Priority-Driven Scheduling of Aperiodic and Sporadic Tasks

Assumptions and Approaches

- Independent tasks,
- Preemptable jobs.
- Execution time of sporadic jobs known after they are released,
- Sporadic jobs that cannot complete in time are rejected.

Objectives, Correctness and Optimality

- Assumption:
  - Periodic tasks meet all their deadline, according to some scheduling algorithm, when there are no aperiodic and sporadic jobs.
  - The operating system maintains three priority queues:
    - Periodic jobs,
    - Aperiodic jobs,
    - Sporadic jobs, preceded by an acceptance test.

Objectives, Correctness and Optimality

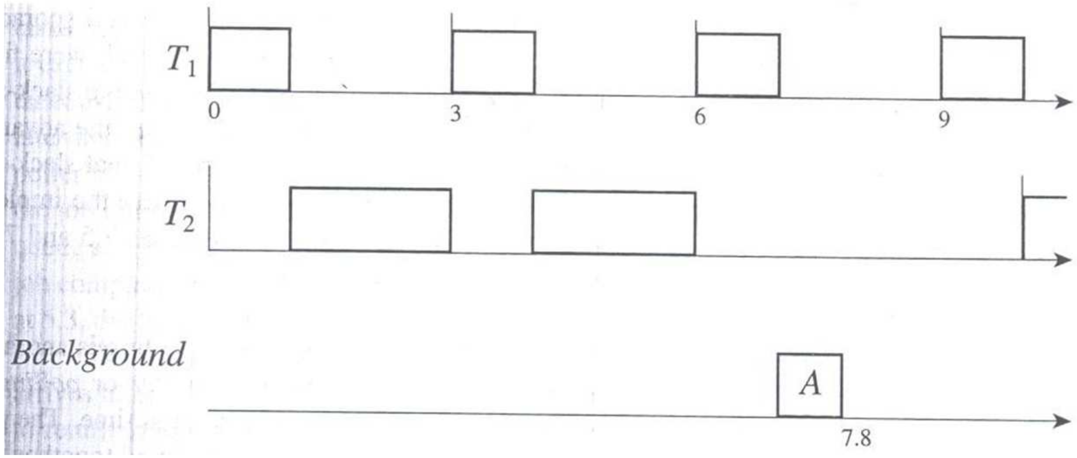
- A *correct* (aperiodic and sporadic jobs) scheduling algorithm produces *correct* schedules: periodic and accepted sporadic jobs never miss their deadlines.
- An *optimal* aperiodic jobs scheduling algorithm minimizes the response time of the aperiodic job at the head of the queue, or the average response time of all aperiodic jobs.
- An *optimal* algorithm for accepting and scheduling sporadic jobs accepts a sporadic job only if it can be *correctly* scheduled.
- Three common used approaches: Background, polled and interrupt-driven approaches.

Background and Interrupt-Driven Approaches

- Aperiodic jobs are scheduled and executed when there is no ready periodic or sporadic job ready for execution.
- Produces correct schedules and Easy to implement, but
- Response time of Aperiodic jobs may be prolonged.
- Example: $T_1 = (3, 1)$ and $T_1 = (10, 4)$. RMA is used. A is released at 0.1 ($e_A = 0.8$).

Background and Interrupt-Driven Approaches

- Response time = 7.7 !!!



- Interrupt-Driven Execution: Execution of periodic tasks is interrupted and the aperiodic job is executed ➔ Periodic task may miss their deadlines.

Slack Stealing

- Postpone the execution of periodic tasks when it's safe to do so (when they have *slack*)
- Back to the example.

