Chapter 7

Date: / /

Scheduling Aperiodic & sporadic Jobs in priority Driven system

Assumptions:

- Aperiodic and sporadic jobs are independent of each other, and of the periodic task.
- We assume that every job can be preempted at any time.
- The parameters of each sporadic job become known after it is released.
- sporadic jobs may arrive at any time, even immediately after each other.
- Their deadlines are arbitary
- some sporadic jobs may not be possible to meet their deadlines → reject the sporadic jobs that cannot complete in time or accept all sporadic task & allow some of them to complete late.
- In the absence of aperiodic and sporadic jobs, the periodic jobs meet deadlines.

Objective, correctness and optimality / system model

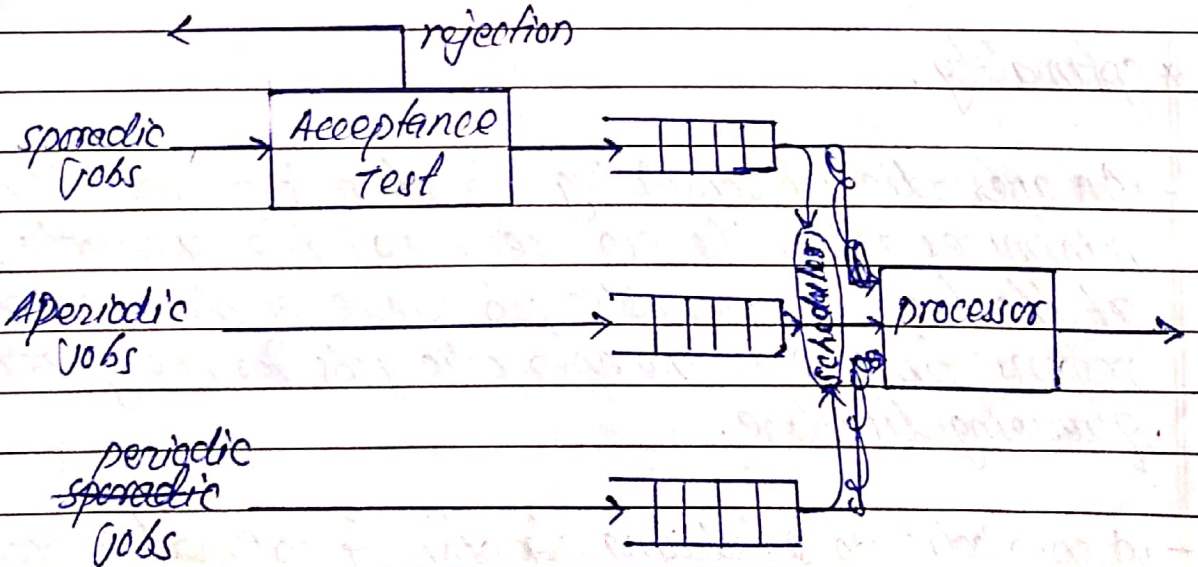


Fig: priority queue maintained by the operating system.

Date: / /

The operating system maintains priority queue of periodic, aperiodic and sporadic jobs.
- The scheduler selects the job at the head of queue and provides to the processor in order to execute it.
- The scheduler tries to complete each aperiodic task as soon as possible without causing periodic and sporadic job to miss their deadlines.
- Based on the execution time and deadline of each newly arrived sporadic jobs are decided as whether to accept or reject. when the job will complete with as thin its deadline, without causing any periodic task or previously accepted sporadic job to miss their deadline the job be accepted. If sporadic job cannot guarantee it will meet its deadline then it is rejected. This is done by acceptance test.

*Correctness:

An algorithm is correct if periodic and accepted sporadic jobs never miss their deadlines.

*optimality:

- An aperiodic job scheduling algorithm is optimal if it minimizes either the response time of the aperiodic job at the head of aperiodic job queue or the average response time of all the aperiodic jobs for the given queueing discipline.

- A sporadic job scheduling algorithm is optimal if accepts a new sporadic jobs and schedules the job to complete by its deadline, if and only if the new job can be correctly scheduled to complete in time.

