3rd at 20, 4th at 24 …
  - So [8, 36] is the “starvation” time for TBS 1 requests
- Fairness definition: *each backlogged server over any given time interval should get fractions of CPU cycles proportional to the server size.*

TBS “Starvation”

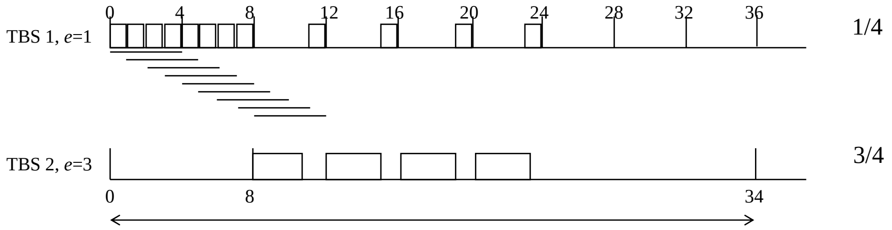


- Key Observation: TBS moves release time forward but deadlines are unchanged. So it is front loaded. Later on, with deadlines further and further away, it cries unfair…
- There is no arrival in TBS4 during [0, 18]. Starting at 18, the deadline of
  - The 9th job of TBS1 has deadline at 4*9= 36
  - The 5th job of TBS2 has deadline at 5*8= 40
  - The 3rd job of TBS3 has deadline at 3*12=36
  - The 1st job of TBS4 has deadline at d = 18+3/(3/8)=26 and 2nd job is at 26+3/(3/8)=34
  - During [18, 27], TBS1 does not have any share of CPU…

A Quick Fix

- TBS moves release times forward but deadlines are unchanged. So it is front loaded. Later on, with deadlines further and further away, it cries unfair…
- CUS does not frontload its executions and hence does not have the same problem. But it cannot use any leftover CPU cycle. So the quick fix is:
- A quick fix is therefore
  - to use TBS servers.
  - When CPU idles, restart all the backlogged TBS servers. By restarting, we mean that we shall erase the histories and view the system as if it starts from t = 0 again.
  - Which lemma did we learn that allows us to do this?

TBS/Restart when idle



- TBS 2 has no arrival during [0,8] and TBS 1 is backlogged since 0 and frontloads the CPU completely.
  - At 1, processor idles and we reset. So the deadline is at 5 instead of 8!
  - At 8, processor idles and we reset. So the deadline is at 12.
  - At 8, TBS2 is backlogged with deadline at 8+3/(3/4) = 12, 2nd at 16, 3rd at 20,…
  - After 8, the deadlines of TBS1 are at 12, 16, 20, 24….

Constant Bandwidth Server (CBS) I

- TBS and CUS assume that the computation time of the aperiodic tasks is known in order to compute the next server deadline. However, sometimes the computation time of an aperiodic task is unknown or variable (see MPEG player).

- Quiz: What does happen when an aperiodic task is served by a TBS/CUS and it does not terminate by its declared execution time?

Constant Bandwidth Server (CBS) II

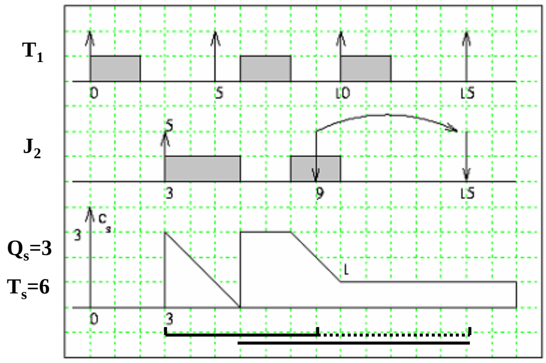
- TBS and CUS assume that the computation time of the aperiodic tasks is known in order to compute the next server deadline. However, sometimes the computation time of an aperiodic task is unknown or variable (see MPEG player).
- Quiz: What does happen when an aperiodic task is served by a TBS/CUS and it does not terminate by its declared execution time?
- It depends on the server implementation: according to the server definition, we do not need to monitor the server budget when scheduling the aperiodic requests. In fact, tasks are scheduled according to their absolute deadline (EDF).
- If an aperiodic overruns, it can cause a hard periodic task to miss its deadline.
- To overcome these problems, the CBS has been introduced