Polled Execution

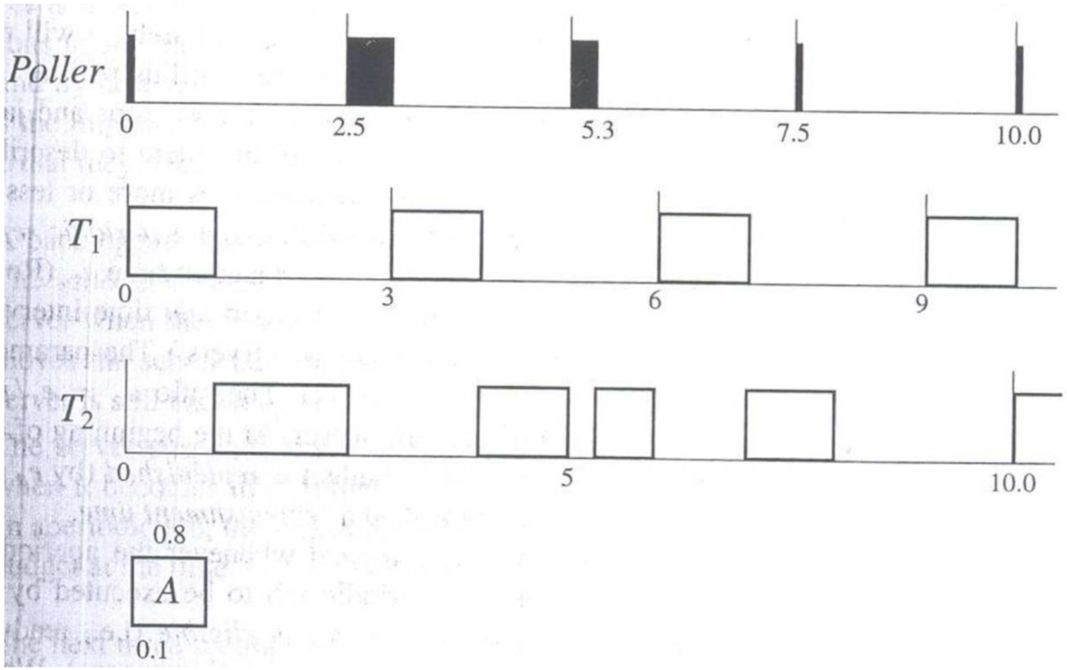
- A *poller* or *polling server* (p_s, e_s) is a periodic task with p_s as a period and e_s as an execution time.
- The poller is scheduled with the other periodic tasks.
- When executing, it examines the aperiodic jobs queue.
  - If queue is nonempty, the poller executed the aperiodic jobs at the head of the queue.
  - If the queue is empty, the poller suspends itself.
  - The Poller is suspended when it has executed for e_s units of time.

Polled Execution: Terminology

- *Periodic Server*: a task that behaves like a periodic task, and is created for purpose of executing aperiodic jobs.
- e_s is called the (*execution*) *budget*.
- $u_s = e_s / p_s$ is called the *size* of the server.
- At the beginning of each period, the budget is *replenished* by (set to) e_s.
- The periodic server is *backlogged* if the aperiodic job queue is nonempty. It is *idle* elsewhere.
- The server is *eligible for execution* only when it is backlogged and has budget.
- When the server is scheduled and executes aperiodic jobs, it *consumes* its budget at the rate of one per time unit. When the budget becomes zero, the server is *exhausted*.

Polled Execution: Example

- Poller: (2.5, 0.5). According to RMA, it has the highest priority.



Bandwidth preserving server algorithms

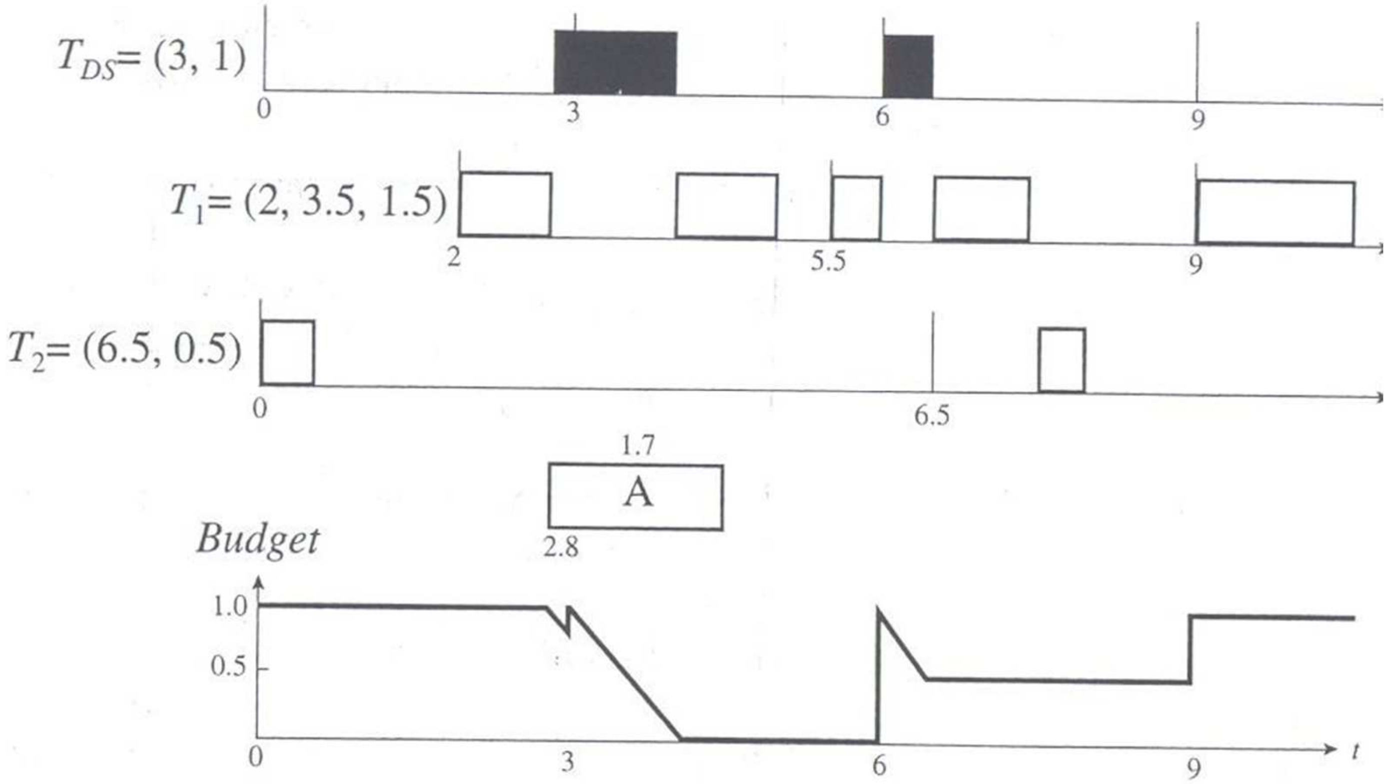
- Algorithms that improve the polling approach: the budget is *preserved* when the aperiodic job queue is empty; the poller is allowed to execute later in the period if any aperiodic job arrives.
- Assumptions:
  - A backlogged bandwidth-preserving server is ready for execution when it has budget. The scheduler suspends the server when it is exhausted or idle. The scheduler moves the server back to the ready queue once it replenishes the server budget if the server is backlogged.
  - The server suspends itself when it becomes idle. The scheduler puts the server back to the ready queue when it (the server) becomes backlogged again if it has budget.