Date: / /

Aperiodic Job scheduling

Four approaches:
- Background scheduling
- Interrupt-driven scheduling
- slack stealing
- Periodic server execution / polled execution

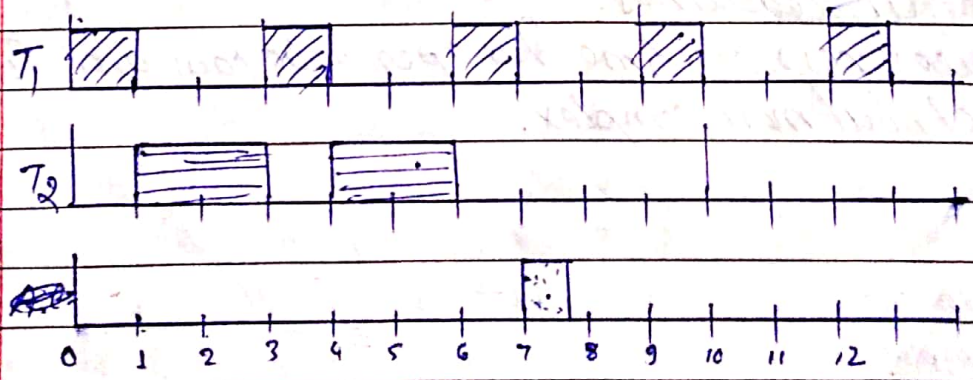
* Background scheduling of Aperiodic Jobs

In this approach, aperiodic jobs are executed in the background i.e. aperiodic jobs are scheduled and executed only at times when there is no periodic or sporadic job ready for execution.

- Periodic task (and accepted sporadic jobs) always meet deadline.
- simple to implement.

problems:
- Execution of aperiodic job may be delayed & long response time for the aperiodic jobs.

E.g.
Consider two periodic tasks $T_1 = (3, 1)$ & $T_2 = (10, 4)$
- tasks are scheduled rate monotonically.
- suppose than an aperiodic job A with execution time equal to 0.8 & released at time 0.1.



Response time of aperiodic job = 7.7 (7.8 - 0.1)
It is a worst case response time.

Date: / /

* Interrupt Driven scheduling of Aperiodic Jobs

- Execute the aperiodic jobs by interrupting the periodic jobs.
- Whenever an aperiodic job arrives, the execution of periodic task is interrupted, and the aperiodic job is executed.
- This approach reduces response times of aperiodic jobs but will often cause periodic/sporadic tasks in the system to miss some deadline.

* slack stealing for Aperiodic Jobs

- Postpone execution of periodic tasks only when it is safe to do so.
- Periodic jobs are be scheduled to complete before their deadline; there may be some slack time bet^n completion of the periodic job and its deadline.
- since we know the execution time of periodic jobs, can move the slack time earlier in the schedule, running periodic jobs 'just in time' to meet their deadline.
- Executes aperiodic jobs in the slack time, to meet their deadlines.
- Reduces response time for aperiodic jobs and is correct, but more complex.

Date: / /

Polled Approach : Polling server

A common way to schedule aperiodic job is using a polling server.

A periodic job (P_s, e_s) scheduled according to the periodic algorithm, generally as the highest priority job. P_s is its polling period and e_s is its execution time.

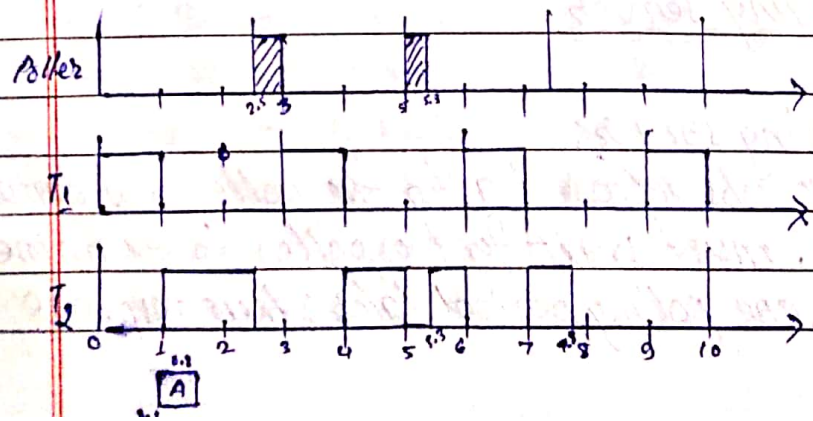
→ A periodic job (P_s, e_s) is a periodic task called as poller or polling server.

→ The poller becomes ready for execution periodically at integer multiple of P_s and scheduled together with periodic tasks in the system according to priority-driven algorithms. When it executes, it examines the aperiodic job queue:

- If the queue is nonempty, the poller executes the job at the head of the queue.
- The poller suspend its execution or is suspended by the scheduler either it has executed for e_s unit of time in the period or when aperiodic job queue becomes empty.

If is ready for execution again at the begining of the next polling period.

