CSCE 990: *Real-Time Systems*

Mixing Real-Time and Non-Real-Time

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/RealTimeSystems

Mixing Real-Time and Non-Real-Time in Priority-Driven Systems

(Chapter 7 of Liu)

- We discussed mixing real-time and non-real-time (aperiodic) jobs in cyclic schedules
- We now address the same issue in priority-driven systems.
- We first consider two straightforward scheduling algorithms for periodic and aperiodic jobs.
- We then look at a class of algorithms called bandwidth-preserving servers that schedule aperiodic jobs in a real-time system.

Periodic and Aperiodic Tasks

(A review of the terminology Liu uses…)

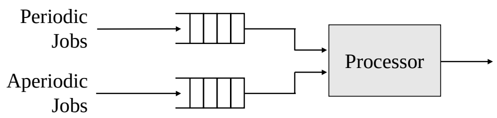
- **Periodic task:** T_i is specified by (ϕ_i, p_i, e_i, D_i).
  - » p_i is the minimum time between job releases.
- **Aperiodic tasks:** non-real-time
  - » Released at arbitrary times.
  - » Has no deadline and e_i is unspecified.
- We assume periodic and aperiodic tasks are independent of each other.

Correct and Optimal Schedules in mixed job systems (more terms…)

- A correct schedule never results in a deadline being missed by periodic tasks.
- A correct scheduling algorithm only produces correct schedules.
- An optimal aperiodic job scheduling algorithm minimizes either
  - » the response time of the aperiodic job at the head of the queue or
  - » the average response time of all aperiodic jobs.

Scheduling Mixed Jobs

- We assume there are separate job queues for real-time (periodic) and non-real-time (aperiodic) jobs.
- How do we minimize response time for aperiodic jobs without impacting periodic?



Background Scheduling

- Periodic jobs are scheduled using any priority-driven scheduling algorithm.
- Aperiodic are scheduled and executed in the background:
  - » Aperiodic jobs are executed only when there is no periodic job ready to execute.
  - » Simple to implement and always produces correct schedules.
    - The lowest priority task executes jobs from the aperiodic job queue.
  - » We can improve response times without jeopardizing deadlines by using a slack stealing algorithm to delay the execution of periodic jobs as long as possible.
    - This is the same thing we did with cyclic executives.
    - However, it is very expensive (in terms of overhead) to implement slack-stealing in priority-driven systems.

Simple Periodic Server

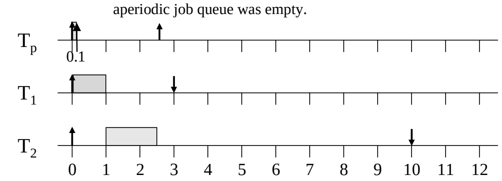
(Liu calls this a Polling server or the Poller)

- Periodic jobs are scheduled using any priority-driven scheduling algorithm.
- Aperiodic are executed by a special periodic server:
  - » The periodic server is a period task $T_p=(p_s, e_s)$.
    - e_s is called the execution budget of the server.
    - The ratio $u_s = e_s/p_s$ is the size of the server.
  - » Suspends as soon as the aperiodic queue is empty or T_p has executed for e_s time units (which ever comes first).
    - This is called exhausting its execution budget.
  - » Once suspended, it cannot execute again until the start of the next period.
    - That is, the execution budget is replenished (reset to e_s time units) at the start of each period.
    - Thus, the start of each period is called the replenishment time for the simple periodic server.

Periodic Server with RM Scheduling

Example Schedule: Two tasks, $T_1 = (3,1)$, $T_2 = (10,4)$, and a periodic server $T_p = (2.5,0.5)$. Assume an aperiodic job J_a arrives at $t = 0.1$ with and execution time of $e_a = 0.8$.

The periodic server cannot execute the job that arrives at time 0.1 since it was suspended at time 0 because the aperiodic job queue was empty.

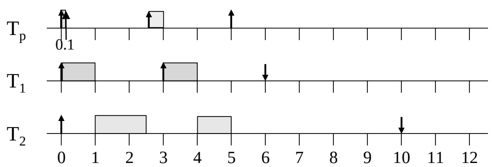


Periodic Server with RM Scheduling

(example continued)

Example Schedule: Two tasks, $T_1 = (3,1)$, $T_2 = (10,4)$, and a periodic server $T_p = (2.5,0.5)$. Assume an aperiodic job J_a arrives at $t = 0.1$ with and execution time of $e_a = 0.8$.

The periodic server executes job J_a until it exhausts its budget.

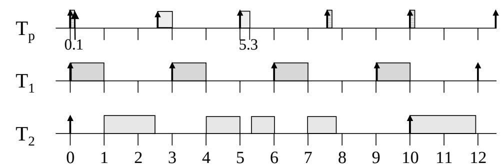


Periodic Server with RM Scheduling

(example concluded)

Example Schedule: Two tasks, $T_1 = (3,1)$, $T_2 = (10,4)$, and a periodic server $T_p = (2.5,0.5)$. Assume an aperiodic job J_a arrives at $t = 0.1$ with and execution time of $e_a = 0.8$.

The response time of the aperiodic job J_a is 5.2.



Improving the Periodic Server

- The problem with the periodic server is that it exhausts its execution budget whenever the aperiodic job queue is empty.
  - » If an aperiodic job arrives ε time units after the start of the period, it must wait until the start of the next period (p_s - ε time units) before it can begin execution.
- We would like to preserve the execution budget of the polling server and use it later in the period to shorten the response time of aperiodic jobs:
  - » Bandwidth-Preserving Servers do just this!

Bandwidth-Preserving Servers

- First some more terms:
  - » The periodic server is backlogged whenever the aperiodic job queue is nonempty or the server is executing a job.
  - » The server is idle whenever it is not backlogged.
  - » The server is eligible for execution when it is backlogged and has an execution budget (greater than zero).
  - » When the server executes, it consumes its execution budget at the rate of one time unit per unit of execution.
  - » Depending on the type of periodic server, it may also consume all or a portion of its execution budget when it is idle: the simple periodic server consumed all of its execution budget when the server was idle.

Bandwidth-Preserving Servers

- Bandwidth-preserving servers differ in their replenishment times and how they preserve their execution budget when idle.
- We assume the scheduler tracks the consumption of the server's execution budget and suspends the server when the budget is exhausted or the server becomes idle.
- The scheduler replenishes the servers execution budget at the appropriate replenishment times, as specified by the type of bandwidth-preserving periodic server.
- The server is only eligible for execution when it is backlogged and its execution budget is non-zero.

Four Bandwidth-Preserving Servers

- Deferrable Servers (1987)
  - » Oldest and simplest of the bandwidth-preserving servers.
  - » Static-priority algorithms by Lehoczky, Sha, and Strosnider.
  - » Deadline-driven algorithms by Ghazalie and Baker (1995).
- Sporadic Servers (1989)
  - » Static-priority algorithms by Sprunt, Sha, and Lehoczky.
  - » Deadline-driven algorithms by Ghazalie and Baker (1995).
- Total Bandwidth Servers (1994, 1995)
  - » Deadline-driven algorithms by Spuri and Buttazzo.
- Constant Utilization Servers (1997)
  - » Deadline-driven algorithms by Deng, Liu, and Sun.

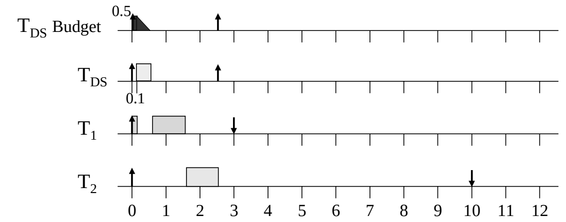
Deferrable Server (DS)

- Let the task $T_{DS} = (p_s, e_s)$ be a deferrable server.
- Consumption Rule:
  - » The execution budget is consumed at the rate of one time unit per unit of execution.
- Replenishment Rule:
  - » The execution budget is set to e_s at time instants kp_s, for $k \geq 0$.
  - » Note: Unused execution budget cannot be carried over to the next period.
- The scheduler treats the deferrable server as a periodic task that may suspend itself during execution (i.e., when the aperiodic queue is empty).