Deferrable Servers

- The simplest of bandwidth-preserving servers.
- When no aperiodic job ready for execution, DS preserves its budget.
- <u>Consumption rule:</u> Budget is consumed at the rate of one per unit time.
- Replenishment rule: Budget is set to e_s (no accumulation) at instants kp_s, for $k = 0, 1, 2, \ldots$
- Example: $T_1 = (2.0, 3.5, 1.5)$, $T_2 = (6.5, 0.5)$, $T_{\mathrm{DS}} = (3, 1)$. RMA is used. A arrives at 2.8; $e = 1.7$.

Deferrable Servers - RMA

- T_1 = (2.0, 3.5, 1.5), T_2 = (6.5, 0.5), T_{DS} = (3, 1).



Deferrable Servers – RMA – Schedulability

- Theorem: A system of n independent, preemptable periodic tasks whose periods satisfy

$$p_s < p_1 < p_2 < \dots < p_n < 2p_s \text{ and } p_n > p_s + e_s$$

and whose relative deadlines are equal to their respective periods. This system is schedulable according to RMA with a DS (p_s, e_s) if

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

Deferrable Servers – RMA – Schedulability

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

- Example: T_1 = (3.0, 0.6), T_2 = (5.0, 0.5), T_3 = (7.0, 1.4), T_{DS} = (4, 0.8).
- The theorem can Not be applied.

Deferrable Servers – RMA – Schedulability

- T_i is a periodic task with a lower priority than the server.
- T_i is surely schedulable if $U_i + u_s + \frac{e_s}{p_i} \leq U_{RM}(i+1)$

where U_i is the total utilization of the i highest-priority tasks in the system.

