**Comparative Study of Various Bandwidth Preserving Servers**

Types of Jobs in a REAL-TIME SYSTEM:

Periodic:

The real-time task that is repeated after a certain time interval is known as periodic real-time task. Basically the periodic real-time tasks are controlled by the clock interrupts. Therefore, the periodic real-time tasks are also called clock-driven tasks. The time interval after which the task repeats itself is known as the period of the task.

For example, when a flight is detected by the radar and until the radar exists, the radar signal zone is an example of periodic real-time task.

Aperiodic:

The real-time task that occurs at any random time is known as a periodic real-time task. Between two aperiodic real-time tasks the time interval may be even zero. Soft real-time tasks are generally aperiodic real-time tasks. It is also possible that these tasks may occur frequently or there might be a large time interval between two aperiodic real-time tasks.

For example, typing on the key-board is an aperiodic real-time task.

Sporadic:

The real-time tasks that reoccur at any random instant and have hard deadlines are known as sporadic real-time tasks. Sporadic real-time tasks have some similarity to aperiodic real-time tasks but these tasks are different from aperiodic real-time tasks. Basically all the high critical tasks are sporadic tasks.

For example, fire handling tasks in industry or emergency message arrival in systems are sporadic real-time tasks.

Applications:

Periodic tasks: Suitable e.g. to applications where it is required sampling regularly a given physical entity (e.g. a temperature, pressure, torque, speed), or actuate regularly on the system via an actuator.

Aperiodic+sporadic tasks: Suitable to scenarios where the event activation instants cannot be forecast, e.g. alarms, human-machine interfaces, external asynchronous interrupts.

Hybrid systems Applications which contain both types of tasks.

Different Bandwidth Preserving Servers:

Deferrable Server:

A deferrable server is the simplest bandwidth-preserving server that improves the response time of the aperiodic jobs as compared to a polling server.

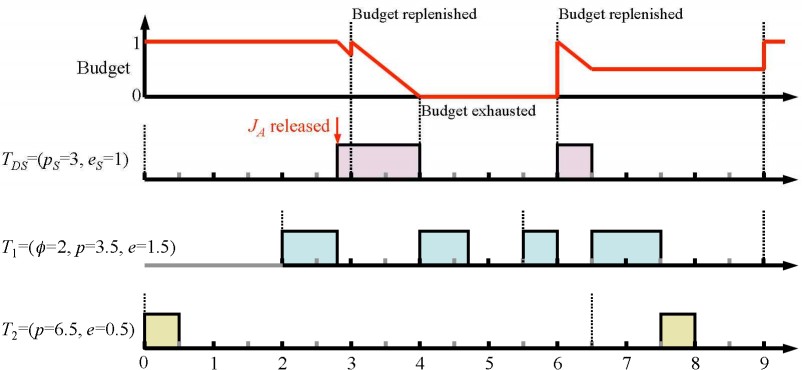
Rules of consumption:

* The budget 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.
* If no aperiodic job is available to execute at start of period then keep it instead of discarding with the hope of new arrival of new aperiodic job.

Rules of replenishment:

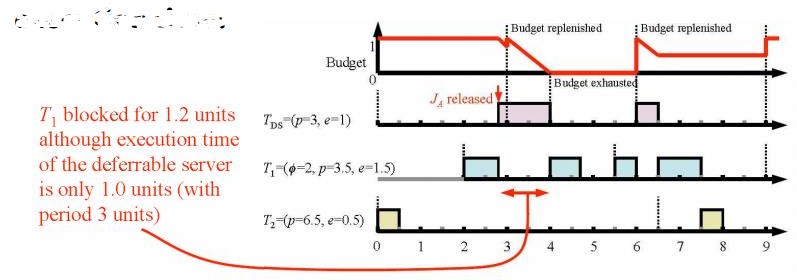
The capacity is replenished at the beginning of each period.

For example: consider a system with two independent periodic tasks T1 (F = 2, P = 3.5, e = 1.5) and T2 (P = 6.5, e = 0.5) with aperiodic jobs A (r=2.75, e = 1.75) be executed in deferrable server TDS = (Ps = 3, es =1). The RM schedule of periodic task and execution of aperiodic task on deferrable server is:



Limitations:

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



Sporadic Server:

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

Consumption rule

1. 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.

C1: The server is executing

C2: The server has executed since t r and END < t

1. When they are not true, the server holds its budget Replenishment rules

R1: When system begins executing and budget is replenished as:

Budget = e S and t r = the current time.

R2: When server begins to execute i.e. at time tf do if END = tf then

t e = max(t r, BEGIN ) else if END < tf then t e = tf

The next replenishment time is set to t e + p S.

R3: The next replenishment occurs at the next replenishment time (= t e + p S), except under the following conditions:

1. If t e + pS is earlier than tf the budget is replenished as soon as it is exhausted
2. If T becomes idle before t e + pS, and becomes busy again at t b, the budget is replenished at min(t b, t e + pS)

Constant Utilization Server:

The constant utilization server is based on the size of the server.It reserves a known fraction (ũs) of the processor time for execution of the server.When the budget is non-zero then 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 the 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.

Then the consumption and replenishment rule of constant utilization server is: Consumption Rule:

* A constant utilization server only consumes budget when it executes.

Replenishment Rules:

* 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 then
* If t < d, do nothing (⇒ server is busy; wait for it to become idle)
* If t ≥ d then set d = t + e/ũ s and e s = e

-At the deadline d of the server

* If the server is backlogged, set d = d + e/ũ s and e s = e ⇒ was busy when job arrived
* If the server is idle, do nothing

i.e. the server is always given enough budget to complete the job at the head of its queue, with known utilization, when the budget is replenished.

For example: ũ s = 0.25 T1 = (3, 0.5), T2 = (4, 1), T3 = (19, 4.5)

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 tasks.This is done by having the scheduler replenish the server budget as soon as the budget is exhausted if the server is backlogged at the time or as soon as the server becomes backlogged.

## Consumption Rule:

* A total bandwidth server only consumes budget when it Replenishment Rule:
* Initially, e s = 0 and d = 0
* When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue

Set d = max(d, t ) + e/ũ s and e s = e

* When the server completes the current aperiodic job, the job is removed from the queue then
* If the server is backlogged, set d = d + e/ũ s and e s = e
* If the server is idle, do nothing

Total bandwidth server is always ready for execution when backlogged and assigns at least fraction ũ s of the processor to a task foe execution.

For example: ũ s = 0.25 T1 = (3, 0.5), T2 = (4, 1), T3 = (19, 4.5)

Weighted Fair Queuing Server:

The constant utilization server and total bandwidth server are used to assign some fraction of processor capacity to an aperiodic task. During assignment of task, there is an issue of fairness and starvation i.e. the fairness cannot be maintained in case of total bandwidth and constant utilization server approach. A scheduling algorithms is fair with in any particular time interval if fraction of processor time in the interval obtained by each backlogged server is proportional to size of server.In such system not only all the task meet their deadlines, but they all make a continuous process according to their share of processor and there is no starvation. The weighted fair queuing algorithms are used to share processor time between servers and designed to ensure fairness is allocation among multiple servers.

Consumption Rule:

It consumes the budget when it is executed. Replenishment Rule:

* Its budget is replenished 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 the queue.