E.g. Consider a system with
Two periodic task T_1 (3, 1), T_2 (10, 4), aperiodic task A with $r = 0.1$ and $e = 0.8$ and a poller (2.5, 0.5) then the aperiodic task be scheduled along with scheduling of periodic task using RM



Response time of task A = 5.3 − 0.1 = 5.2

Date: / /

Periodic server

A task that behaves much like a periodic task, but is created for the purpose of executing aperiodic jobs, is called as periodic server. A periodic server, $T_{ps} = (P_{ps}, e_{ps})$ never executes more than e_{ps} units of time in any interval of length P_{ps} where e_{ps} is called execution budget of the periodic server. When a server is scheduled and executes aperiodic jobs, it consumes its budget at the rate of 1 per unit time and the budget has been exhausted when it reaches 0. A time instant when the budget is replenished is called replenishment time.

The periodic server is backlogged whenever the aperiodic job queue is ~~em~~ nonempty i.e. it is idle if the queue is empty. The server is eligible (i.e. ready) for execution only when it is backlogged and has budget (i.e. its budget is non zero).

A polling server is a simple kind of periodic server.
- The budget is replenished by e_s unit at the beginning of each period.
- The budget is immediately consumed if there is no work when the server is scheduled.

Bandwidth preserving server

Problems with polling server:

Aperiodic jobs released after the poller has found the queue empty, must wait for the poller to examine the queue again, one polling period later: their response

Date: / /

time is longer. BPS solves the problem of polling server
- Bandwidth preserving servers provide the condition under which its execution budget is preserved & consumed.
- Preserve the execution budget of the poller when it finds an empty queue and allow it to execute later in period if any aperiodic job arrives.
- Bandwidth preserving servers shortens the response time of some aperiodic jobs.

Bandwidth preserving servers are periodic servers with additional rules for consumption and replenishment of their budget. The working principle of this server are:
1) Ready for execution when it has budget
2) Suspends the server when budget is over
3) Scheduler moves server back to queue once it replenishes its budget.
4) If arrival of an aperiodic job causes the server to become backlogged, and it has budget, the server is put back on the ready queue.

Algorithms that ~~provide~~ improve the polling approach by preserving the execution budget of the poller are called bandwidth preserving server algorithms.

Different types of bandwidth preserving servers are:
- Deferrable servers
- sporadic servers
- Constant utilization servers
- Total bandwidth servers
- Weighted fair queuing servers

Date: / /

Deferrable server

→ A deferrable server is the simples of bandwidth-preserving servers.
→ The execution budget of a deferrable server with period P_s and execution budget e_s is replenished periodically with period P_s.
→ Unlike a poller, however, when a deferrable server finds no aperiodic jobs ready for execution, it preserves its budget throughout the period.

Operation of deferrable servers:

The consumption and replenishment rules that define a deferrable server $T_{DS}(P_s, e_s)$ are:

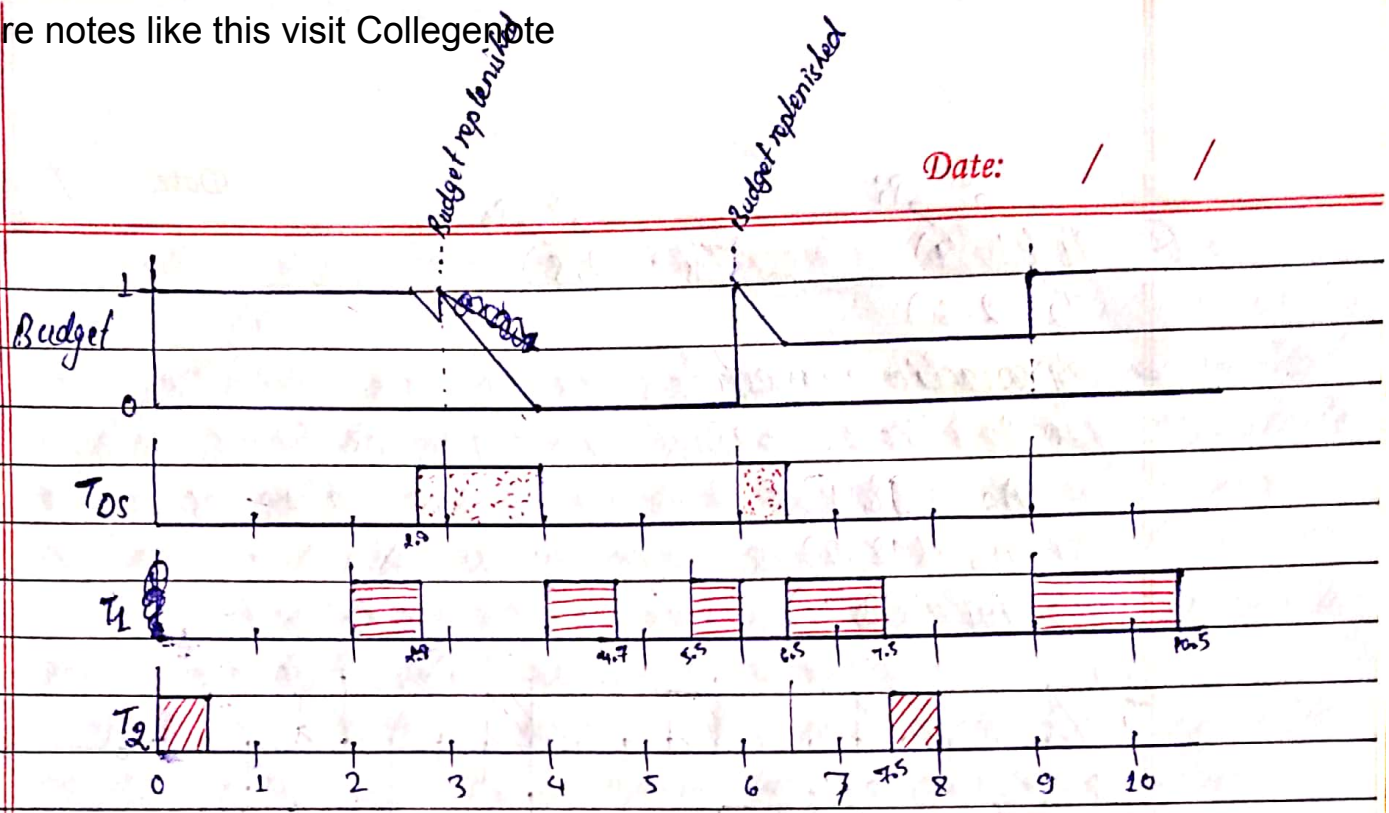
- **Consumption Rule:**
  - The execution budget of the server is consumed at the rate of one per unit time whenever the server executes
  - Unused budget is retained throughout the period, to be used whenever there are aperiodic jobs to execute.

- **Replenishment Rule:**
  The execution budget of server is set to e_s at time instant kP_s, for $k = 0, 1, 2, \ldots$ (At multiple of period)

Example

Consider a system with two independent periodic tasks T_1 ($\phi = 2$, $P = 3.5$, $e = 1.5$) and T_2 ($P = 6.5$, $e = 0.5$) with aperiodic job A ($r = \cancel{2.5}\ 2.9$, $e = \cancel{2.5}\ 1.7$) be executed in deferrable server $T_{DS} = (P_s = 3, e_s = 1)$.
- Periodic task & server are scheduled rate-monotonically

Date: / /



- At time 0, the server is given 1 unit of budget. The budget stays at unit until time 2.8. When A arrives, the deferrable server executes the job. Its budget decreases as it executes.
- Immediately before the replenishment time 3.0, its budget is equal to 0.8 (since 0.2 is consumed by aperiodic job A). This 0.8 unit is lost at time 3.0, but the server acquires a new unit of budget. Hence server continues to execute.
- At time 4.0, its budget is exhausted. The server is suspended, and the aperiodic job A waits.
- At time 6.0, its budget replenished, the server resumes to execute A.
- At time 6.5, job A completes. The server ~~still~~ still has 0.5 unit of budget. Since no aperiodic job waits in the queue, the server suspend itself holding this budget.

Date: /

(2) T_1 (1, 4) T_{DS} = (2, 5)

T_2 (2, 6)