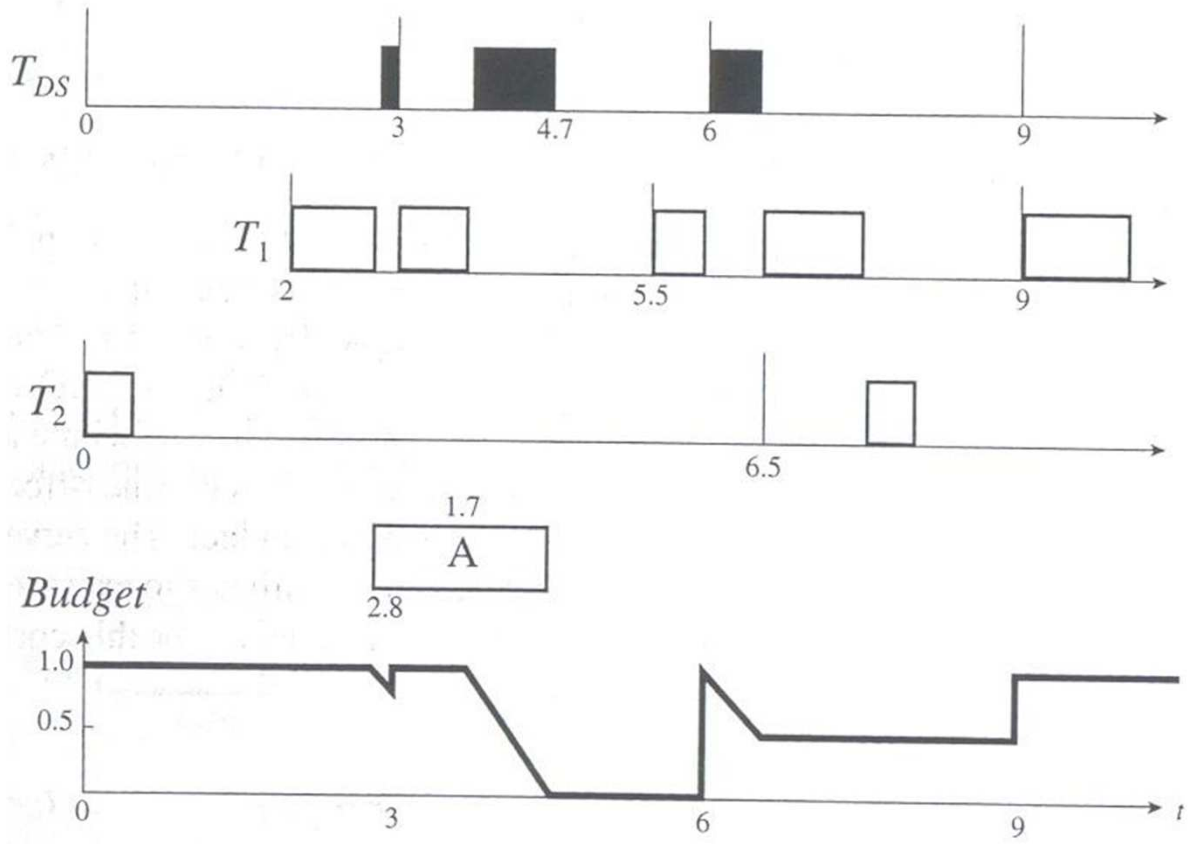
- Example: $T_1 = (3, 0.6)$, $T_2 = (5, 0.5)$, $T_3 = (7., 1.4)$, $T_{DS} = (4, 0.8)$.
- T_1 is not affected by the server(why?).

Deferrable Servers – RMA – Schedulability

- Example: T_1 = (3, 0.6), T_2 = (5, 0.5), T_3 = (7., 1.4), T_{DS} = (4, 0.8).
- T_2: The expression $U_2 + u_s + e_s/p_2$ is computed. Its value is then compared to $U_{RM}(3)$.
- $U_2 + u_s + e_s/p_2 = 0.66 < U_{RM}(3) \rightarrow T_2$ is schedulable.
- Redo the same computations for T_3. T_3 may not be schedulable (in fact, it is).

Deferrable Servers - EDF

- T_1 = (2.0, 3.5, 1.5), T_2 = (6.5, 0.5), T_{DS} = (3, 1).



Deferrable Servers – EDF - Schedulability

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

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

- In the previous example, all three tasks are schedulable.

Sporadic Servers

- DS may delay lower-priority tasks.
- Sporadic Servers (SS) rules ensure that each sporadic server (p_s, e_s) never demands more processor time than the periodic task (p_s, e_s).

Sporadic Server in Fixed-Priority Systems: Notations

- **T**: system of *n* independent, preemptable periodic tasks.
- $\mathbf{T}_H$: subset of periodic tasks with higher priorities than the server priority.
- **T**/ $\mathbf{T}_H$ are either busy or idle.
- Server *busy interval*: [an aperiodic job arrives at an empty queue, the queue becomes empty again].

Sporadic Server in Fixed-Priority Systems: Notations

- t_r: the latest (actual) replenishment time.
- t_f: the first instant after t_r at witch server begins to execute.
- t_e: the latest *effective* replenishment time.
- At any time t:
  - *BEGIN*: beginning instant of the earliest busy interval among the latest contiguous sequence of busy intervals of $\mathbf{T}_H$ that started before t.
  - *END*: end of the latest busy interval if this interval ends before t, infinity if the interval ends after t.

Simple Sporadic Server

- <u>Consumption Rules:</u> At any $t > t_r$, budget is consumed at the rate of 1 per unit time until budget is exhausted when
  - C1: the server is executing

  OR
  - C2: the server has executed since t_r and $END < t$.
- <u>Replenishment Rules:</u>
  - R1: Initially when system begins execution and each time when budget is replenished, budget = e_s and t_r = current time.
  - R2: At time t_f, if $END = t_f$ then $t_e = \max(t_r, BEGIN)$, if $END < t_f$ then $t_e = t_f$. Next replenishment time is $t_e + p_s$.
  - R3: a) If $t_e + p_s$ is earlier than t_f, budget is replenished as soon as it is exhausted.

    b) If **T** becomes idle before $t_e + p_s$ and becomes busy again at t_b, budget is replenished at $\min(t_e + p_s, t_b)$.