DS with RM Scheduling

Example Schedule: Same two tasks, $T_1 = (3,1)$, $T_2 = (10,4)$, and deferrable server $T_{DS} = (2.5,0.5)$. Assume an aperiodic job J_a arrives at time t = 0.1 with and execution time of $e_a = 0.8$ (again).

The DS can execute the job that arrives at time 0.1 since it preserved its budget when the aperiodic job queue was empty.

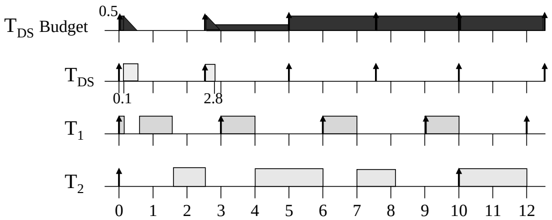


DS with RM Scheduling

(example concluded)

Example Schedule: Same two tasks, $T_1 = (3,1)$, $T_2 = (10,4)$, and deferrable server $T_{DS} = (2.5,0.5)$. Assume an aperiodic job J_a arrives at time t = 0.1 with and execution time of $e_a = 0.8$ (again).

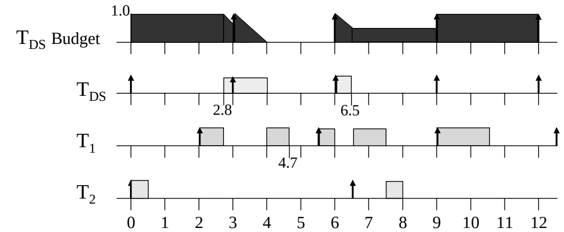
The response time of the aperiodic job J_a is 2.7
It was 5.2 with the simple periodic server.



DS with RM Scheduling

Another Example: Two tasks, $T_1 = (2,3.5,1.5)$, $T_2 = (6.5,0.5)$, and a deferrable server $T_{DS} = (3,1)$. Assume an aperiodic job J_a arrives at time t = 2.8 with and execution time of $e_a = 1.7$.

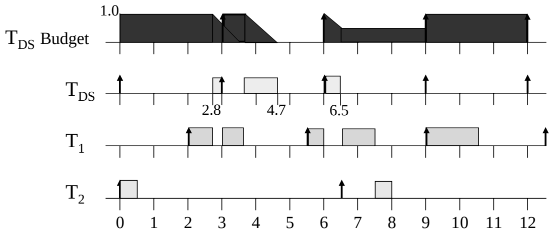
The response time of the aperiodic job J_a is 3.7.



DS with EDF Scheduling

Same Task Set: Two tasks, $T_1 = (2,3.5,1.5)$, $T_2 = (6.5,0.5)$, and a deferrable server $T_{DS} = (3,1)$. Assume an aperiodic job J_a arrives at time t = 2.8 with and execution time of $e_a = 1.7$.

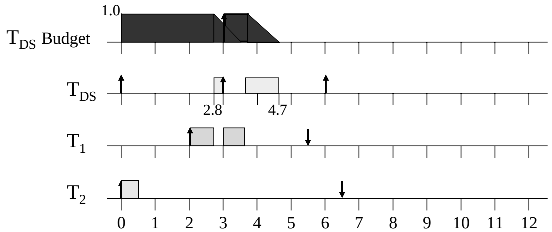
The response time of the aperiodic job J_a is still 3.7.



DS with EDF and Background Scheduling

Same Task Set: Two tasks, $T_1 = (2,3.5,1.5)$, $T_2 = (6.5,0.5)$, and $T_{DS} = (3,1)$ with background scheduling. Assume an aperiodic job J_a arrives at time t = 2.8 with and execution time of $e_a = 1.7$.

The DS exhausts its budget at time 4.7...

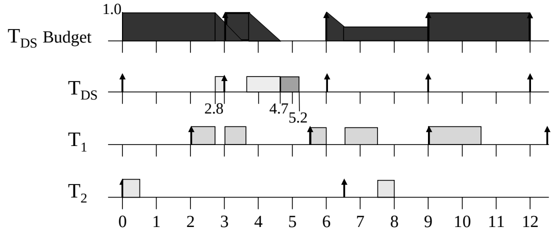


DS with EDF and Background Scheduling

(example concluded)

Same Task Set: Two tasks, $T_1 = (2,3.5,1.5)$, $T_2 = (6.5,0.5)$, and $T_{DS} = (3,1)$ with background scheduling. Assume an aperiodic job J_a arrives at time t = 2.8 with and execution time of $e_a = 1.7$.

However, using background scheduling, the response time of the aperiodic job J_a is reduced to 2.4.



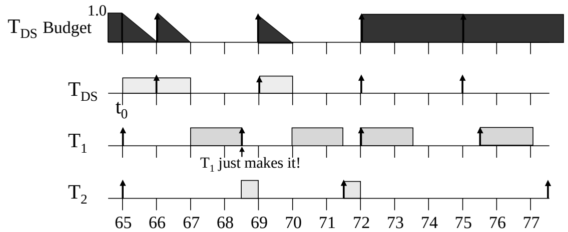
DS with Background Scheduling

- We can also combine background scheduling of the deferrable server with RM.
  - » For the deferrable server example task set, the response time doesn't change. Why?
- Why complicate things by adding background scheduling of the deferrable server?
- Why not just give the deferrable scheduler a larger execution budget? See the next slide!

DS with RM Scheduling Revisited

Modified Example: Same two tasks, $T_1 = (2,3.5,1.5)$, $T_2 = (6.5,0.5)$, and deferrable server $T_{DS} = (3,1)$. Assume an aperiodic job J_a arrives at time $t_0 = 65$ with and execution time of $e_a = 3$.

A larger execution budget for T_{DS} would result in T_1 missing a deadline. Time $t_0 = 65$ is a critical instant for this task set.



Schedulability and DS

- There are no known necessary and sufficient schedulability conditions for task sets that contain a DS with arbitrary priority. We will see why shortly.
- However, we can extend TDA and Generalized TDA to yield necessary and sufficient schedulability tests when the DS is the highest priority task in a periodic (real-world sporadic) task set.
- We start with a critical instant lemma for systems with a DS.

Critical Instants in Fixed-Priority Systems with a Deferrable Server

Lemma 7-1: [Lehoczky, Sha, and Strosnider] In a fixed-priority system in which the relative deadline of every independent, preemptable periodic task is no greater than its period and there is a deferrable server (p_s,e_s) with the highest priority among all tasks, a critical instant of every periodic task T_i occurs at time t_0 when all of the following are true.

1. One of its jobs $J_{i,c}$ is released at t_0.
2. A job in every higher-priority task is release at the same time.
3. The budget of the server is e_s at t_0, one or more aperiodic jobs are released at t_0, and they keep the server backlogged hereafter.
4. The next replenishment time of the server is $t_0 + e_s$.

Discussion of Lemma 7.1

The Proof of Lemma 7.1 is a straightforward extension of the proof we gave for Theorem 6.5. Convince yourself of this!

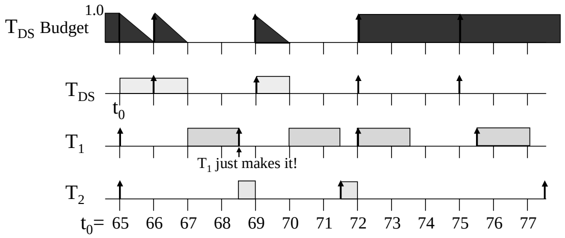
Note: We are not saying that $T_{DS}, T_1, \ldots, T_i$ will all necessarily release jobs at the same time, but if this does happen, we are claiming that the time of release will be a critical instant for T_i.