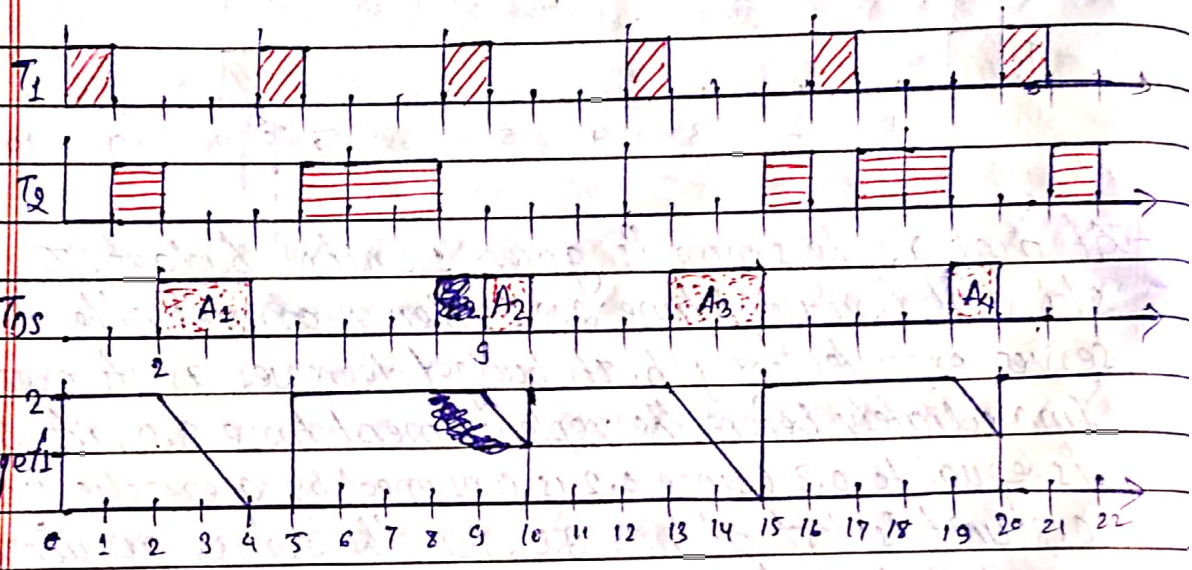
Aperiodic requests:

A_1 (r = 2, e = 2)

A_2 (8, 1)

A_3 (12, 2)

A_4 (19, 1)



* Limitation of Deferrable servers:

In first e.g.

T_1 blocked for 1.2 units (2.8 - 4) although execution time of deferrable server is only 1.0 units (with period 3 units)

- They may delay lower-priority tasks for more time ~~than periodic~~ than a periodic task with the same period and execution time.

Date: / /

Sporadic server in fixed priority system

A different types of bandwidth preserving server that is designed to eliminate the limitation of deferrable server is called sporadic server. It has more complex consumption and replenishment rules ensure that a sporadic server with period P_s and budget e_s never demands more processer time time than a periodic task with the same parameters.

Consider a system T of N independent preemptable periodic tasks and a single ~~periodic~~ sporadic server ~~task~~ with parameter (P_s, e_s) then

- server has an arbitary priority π_s.
- T_H is the subset of periodic task with higher priorities than the server.
- t_r denotes the latest (actual) replenishment time.
- t_f denotes the first instant after t_r at which the server begins to execute.
- t_e denotes the latest effective replenishment time.

At any time t define:

- BEGIN as the ~~seed~~ start of the earliest busy interval in the most recent contiguous sequence of busy intervals of T_H starting before t.
- END as the end of the latest busy interval in this sequence if this interval ends before t; define END $= \infty$ if the interval ends after t.

The consumption and replenishment rules be stated as

Date: / /

Consumption Rule:

- At any time t after t_r, server's budget is consumed at the rate of 1 per unit time until the budget is exhausted when either one of following two conditions is true:
  - C_1 : The server is executing
  - C_2 : The server has executed since t_r and END $< t$.
- When they are not true, the server holds its budget.

Replenishment Rule:

There are three rules

R_1: When system begins execution and each time when the budget is replenished, set the budget to e_s and t_r = current time.

R_2: When server begins to execute i.e. at time t_f do

If END = t_f then

$t_e = \max(t_r, \text{BEGIN})$

else if END $< t_f$ the

$t_e = t_f$

The next replenishment time is set to $t_e + P_s$.

R_3 : The next replenishment occurs at the next replenishment time ($t_e + P_s$) except under the following conditions

1. If $t_e + P_s$ is earlier than t_f, the budget is replenished as soon as it is exhausted.
2. If T becomes idle before $t_e + P_s$ and becomes busy again at t_b, the budget is replenished at min($t_e + P_s$, t_b