Constant Bandwidth Server (CBS) III

- The CBS is similar to the TBS server but it does not need the aperiodic task computation time in order to compute the server deadline.
- **The execution of the aperiodic tasks is continuously monitored** by means of the *current budget* c_s: it represents the remaining available computation time for the *current server deadline* d_s.
- Q_s is the maximum budget; P_s is the server period. The ratio $U_s = Q_s / P_s$ is the server bandwidth.

Constant Bandwidth Server (CBS) IV

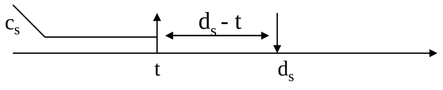
- The key idea:
  - if an aperiodic arrives at time t, its available budget is $c_s = Q_s$ and the server deadline is $d_s = t + P_s$;
  - if the aperiodic request completes and there is some budget left, the next request can be served by using the same server deadline and the leftover budget
  - If the budget is exhausted, set $c_s = Q_s$ and $d_s = d_s + P_s$;



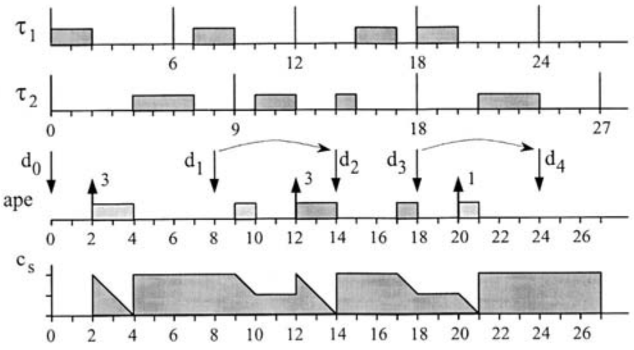
An example of a job J_2 (r_2 = 3, c_2 = 5) handled by the CBS server

Constant Bandwidth Server (CBS) V

- CBS rules:
  - if a request arrives when another aperiodic request is pending, the new request is inserted in the server queue (the server queue is sorted according to any policy);
  - Whenever the budget is exhausted ($c_s = 0$), it is always recharged up to the maximum value ($c_s = Q_s$) and the server deadline is postponed by a server period ($d_s = d_s + T_s$);
  - When a request is finished, the next request (if any) waiting in the server queue is served with the current budget c_s and server deadline d_s;
  - when a new request arrives at time t and the server is currently idle, if $c_s < (d_s - t) U_s$, the request is served by using the current budget c_s and server deadline d_s. Otherwise, the budget is recharged up to the maximum value ($c_s = Q_s$) and the server deadline is set as $d_s = t + T_s$) – remaining budget c_s can be used only if it does not badly impact other tasks during [t, d_s].



CBS example

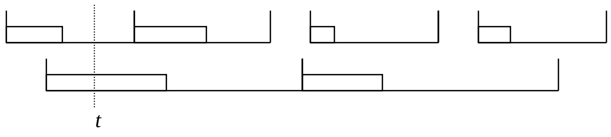


	C	p
T_1	2	6
T_2	3	9
T_s	2	6

t = 2	Request arrives (CBS idle)	$c_s = Q_s$ $d_s = t + T_s = 8$
t = 4	Budget exhausted	$c_s = Q_s$ $d_s = d_s + T_s = 14$
t = 12	Request arrives (CBS idle)	$c_s = Q_s$ $d_s = t + T_s = 18$
t = 14	Budget exhausted	$c_s = Q_s$ $d_s = d_s + T_s = 24$
t = 20	Request arrives (CBS idle)	c_s and d_s are kept unchanged
t = 21	Budget exhausted	$c_s = Q_s$ $d_s = d_s + T_s = 30$