We can use the critical instant t_0, defined by Lemma 7.1, to derive necessary and sufficient conditions for the schedulability of a task set when the DS has highest priority.

First, lets take a look at a processor demand anomaly created by the bandwidth preserving DS.

Discussion of Lemma 7.1 (cont.)

All four conditions of Lemma 7.1 hold in the last example:



Notice that the processor demand created by the DS in an interval from [65,68.5] is twice what it would be if it were an ordinary periodic task! This is because we preserve the bandwidth of the DS.

TDA with a DS

Observation: Cases (1) and (2) of Lemma 7.1 define a critical instant for any fixed-priority task set. When cases (3) and (4) of Lemma 7.1 are true, the processor demand created by the DS in an interval of length *t* can be at most

$$e_s + \left\lceil \frac{t - e_s}{p_s} \right\rceil e_s \qquad (*)$$

Thus, TDA and Generalized TDA with blocking terms can be extended to systems with a DS that executes at the highest priority. The TDA function becomes:

$$\boxed{w_i(t) = e_i + b_i + e_s + \left\lceil \frac{t - e_s}{p_s} \right\rceil e_s + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \quad \text{for } 0 < t \le \min(D_i, p_i)}$$

DS with Highest Fixed Priority

- When the DS is the highest priority process in ta fixed-priority system:
  - » It may be able to execute an extra e_s time units more than a normal periodic task in the feasible interval of task T_i, as expressed in Equation (∗) and the modified $w_i(t)$.
- Thus, the TDA method, using the modified $w_i(t)$ provides a necessary and sufficient condition for fixed-priority systems with one DS executing at the highest priority.

DS with Arbitrary Fixed Priority

- When the DS is not the highest priority process:
  - » It may not be able to execute the extra e_s time units expressed in Equation (∗) and the modified $w_i(t)$.
  - » However, the time-demand function of a task T_i with lower priority than an arbitrary-priority DS is bounded from above by the modified $w_i(t)$.
- Thus, the TDA method provides a sufficient (but not necessary) condition for fixed-priority systems with one arbitrary-priority DS.

Multiple Arbitrary Fixed-Priority DS

We may want to differentiate aperiodic jobs by executing them at different priorities. To do this, we use multiple DS with different priorities and task parameters $(p_{s,k}, e_{s,k})$.

The TDA and Generalized TDA with blocking terms can be further extended to these systems. Specifically, the time demand function $w_i(t)$ of a periodic task T_i with a lower priority than *m* DS becomes:

$$\boxed{w_i(t) = e_i + b_i + \sum_{k=1}^{m}\left(1 + \left\lceil \frac{t - e_{s,k}}{p_{s,k}} \right\rceil\right) e_{s,k} + \sum_{k=1}^{i-1}\left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \quad \text{for } 0 < t \le \min(D_i, p_i)}$$

Schedulable Utilization with a Fixed-Priority DS

- We now look at utilization based scheduling tests for fixed-priority systems with one DS.
- There are no known necessary and sufficient schedulable utilization conditions for fixed-priority systems with a DS.
- However, there does exist a sufficient condition for RM when the DS has the shortest period plus some other conditions...

RM Schedulable Utilization with a DS

Theorem 7.2 Consider a system of *n* independent, preemptable periodic tasks whose periods satisfy the inequalities $p_s < p_1 < p_2 < \ldots < p_n < 2p_s$ and $p_n > p_s + e_s$ and whose relative deadlines are equal to their respective periods. This system is schedulable rate monotonically with a deferrable server (p_s, e_s) if their total utilization is less than or equal to

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

where u_s is the utilization e_s/p_s of the server.

Proof: Similar to Thm 6.11 and left as an exercise!

Note that this is only a **sufficient** schedulability test.

RM Schedulable Utilization with a DS and Arbitrary Periods

Observe: If $p_i < p_s$, then task T_i is unaffected by the DS. If $p_i > p_s$, then it may be blocked an extra e_s time units in a feasible interval.

Theorem: Consider a system of *n* independent, preemptable periodic tasks whose relative deadlines are equal to their respective periods. Task T_i with $p_i > p_s$ is schedulable rate monotonically with a deferrable server (p_s, e_s) if

$$U_i + u_s + \frac{e_s + b_i}{p_i} \le U_{RM}(i+1)$$

where u_s is the utilization e_s/p_s of the server, U_i is the total utilization of the tasks $T_1 \ldots T_i$, and b_i is the blocking time encountered by task T_i from lower priority tasks.

Schedulability of Deadline-Driven System with a DS

- In fixed-priority systems, the DS behaves like a periodic task (p_s, e_s) except that it could execute an extra amount of time (at most e_s time units) in the feasible interval of any lower priority job.
- In a deadline-driven system, the DS can execute at most e_s time units in the feasible interval of any job (under certain conditions).
- We present a sufficient (but not necessary) schedulability condition for the EDF algorithm.
- First, a bound on the processor demand created by a DS.

Bounding the Demand of a DS in an EDF Scheduled System

- An interval $(a, b]$ is post-idle if either $a = 0$, or if no job with a deadline in the interval $(a+1, b]$ executes in the interval $(a-1, a]$.
  - » The implication of this definition is that all jobs with deadlines in $(a+1, b]$ are "idle" during the interval $(a-1, a]$ in the sense that all jobs released before time a with deadlines in $(a+1, b]$ have completed execution before time a (*i.e.*, either the processor is idle in $(a-1, a]$ or a job with deadline at time a or after time b executes in $(a-1, a]$).
- The following lemma gives us a simple upper bound for the processor demand in a post-idle interval of length L.

Maximum Demand of a DS

Lemma: The maximum demand $w_{DS}(L)$ of a DS= (p_s, e_s) during a post-idle interval of length L in an EDF scheduled system of *n* independent, preemptable periodic tasks is bounded such that

$$w_{DS}(L) \leq u_s (L + p_s - e_s)$$

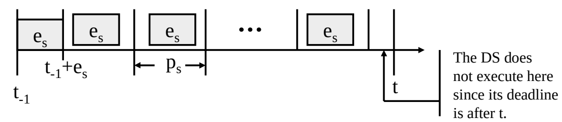
where u_s is the utilization e_s/p_s of the server.

Proof: Let $(t_{-1}, t]$ be a post-idle interval. The maximum demand for DS occurs when

1. At time t_{-1}, its budget is equal to e_s and the server's deadline (and budget replenished time) is $t_{-1} + e_s$.
2. One or more aperiodic jobs arrive at t_{-1} and the DS is backlogged until at least time t.
3. The server's deadline $t_{-1} + e_s$ is earlier than the deadlines of all the periodic jobs that are ready for execution in the interval $(t_{-1}, t_{-1} + e_s]$.

Proof Continued

Maximum demand created by DS in a post-idle interval under EDF:



Observe that under these conditions, the maximum demand created by DS in the post-idle interval $(t_{-1},t]$ is at most

$$e_s + \left\lfloor \frac{t-(t_{-1}+e_s)}{p_s} \right\rfloor e_s = e_s + \left\lfloor \frac{t-t_{-1}-e_s}{p_s} \right\rfloor e_s$$

Proof Continued

Thus, $$w_{DS}(t-t_{-1}) \le e_s + \left\lfloor \frac{t-t_{-1}-e_s}{p_s} \right\rfloor e_s \le e_s + \frac{t-t_{-1}-e_s}{p_s} e_s$$

$$= \frac{p_s}{p_s} e_s + \frac{t-t_{-1}-e_s}{p_s} e_s = p_s u_s + (t-t_{-1}-e_s) u_s$$

$$= u_s (p_s + (t-t_{-1}-e_s))$$

$$= u_s (t-t_{-1}+p_s-e_s)$$

Since $(t_{-1}, t]$ is a post-idle interval of length $L = t - t_{-1}$,

$$w_{DS}(L) \le u_s (L + p_s - e_s).$$

Schedulability with a DS

- Combining this result with Theorem 6.2, we get the following theorem from Ghazalie and Baker:

Theorem 7.3: A periodic task T_i in a system of *n* independent, preemptable periodic tasks is schedulable with a DS= (p_s, e_s) according to the EDF algorithm if

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + u_s \left(1 + \frac{p_s - e_s}{D_i}\right) \le 1 \qquad (7.5)$$

where u_s is the utilization e_s/p_s of the server.

