FEASIBILITY ANALYSIS

In order to determine the feasibility of a periodic task-set under the presence of a Deferrable Server two methods can be used: a "bound" LeastUpperBound (LUB) one and an "exact" ResponseTimeAnalysis (RTA) one.

Bound method:
The following formula can be used to calculate a LUB for the "bound" method:

$U_p \leq n(K^{1/2} - 1)$

and $K = \frac{U_s+2}{2U_s+1}$

or the following formula can be used for a tighter hyperbolic bound:

$$\prod_{i=1}^{n}(U_i+1) \leq \frac{U_s+2}{2U_s+1}$$

Where:
U_p: utilisation of periodic task-set
U_s: utilisation of server
U_i: utilisation of individual task
n: number of periodic tasks

If the above inequalities hold, the periodic task-set is certainly schedulable. If not, an "exact" RTA can be tried instead.

Note: Different bounds are derived based on the relations between the periods of the highest-priority and lowest-priority tasks in the periodic task-set and that of the server. The above bounds have been derived for a general case where the server can execute three times ($3C_s$) during one period of the highest-priority periodic task. For the formulas of other period relations, please see [1].

Exact method:
The following recurrent formula can be used to calculate exact response times of the periodic task-set. Note, this formula needs to be applied against each task.

The initial response time for the current task is the following:

$$R(0) = 2C_s + \sum_{j=1(j\in hp)}^{k} C_j \left\lceil \frac{T_s}{T_j} \right\rceil$$

And each recurrent step can be calculated by the following formula:

$$R(l+1) = C_s\left(1+\left\lceil \frac{(R_l - T_s)}{T_s} \right\rceil\right) + \sum_{j=1(j\in hp)}^{k} C_j \left\lceil \frac{R_l}{T_j} \right\rceil$$

Where:
C_s: total server cost (its capacity)
T_s: server period
hp: set of the tasks having a higher priority than the task currently calculating
C_k: cost of a higher priority task
T_k: period of a higher priority task

If an $l \geqslant 1$ exists for which $R(l) > T_k$ then the task-set is not schedulable.

If however, an $l \geqslant 1$ exists for which $R(l) = R(l+1) \leq T_k$ exists then the task-set is certainly schedulable.

SIZING

The following formula can be used to calculate the maximum utilisation of the server that will guarantee the feasibility of the periodic task-set:

$U_{s(max)} = \frac{2-P}{2P-1}$

Where:

$P = \prod_{i=1}^{n} (U_i + 1)$

n: number of periodic tasks

The analysis assumes that the Deferrable Server task will run at the highest priority in the system and since priorities are assigned rate-monotonically, the maximum period the server can have must be equal to the shortest period in the periodic task-set.
So, if T_1 is the period of the highest-priority task in the periodic set then:
$T_{s(max)} = T_1$
and the server cost can be calculated as per:
$C_s = U_s T_s$

In theory, setting $T_s = T_1$ therefore $P_s = P_1$ (server priority equal to highest priority task) would work fine so long as ties are broken in favour of the server. In practice though, a higher priority will need to be given to the server task.

It is also advisable to set the period of the server as large as possible (assuming same or greater utilisations can be achieved with this value when compared with smaller values). See, section on tuning suggestions too.

TUNING SUGGESTIONS

The following are some suggestions that can come in handy when trying to maximise the effectiveness of a Deferrable Server:

- The typical Liu/Layland assumptions for the periodic task-set must be in place (deadline equal to period, tasks are independent and ready to run at the beginning of their period).
- Priorities should be assigned rate-monotonically and the server should be assigned the highest priority in the system so that it does not suffer any interference from other tasks.
- The server capacity should be large enough so that a burst of aperiodic events can be processed successfully.
- The largest the period of the server, the longer the capacity of the server can be retained to service aperiodic events. For example, considering two configurations with the same utilisation (C_s=1,T_s=2) and (C_s=2,T_s=4)it is obvious that the latter can serve two units at time t = 2 whereas the former would be able to serve only one. This is also an advantage over other solutions like a Polling Server which has a very strict window of service (the beginning of it's period plus its capacity). This property should, therefore, be exploited by setting $T_s = T_1$ (assuming equal or higher utilisation can be achieved when compared with smaller period values).

The server comprises two components: the Handler (a BoundAsyncEventHandler) and the Replenisher (a periodic RealtimeThread).

The Replenisher should be assigned the highest priority in the system, the Handler the second highest and the next highest should be given to the first task having the shortest period in the periodic task-set. So, if P_r, P_h and P_{t1} are the priorities of the Replenisher, the Handler and the first task in the set then the following relation should hold:

$$P_r < P_h < P_{t1}$$

Based on experimental evidence [1/section V], the servers that demonstrated the best performance (when compared with an ideal queuing model) under different periodic and aperiodic loads, were the ones that had been sized using the “exact” RTA method rather than a “bound” LUB one . Therefore, using “exact” servers where possible may be more beneficial.

BLOCKING FACTORS

Since the Replenisher and the Handler have the highest priorities in the system and don’t synchronise in any way with any other task in the periodic task-set, they can’t suffer direct or push-through blocking. The Replenisher however does synchronise with the Handler (via the Handler object itself) and can incur some direct blocking which may result in longer response times. The methods with which it synchronises in the DeferrableServerEventHandler class are:

- DeferrableServerEventHandler::replenishBudget
- DeferrableServerEventHandler::canProcess
- DeferrableServerEventHandler::adjustForNextRun

If t_{rb}, t_{cp}, t_{afnr} are the lengths of the critical sections of the above methods respectively, the maximum blocking that the Replenisher can suffer is:

$$B_{R(max)} = \max(t_{rb}, t_{cp}, t_{afnr})$$

REFERENCES

[1] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. IEEE Trans. Comput.,44(1):73-91, 1995.
[2] G. C. Buttazzo. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms And Applications, volume 23 of Real-Time Systems Series. Springer Verlag, third edition, LLC 2011.