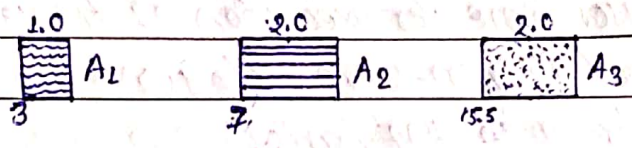
Date: / /

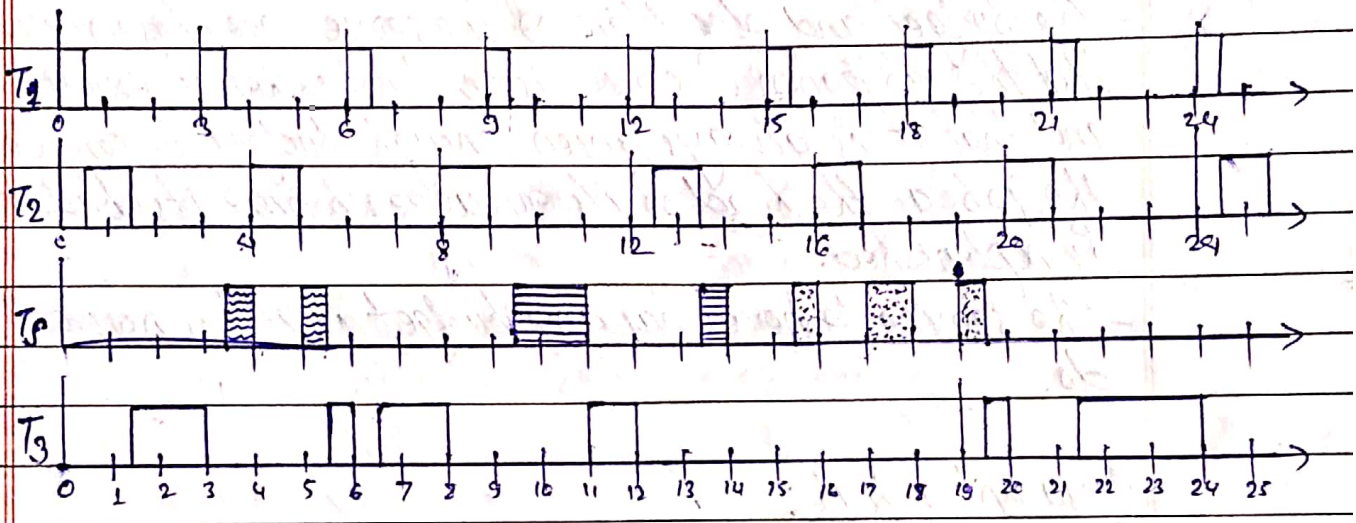
Eg.

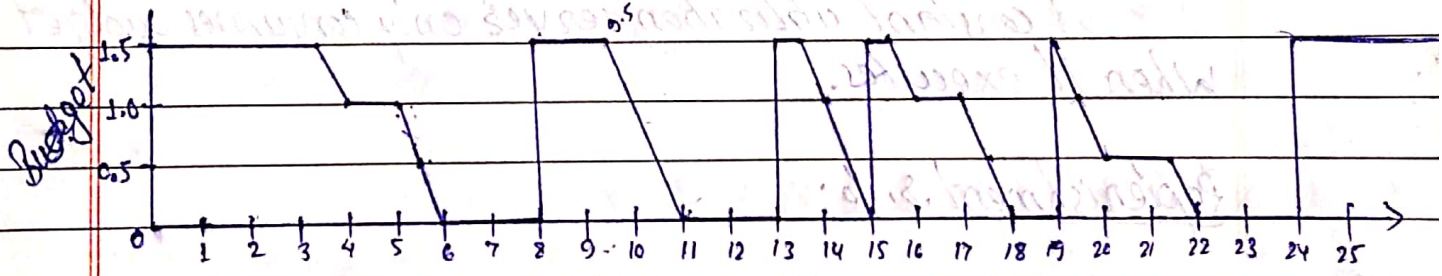
Consider, the budget of the server $(5, 1.5)$ is 1.5 initially. It is scheduled rate-monotonically with three periodic tasks: $T_1 = (3, 0.5)$, $T_2 = (4, 1.0)$, and $T_3 = (19, 4.5)$. They are schedulable even when the aperiodic job queue is busy all the time.

$T_1 > T_2 > T_S > T_3$

$T_4 = (T_1, T_2)$







Date: / /

Constant utilization server

- The constant utilization server is based on the size of the server.
- A constant utilization server reserves a known fraction U_s of the processor time for execution of the server.
- When the budget is non-zero, the server is scheduled with other tasks on an EDF basis.
- The budget and deadline of the server are chosen such that the utilization of server is constant when it executes, and that it is always given enough budget to complete the job at the head of its queue each time its budget is replenished.
- The server never has any budget if it has no work to do.

Consumption rule:

A constant utilization server only consumes budget when it executes.

Replenishment rule:

- Initially, set budget $e_s = 0$ and deadline $d = 0$
- When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue
  - If $t < d$, do nothing
  - If $t \geq d$ then set $d = t + e/U_s$ and $e_s = e$
- At the deadline d of the server
  - If the server is backlogged, set $d = d + e/U_s$ and $e_s = e$
    ⇒ was busy when job arrived
  - If the server is idle, do nothing.