Proof of Theorem 7.3

Suppose Equation (7.5) holds for task T_i but a deadline is missed. Let t_d be the earliest point in time at which a deadline is missed and t_{-1} be the start of last post-idle interval that includes time t_d. Thus, a deadline is missed in the post-idle interval $(t_{-1}, t_d]$.

From Theorem 6.2 and the previous lemma, the demand in this interval is at most

$$\sum_{k=1}^{n}\left\lfloor \frac{t_d - t_i}{\min(D_k, p_k)} \right\rfloor e_k + u_s(t_d - t_i + p_s - e_s)$$

Because a deadline is missed at t_d, demand over $(t_{-1}, t_d]$ exceeds $t_d - t_{-1}$. Thus, we have

$$\begin{aligned} t_d - t_i &< \sum_{k=1}^{n}\left\lfloor \frac{t_d - t_i}{\min(D_k, p_k)} \right\rfloor e_k + u_s(t_d - t_i + p_s - e_s) \\ &\le \sum_{k=1}^{n} \frac{t_d - t_i}{\min(D_k, p_k)} e_k + u_s(t_d - t_i + p_s - e_s) \end{aligned}$$

Proof Continued

Dividing both sides by $(t_d - t_{-1})$, we get

$$\begin{aligned} 1 &< \sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + \frac{u_s(t_d - t_i + p_s - e_s)}{t_d - t_i} \\ &= \sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + u_s\left(1 + \frac{p_s - e_s}{t_d - t_i}\right) \\ &\le \sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + u_s\left(1 + \frac{p_s - e_s}{D_i}\right) \end{aligned}$$

Since $D_i \le (t_d - t_{-1})$.

This contradicts our assumption that Equation (7.5) holds.

Multiple DS

We may want to differentiate aperiodic jobs by executing them at different priorities. To do this in a deadline-driven system, we (again) use multiple DS with different priorities and task parameters $(p_{s,k}, e_{s,k})$.

Corollary: A periodic task T_i in a system of *n* independent, preemptable periodic tasks is schedulable with *m* a DS= $(p_{s,k}, e_{s,k})$ according to the EDF algorithm if

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + \sum_{k=1}^{m} u_{s,k}\left(1 + \frac{p_{s,k} - e_{s,k}}{D_i}\right) \le 1$$

where $u_{s,k}$ is the utilization $e_{s,k}/p_{s,k}$ of server *k*.

The proof is left as an exercise.

DS Punch Line

- In both fixed-priority and deadline-driven systems, we see that the DS behaves like a periodic task with parameters (p_s, e_s) except it may execute an additional amount of time in the feasible interval of any lower priority job.
- This is because, the bandwidth-preserving conditions result in a scheduling algorithm that is non-work-conserving with respect to a normal periodic task.

Sporadic Servers

- Sporadic Servers (SS) were designed to overcome the additional blocking time a DS may impose on lower-priority jobs.
- All sporadic servers are bandwidth preserving, but the consumption and replenishment rules ensure that a SS, specified a $T_S = (p_s, e_s)$ never creates more demand than a periodic (“real-world” sporadic) task with the same task parameters.
- Thus, schedulability of a system with a SS is determined exactly as a system without a SS.

Sporadic Servers (SS)

- We will look at two SS for fixed-priority systems and one for deadline-driven systems.
- They differ in complexity (and thus overhead) due to different consumption and replenishment rules.
- We assume, as with a DS, that the scheduler monitors the execution budget of the SS.
- However, in all cases schedulability conditions remain unchanged from an equivalent system without an SS.

Simple SS in a Fixed-Priority System

- First some (new) terms:
  - » Let **T** be a set of *n* independent, preemptable periodic tasks.
  - » The (arbitrary) priority of the server T_S in **T** is π_s.
  - » $\mathbf{T_H}$ is the subset of tasks that have higher priority than T_S.
  - » **T** ($\mathbf{T_H}$) is idle when no job in **T** ($\mathbf{T_H}$) is eligible for execution.
    - **T** ($\mathbf{T_H}$) is busy when it is not idle.
  - » Let BEGIN be the instant in time when $\mathbf{T_H}$ transitions from idle to busy, and END be the instant in time when it becomes idle again (or infinity if $\mathbf{T_H}$ is still busy) .
    - The interval (BEGIN, END] is a busy interval.
  - » t_r is the last replenishment time of T_S.
  - » t_r' is the next scheduled replenishment time of T_S.
  - » t_e is the *effective* replenishment time of T_S.
  - » t_f is the first instant after t_r at which T_S begins to execute.

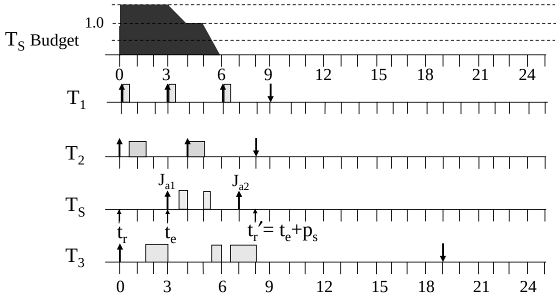
Simple SS in a Fixed-Priority System

- Consumption Rule: at any time t after t_r, T_S consumes its budget at the rate of one time unit per unit of execution until the budget is exhausted when either
  - C1 T_S is executing, or
  - C2 T_S has executed since t_r and END < t. (END < t $\Rightarrow$ T_H is currently idle.)
- Replenishment Rule: t_r is set to the current time whenever the execution budget is replenished with e_s time units by the scheduler.
  - R1 Initially, $t_r = t_e = 0$ and $t_r' = p_s$ (assuming the system starts at time 0).
  - R2 At time t_f,
    - if END = t_f, $t_e = \max(t_r, \text{BEGIN})$.
    - If END < t_f, $t_e = t_f$.

    The next scheduled replenishment time is $t_r' = t_e + p_s$.
  - R3 The next replenishment occurs at t_r' except
    - (a) If $t_r' < t_f$, then the budget is replenished as soon as it is exhausted.
    - (b) If **T** is idle before t_r' and then begins a new busy interval at t_b, then the budget is replenished at $\min(t_r', t_b)$.

Simple SS with RM Scheduling

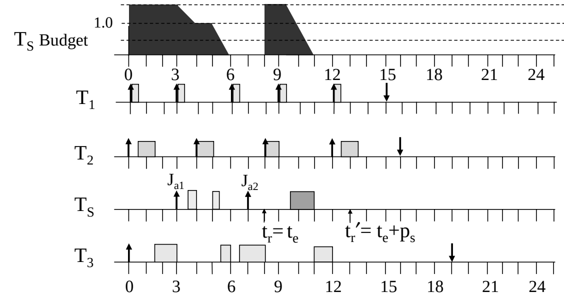
Example Schedule: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



Simple SS with RM Scheduling

(example continued)

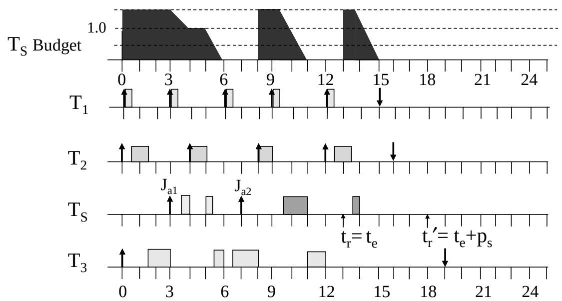
Example Schedule: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



Simple SS with RM Scheduling

(example continued)