Schedulability Analysis of Sporadic Tasks

- A sporadic task consists of a stream of jobs with variable execution times and irregular arrivals. Unlike aperiodic tasks, sporadic tasks have firm deadlines. There is usually a minimal inter-arrival time and max execution time.
- The instantaneous utilization of a sporadic job J_i is the ratio $e_i/(d_i – r_i)$. A sporadic job is said to be active within its feasible interval $[r_i, d_i]$
- **Sporadic Job Theorem**: Given a set of independent, preemptive sporadic jobs, they are schedulable by EDF if the total instantaneous utilization of all active jobs is no greater than 1 at all times t.

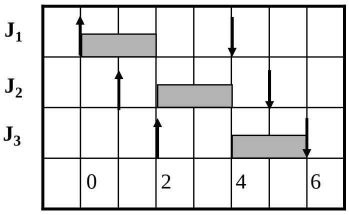


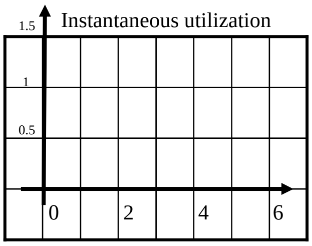
Who is interested in the proof can check the Liu book at page 219

Necessity of Sporadic Job Theorem?

- Notice that the sporadic job theorem is only a sufficient test, not necessary.

- To convince yourself: consider three sporadic jobs J_1(r = 0, e = 2, d = 4); J_2(r = 1, e = 2, d = 5); J_3(r = 2, e = 2, d = 6). The task set is schedulable, however the total instantaneous utilization of the active jobs is 1.5 in interval [2, 4].



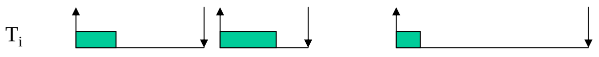


Quiz: What is the instantaneous utilization of active jobs in interval [0, 6]?

Schedulability Analysis of Sporadic Tasks
(Case 1: max instantaneous utilization is known)

- The utilization of a sporadic task T_i is the max of its jobs' instantaneous utilization:

$$\tilde{u}_i = \max_j \frac{e_{i,j}}{d_{i,j} - r_{i,j}}$$



- **Theorem**: A set of sporadic tasks using EDF is schedulable if their total utilization is no greater than 1.

- **Corollary**: A system of periodic tasks and sporadic tasks is schedulable under EDF if the total utilization is no greater than 1.

Schedulability Analysis of Sporadic Tasks

(Case 2: max instantaneous utilization is unknown)
- online “Job by job” admission -

- All these results are sufficient conditions based on worst case analysis.
- On-line admission test:
  - Find the available capacity for sporadic tasks; e.g., if periodic tasks take 70% then there is 30% available bandwidth for sporadic tasks
  - Keep track of the total instantaneous utilization of all active sporadic JOBS.
    - When a job arrives, add its instantaneous utilization to the total
    - When a job deadline is reached, subtract its instantaneous utilization from the total
    - Make sure that the total instantaneous utilization is no greater than the available bandwidth for sporadic jobs.

How to handle sporadic tasks in Fixed-Priority System?

- If min-inter-release-time (p) and max-execution-time (e) are known, reserve a portion using a sporadic server with capacity (e/p). If all periodic tasks and sporadic servers are schedulable, all sporadic jobs meet their deadlines.
- Otherwise, Online “Job-by-Job” admission is the only choice
  - On-line admission test:
    - Find the available capacity for sporadic jobs; e.g., if periodic tasks take 70% then there is 30% available bandwidth for sporadic tasks
    - Make a sporadic server with capacity 30% (e.g., $e_s = 3$, $p_s = 10$).
    - When a sporadic job J (e, d) arrives, check if the sporadic server can provide *e* time units before *d*. But, how? It maybe a problem in the final exam.