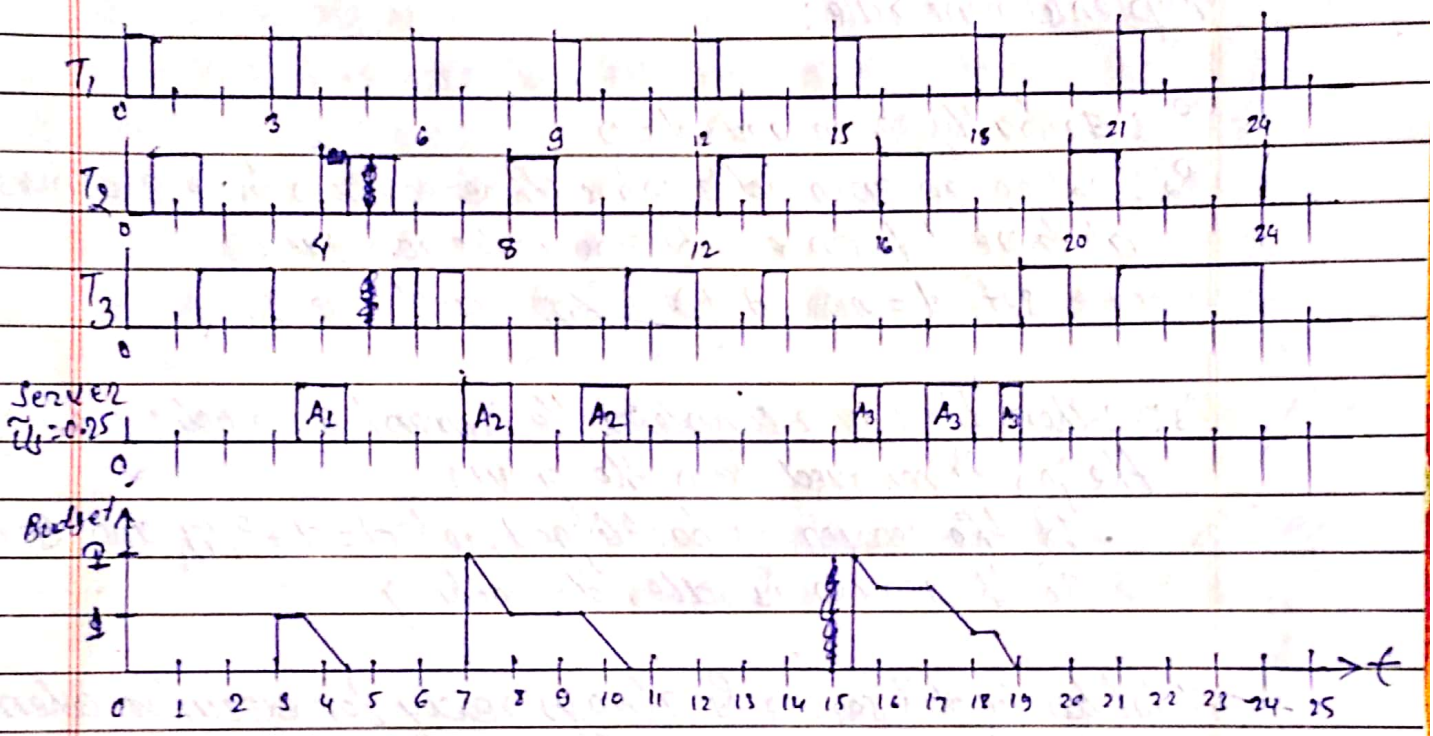
Date: / /

E.g.

$T_1 = (3, 0.5)$, $T_2 = (4, 1)$, $T_3 = (19, 4.5)$ & $\tilde{U}_s = 0.25$

Aperiodic jobs $A_1 = (3, 1)$, $A_2 = (6.9, 2)$ & $A_3 = (15.5, 2)$

(r, e)



Total Bandwidth server

- A constant utilization server gives a known fraction of processor capacity to a task but cannot claim unused capacity to complete the task earlier.
- A total bandwidth server improves responsiveness by allowing a server to claim background time not used by the periodic task. This is achieved by having the scheduler replenish the server budget as soon as the budget is exhausted if the server is backlogged or as soon as the server becomes backlogged.

Date: / /

Consumption rule:
A total bandwidth server only consumes budget when it executes.