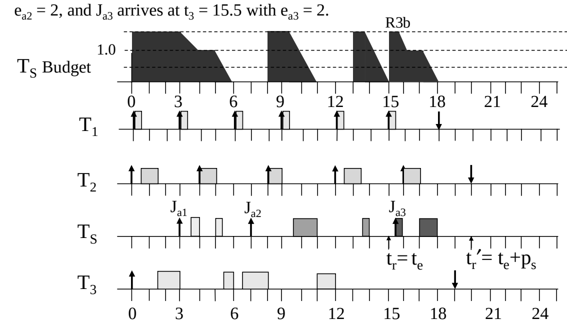
Example Schedule: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



Simple SS with RM Scheduling

(example continued)

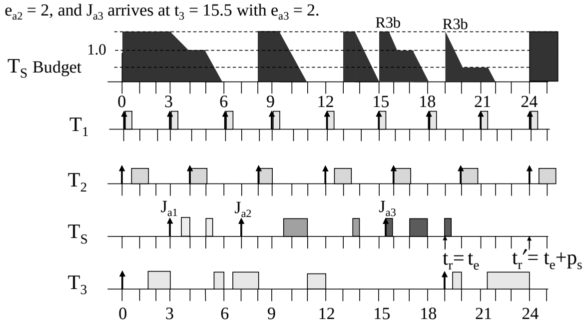
Example Schedule: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



Simple SS with RM Scheduling

(example concluded)

Example Schedule: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



Correctness of Simple SS

- The Simple SS behaves exactly as a "real-world" sporadic task except when Rule R3b is applied.
- Rule R3b takes advantage of the schedulability test for a fixed-priority periodic ("real-world" sporadic) task set **T**.
  - » We know that if the system **T** transitions from an idle state to a busy interval, all jobs will make their deadlines--even if they are all released at the same instant (at the start of the new busy interval).
  - » Thus Rule R3b replenishes the Simple SS at this instant since it will not affect schedulability!

Enhancements to the Simple SS

- We can improve response times of aperiodic jobs by combining the Background Server with the Simple SS to create a Sporadic/Background Server (SBS).
- Consumption Rules are the same as for the Simple SS except when the task system **T** is idle.
  - » As long as T is idle, the execution budget stays at e_s.
- Replenishment Rules are the same as for the Simple SS except Rule R3b.
  - » The SBS budget is replenished at the beginning of each idle interval of T. t_r is set at the end of the idle interval.

Other Enhancements to the Simple SS

- We can also improve response times of aperiodic jobs by replenishing the server's execution budget in small chunks during its period rather than with a single replenishment of e_s time units at the end.
  - » This adds to the complexity of the consumption and replenishment rules, of course.
- Sprunt, Sha, and Lehoczky proposed such a server:
  - » The SpSL sporadic server preserves unconsumed chunks of budget whenever possible and replenishes the consumed chunks as soon as possible.
  - » Thus, it emulates several periodic tasks with parameters $(p_s, e_{s,k})$ such that $\Sigma\, e_{s,k} = e_s$.

SpSL in a Fixed-Priority System

- Breaking of Execution Budget into Chunks:
  - B1 Initially, the budget = e_s and $t_r = 0$. There is only one chunk of budget.
  - B2 Whenever the server is suspended, the current budget e, if not exhausted, is broken up into two chunks.
    - The first chunk is the portion that was consumed during the last server busy interval, e_1.
      - Its next replenishment time, t_{r1}', is the same as the original chunk's: t_r'. The replenishment amount will be e_1.
    - The second chunk is the remaining budget, e_2.
      - Its last replenishment time is tentatively set to $t_{r2} = t_e$, which will be reset if this budget is used before t_{r1}'. Otherwise, the two chunks will be combined into one budget at time t_{r1}'.
- Consumption Rules:
  - C1 The server consumes budgets (when there is more than one budget chunk) in the order of their last replenishment times. That is, the budget with smallest t_r is consumed first.
  - C2 The server consumes its budget only when it executes.
- Replenishment Rules: The next replenishment time of each chunk of budget is set according to rules R2 and R3 of the simple SS. The budget chunks are combined whenever they are replenished at the same time (e.g. R3b).

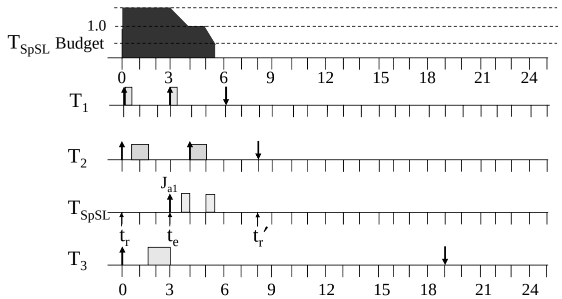
SpSL Rules R2 and R3

- Replenishment rules R2 and R3 from the Simple SS:
  - R2 At time t_f,
    - if END = t_f, $t_e = \max(t_r, \text{BEGIN})$.
    - If END < t_f, $t_e = t_f$.

    The next scheduled replenishment time is $t_r' = t_e + p_s$.
  - R3 The next replenishment occurs at t_r' except
    - (a) If $t_r' < t_f$, then the budget is replenished as soon as it is exhausted.
    - (b) If **T** is idle before t_r' and then begins a new busy interval at t_b, then the budget is replenished at $\min(t_r', t_b)$.
- Notice that when R3b applies, all budget chunks will be combined into a single budget again.

SpSL with RM Scheduling

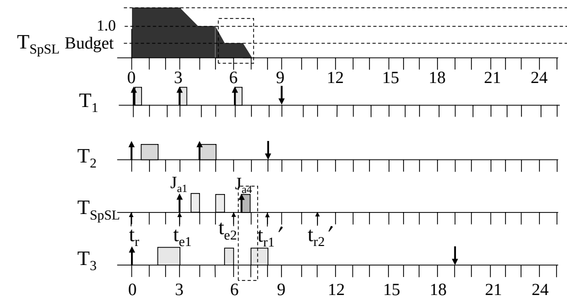
Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



SpSL with RM Scheduling

(example continued)

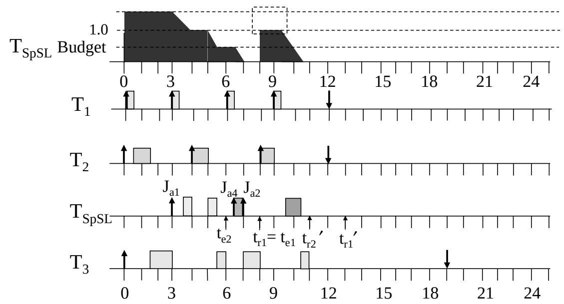
Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



SpSL with RM Scheduling

(example continued)

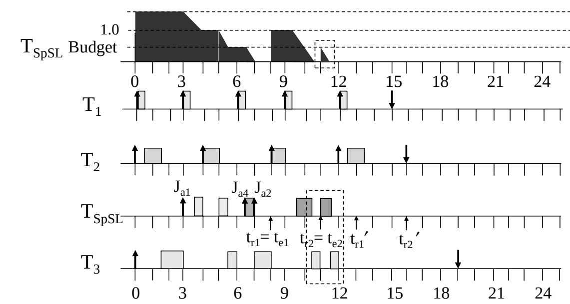
Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



SpSL with RM Scheduling

(example continued)

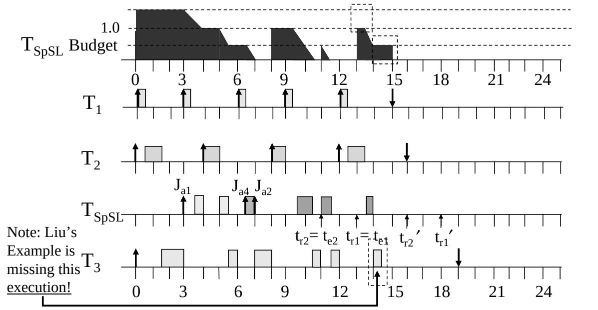
Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



SpSL with RM Scheduling

(example continued)

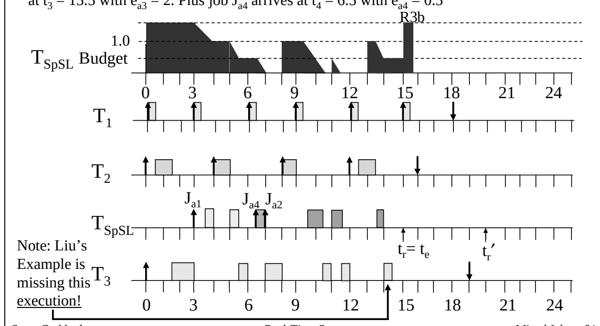
Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



SpSL with RM Scheduling

(example continued)

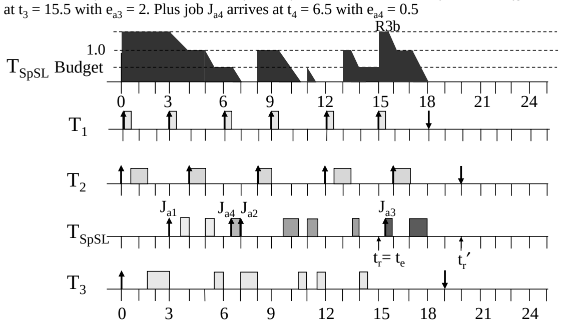
Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



SpSL with RM Scheduling

(example continued)

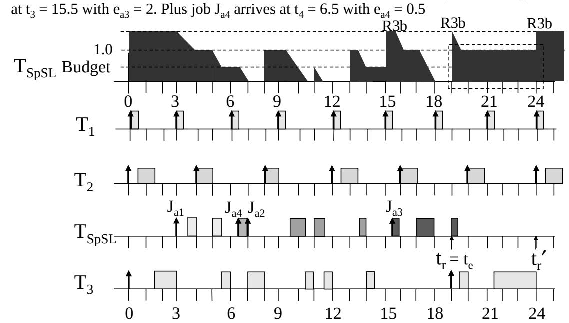
Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



SpSL with RM Scheduling

(example concluded)

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_{SpSL} = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$. Plus job J_{a4} arrives at $t_4 = 6.5$ with $e_{a4} = 0.5$



Enhancing the SpSL Server

- We can further improve response times of aperiodic jobs by combining the SpSL with the Background Server to create a SpSL/Background Server.
  - » Try to write precisely the consumption and replenishment rules for this server!
- We can also enhance the SpSL by using a technique called Priority Exchanges.
  - » When the server has no work, it trades time with an executing lower priority task.
  - » See Liu for details.

Simple SS in Deadline-Driven Systems

- Let $T_S=(p_s,e_s)$ be a simple sporadic server (SS) in a task system **T** scheduled with EDF.
- The server is ready for execution only when it is backlogged and its deadline d_s is set.
  - » Thus, the server is suspended whenever its deadline is undefined or when the server is idle.
- Consumption Rules: T_S consumes its budget at the rate of one time unit per unit of execution until the budget is exhausted when either

  C1 T_S is executing, or

  C2 d_s is defined, the server is idle, and there is no job with a deadline before d_s ready for execution.

Simple SS in Deadline-Driven Systems

- Replenishment Rule: t_r is set to the current time whenever the execution budget is replenished with e_s time units by the scheduler.

  R1 Initially, $t_r = 0$, t_e and d_s are undefined.

  R2 Whenever t_e is defined, $d_s = t_e + p_s$, and $t_r' = t_e + p_s$. When t_e is undefined, t_e is determined (defined) as follows:

  (a) At time t when an aperiodic job arrives an the server is idle, the value of t_e is determined based on the history of the system before t as follows:

  i. If only jobs with deadlines earlier than $t_r + p_s$ have executed throughout the interval (t_r, t), $t_e = t_r$.

  ii. If a job with a deadline after $t_r + p_s$ has executed in the interval (t_r, t), $t_e = t$.

  (b) At replenishment time t_r,

  i. If the server is backlogged, $t_e = t_r$.

  ii. If the server is idle, t_e and d_s become undefined.

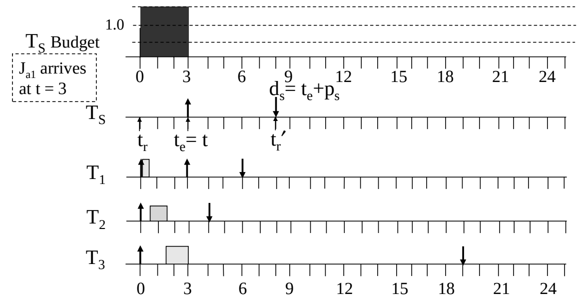
  R3 The next replenishment occurs at t_r' except

  (a) If $t_r' < t$ (from R2a), the budget is replenished as soon as it is exhausted.

  (b) The budget is replenished at the end of each idle interval of **T**.

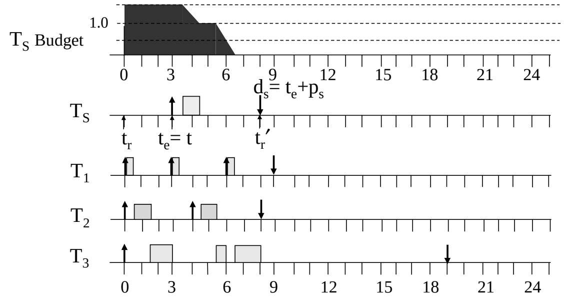
Simple SS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



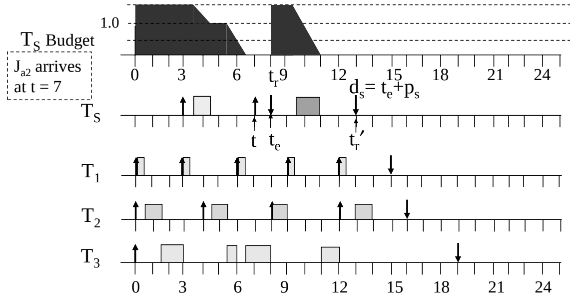
Simple SS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



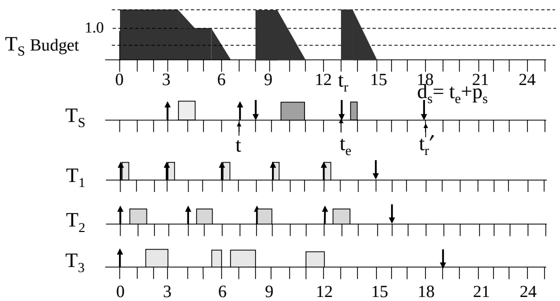
Simple SS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



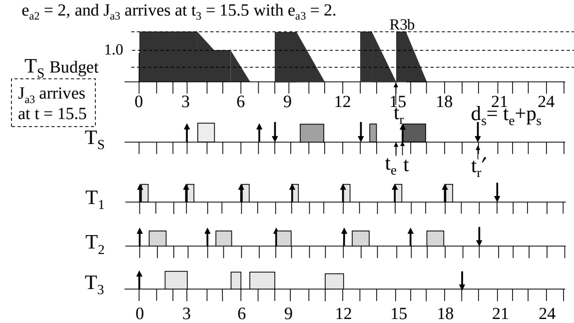
Simple SS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



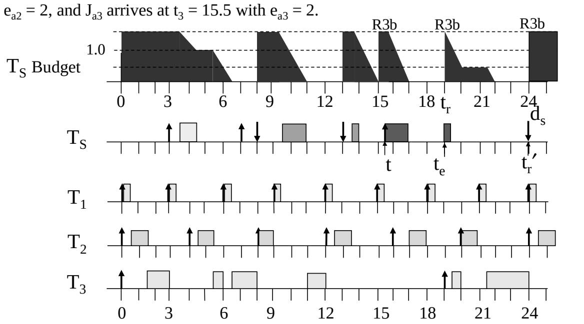
Simple SS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



Simple SS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $T_S = (5,1.5)$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 7$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 15.5$ with $e_{a3} = 2$.