Replenishment rule:

R_1: Initially $e_s = 0$ and $d = 0$
R_2: When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue
set $d = \max(d, t) + e/U_s$ and $e_s = e$

R_3: When the server completes the current aperiodic job, the job is removed from the queue
- If the server is backlogged, set $d = d + e/U_s$ and $e_s = e$
- If the server is idle, do nothing

- Total bandwidth server is always ready for execution when backlogged.

Weighted Fair Queuing server

- The constant utilization server and total bandwidth server are used to assign some fraction of processor to an aperiodic task. During the assignment of task, there is an issue of fairness and starvation.
- A scheduling algorithm is fair within any bandwidth time interval if the fraction of processor time in the interval obtained by each backlogged server is proportional to the server size.

Date: / /

The ~~weed~~ wailed fair queuing algorithms are used to share processor time bet^n server and designed to ensure fairness is allocating among multiple servers.

Consumption rule:

It consumes budget when it executes.

Replenishment rule:

- Its budget is replenishment when it first becomes backlogged after being idle.
- As long as it is backlogged, its budget is replenished each time it completes a job
- At each replenished time, the server budget is set to the execution time of the job at the head of queue.

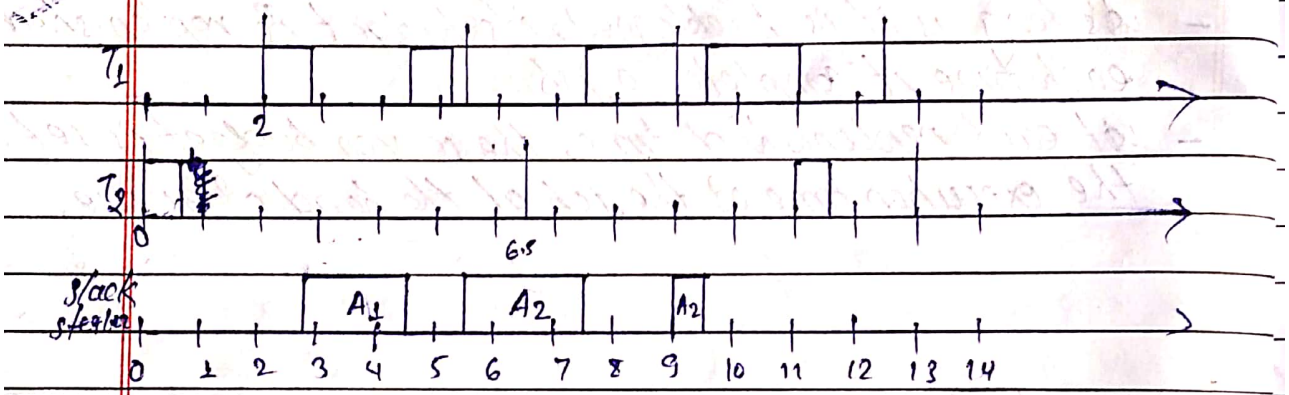
Slack stealing in deadline driven system

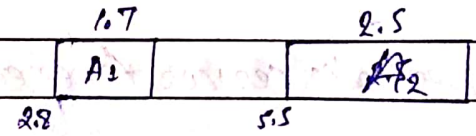
- The slack stealer is a periodic server to execute aperiodic job using slack time.
- The slack stealer is ready for execution whenever the aperiodic job queue is nonempty and is suspended when the queue is empty.
- The scheduler monitors the periodic task in order to keep track of the amount of available slack.
- It gives ~~the highest~~ the slack stealer the highest priority whenever there is a slack and the lowest priority whenever there is no slack.
- When the slack stealer executes, it executes the aperiodic

Date: / /

job at the head of the aperiodic job queue.
- The available slack is always used if there is an aperiodic job ready to be executed.

e.g.
A system of two periodic tasks, $T_1 = (2.0, 3.5, 1.5)$ and $T_2 = (6.5, 0.5)$. In addition to the aperiodic job that has execution time 1.7 & is released at 2.8, suppose another aperiodic job with execution time 2.5 is released at time 5.5. These jobs are A_1 and A_2, respectively.





Slack stealing in ~~deadline driven~~ fixed-priority system

In principle, slack stealing in a fixed-priority system works in the same way as slack stealing in a deadline-driven system. However the computation & usage of

Date: / /

the slack are both more complicated in fixed-priority system.

Consider, the system contains three periodic tasks $T_1 = (3, 1)$, $T_2 = (4, 1)$ & $T_3 = (6, 1)$

→ (pdf)

Real time performance of jobs with soft timing constraints

- Traffic model
- Leaky Bucket model