Enhancing the Simple SS Under EDF

- We can improve response times of aperiodic jobs by combining the Background Server with the Simple SS to create a Sporadic/Background Server (SBS) that executes under EDF.
  - » Try to write precisely the consumption and replenishment rules for this server!
- We can also enhance the Simple SS by replenishing the budget in chunks as the SpSL server did.
  - » Such a server was proposed by Ghazalie and Baker.
  - » Its rules are defined in Prob 7.13 of Liu's text.

Aperiodic Job Servers for Deadline-Driven Systems

- The Total Bandwidth Server (TBS) was created by Spuri and Butazzo (RTSS '94) to schedule
  - aperiodic task whose arrival time was unknown but
  - whose worst-case execution time (wcet) was known.
  - A trivial admission control algorithm used with the TBS can also schedule sporadic jobs (aperiodic jobs whose wcet and deadline is known).
- The constant utilization server was created by Deng, Liu, and Sun (Euromicro Workshop '97) to schedule
  - aperiodic task whose arrival time was unknown but
  - whose wcet and deadline was known.
  - However, they also wanted to schedule aperiodic jobs with no deadlines and whose wcet was unknown.
  - This server is almost the same as the TBS.

Aperiodic Job Servers for Deadline-Driven Systems

- Liu's presentation of the constant utilization server and the TBS is poorly motivated and nearly unintelligible.
- You should read the papers, they make more sense than the material in the book.
- We will first cover the TBS and then the constant utilization server since it may be easier to understand the value of the constant utilization server when they are presented in this order.

Total Bandwidth Server (TBS)

- One way to reduce the response time of aperiodic jobs whose wcet is known in a deadline-driven system is to
  - » allocate a fixed (maximum) percentage, U_S, of the processor to the serve aperiodic jobs, and
  - » make sure the aperiodic load never exceeds this maximum utilization value.
  - » When an aperiodic job comes in, assign it a deadline such that the demand created by all of the aperiodic jobs in any feasible interval never exceeds the maximum utilization allocated to aperiodic jobs.
- This approach is the main idea behind the TBS.

Note: We use U_S to denote the server size (as Spuri and Buttazzo do) rather than $\tilde{u}_s$ as Liu does.

TBS in Deadline-Driven Systems

- Let TB be a TBS of size U_S in a task system **T** scheduled with EDF. Thus, the server is allocated U_S percent of the total processor bandwidth.
- The server is ready for execution only when it is backlogged.
  - » In other words: the server is only suspended when it is idle.
- Consumption Rule: When executing, TB consumes its budget at the rate of one time unit per unit of execution until the budget is exhausted.
- Replenishment Rule:

  R1 Initially, the execution budget $e_s = 0$ and the server's deadline $d_s = 0$.

  R2 At time t when aperiodic job J_i with execution time e_i arrives and the server is idle, set $d_s = \max(d_s, t) + e_i/U_S$ and $e_s = e_i$. If the server is backlogged, no nothing.

  R3 When the server completes the currently executing aperiodic job, J_i,

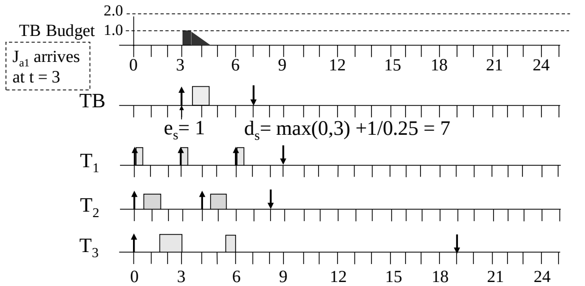
  (a) If the server is backlogged, update the server deadline and budget:

  $d_s = d_s + e_{i+1}/U_S$ and $e_s = e_{i+1}$.

  (b) If the server is idle, do nothing.

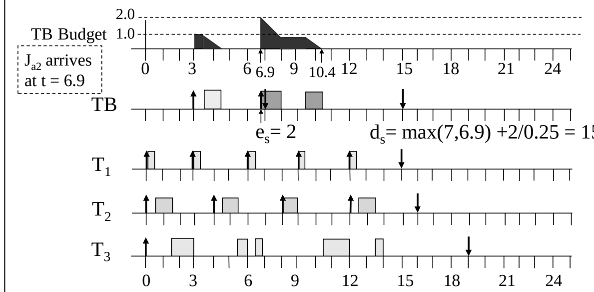
TBS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, but $U_S = 0.25$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 6.9$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 14$ with $e_{a3} = 2$.



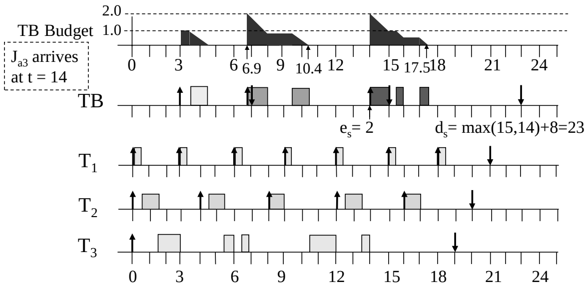
TBS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, but $U_S = 0.25$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 6.9$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 14$ with $e_{a3} = 2$.



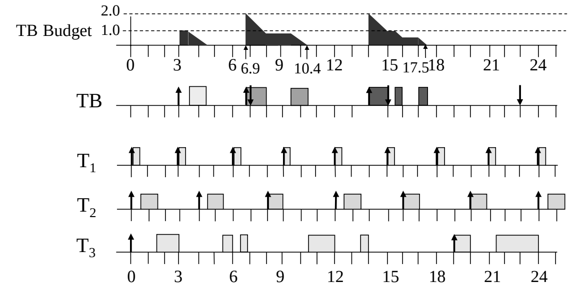
TBS with EDF Scheduling

Same Task Set: T_1 = (3,0.5), T_2 = (4,1), T_3 = (19,4.5), but U_S = 0.25. Assume aperiodic job J_{a1} arrives at t_1 = 3 with e_{a1} = 1, J_{a2} arrives at t_2 = 6.9 with e_{a2} = 2, and J_{a3} arrives at t_3 = 14 with e_{a3} = 2.



TBS with EDF Scheduling

Same Task Set: T_1 = (3,0.5), T_2 = (4,1), T_3 = (19,4.5), but U_S = 0.25. Assume aperiodic job J_{a1} arrives at t_1 = 3 with e_{a1} = 1, J_{a2} arrives at t_2 = 6.9 with e_{a2} = 2, and J_{a3} arrives at t_3 = 14 with e_{a3} = 2.



TBS Processor Demand

- Observe that the total bandwidth of the server is allocated to one aperiodic job at a time.
- Moreover, if J_i's execution time is less than its specified wcet, e_i, and job J_{i+1} arrives before the deadline for job J_i, the TBS effectively performs background execution by replenishing the server's budget as soon as job J_i is done.
- However, the deadline value $d_s = \max(d_s,t) + e_{i+1}/U_S$ for job J_{i+1} is is no earlier than e_{i+1}/U_S time units from the deadline of job J_i.
- Thus, the demand created by any *n* jobs in an interval $(t_1,t_2]$ is at most

$$\left(\sum_{i=1}^{n} \frac{e_i}{\frac{e_i}{U_S}}\right) = \left(\sum_{i=1}^{n} \frac{e_i}{e_i}\right) U_S = nU_S \le (t_2 - t_1)(U_S)$$

Schedulability with a TBS

- Combining this observation with Theorem 6.1, we get the following theorem from Spuri and Buttazzo:

Theorem: A system **T** of *n* independent, preemptable, periodic tasks with relative deadlines equal to their periods is schedulable with a TBS if and only if

$$U_T + U_S \le 1$$

where $U_T = \sum_{k=1}^{n} \frac{e_k}{p_k}$ is the processor utilization of the periodic tasks and U_S is the processor utilization of the TBS.

Schedulability with a TBS

- Combining this result with Theorem 6.2, we get the following theorem:

Theorem: A system **T** of *n* independent, preemptable, periodic tasks is schedulable with a TBS if

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + U_S \leq 1$$

where U_S is the processor utilization of the TBS.

Sporadic jobs and the TBS

- Recall that a sporadic job is like an aperiodic job except it has a hard deadline.
  - » Assume each sporadic job J_i has a release time r_i, wcet e_i, and a relative deadline D_i: $J_i = (r_i, e_i, D_i)$
- The TBS assigns deadlines such that the (absolute) deadline d_i assigned any job J_i is $d_i = \max(r_i, d_{i-1}) + e_i/U_S$ where $d_0=1$.
- Thus the TBS can guarantee the deadlines of all accepted sporadic jobs if it only accepts job J_{i+1} if $r_{i+1} + D_{i+1} \leq d_{i+1}$ and rejects it otherwise.

Periodic, Aperiodic, and Sporadic Jobs

- In some systems, we may want to support all three types of jobs: periodic, aperiodic, and sporadic.
- The TBS can do this by accepting all aperiodic jobs and only accepting a sporadic job if its deadline can be met.
- But there is no value in completing sporadic jobs before their deadline, which the TBS will do.
- Thus Deng, Liu, and Sun created the constant utilization server (CUS).
  - » We can schedule sporadic jobs with the CUS and aperiodic jobs with the TBS in the same system.

CUS in Deadline-Driven Systems

- Let CU be a CUS of size U_S in a task system **T** scheduled with EDF.
- As with the TBS, the server is ready for execution only when it is backlogged.
  - » Thus, the server is only suspended when the server is idle.
- Consumption Rule: When executing, CU consumes its budget at the rate of one time unit per unit of execution until the budget is exhausted.
- Replenishment Rule:

R1 Initially, the execution budget $e_s = 0$ and the server's deadline $d_s = 0$.

R2 At time t when aperiodic job J_i with execution time e_i arrives and the server is idle,

(a) If $t < d_s$, do nothing;

(b) If $t \geq d_s$, $d_s = t + e_i/U_S$ and $e_s = e_i$.

R3 At the deadline d_s of the server,

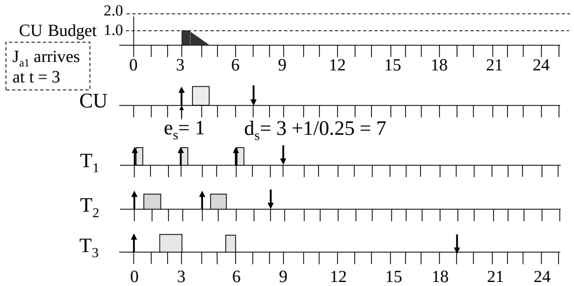
(a) If the server is backlogged, update the server deadline and budget:

$d_s = d_s + e_{i+1}/U_S$ and $e_s = e_{i+1}$.

(b) If the server is idle, do nothing.

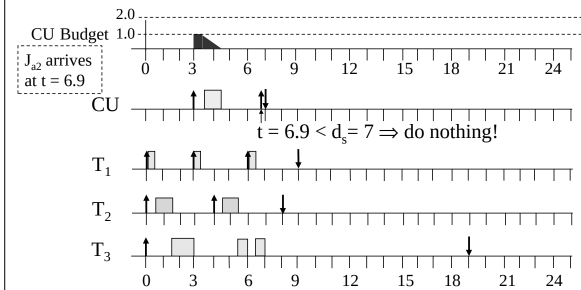
CUS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $U_S = 0.25$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 6.9$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 14$ with $e_{a3} = 2$.



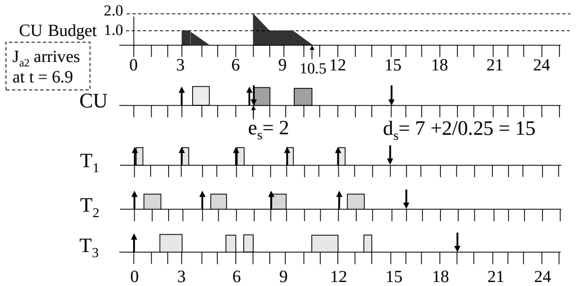
CUS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $U_S = 0.25$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 6.9$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 14$ with $e_{a3} = 2$.



CUS with EDF Scheduling

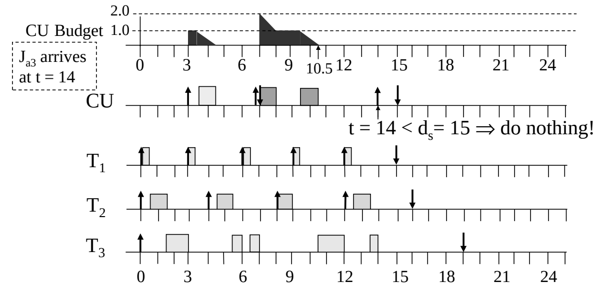
Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $U_S = 0.25$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 6.9$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 14$ with $e_{a3} = 2$.



Steve Goddard Real-Time Systems Mixed Jobs - 93

CUS with EDF Scheduling

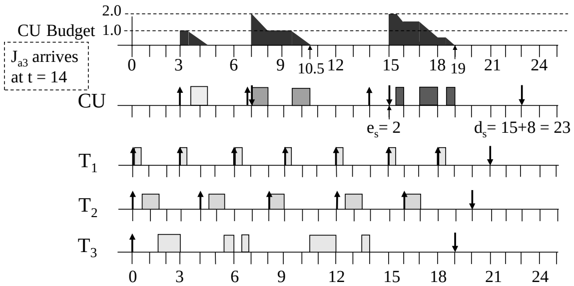
Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $U_S = 0.25$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 6.9$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 14$ with $e_{a3} = 2$.



Steve Goddard Real-Time Systems Mixed Jobs - 94

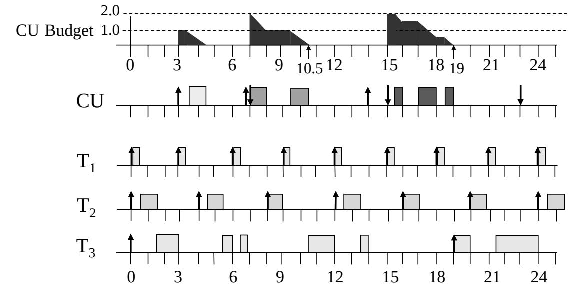
CUS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $U_S = 0.25$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 6.9$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 14$ with $e_{a3} = 2$.



CUS with EDF Scheduling

Same Task Set: $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (19,4.5)$, and $U_S = 0.25$. Assume aperiodic job J_{a1} arrives at $t_1 = 3$ with $e_{a1} = 1$, J_{a2} arrives at $t_2 = 6.9$ with $e_{a2} = 2$, and J_{a3} arrives at $t_3 = 14$ with $e_{a3} = 2$.



Comments on the CUS

- Deadlines and replenish amounts are the same in both servers.
- The main difference between the CUS and the TBS is that the CUS never replenishes the server's budget early.
- Thus, the TBS actually yields better average response times for aperiodic jobs (and it was created before the CUS…)
- Moreover, it would appear that the TBS may be able to accept more sporadic jobs than the CUS.
- The value of the CUS is not clear, and Liu does a terrible job arguing for it!

TBS and CUS Summary

- Both servers can be modified to support the case when the wcet of aperiodic jobs is unkown:
  - » fix the execution budget at some value e_s and assume the server has a period of e_s/U_S.
- Both servers can be modified to "reclaim unused resources" when the actual execution time e is less than the wcet e_s that we assumed:
  - » reduce the current deadline of the server by $(e_s-e)/U_S$ units before replenishing the budget.
- We can have multiple TBS/CUS servers as long as the total processor utilization/density is not greater than 1.