Constant Bandwidth Server

Lochana Abhayawardana
dept. Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
lochana.abhayawardana@stud.hshl.de

***Abstract*—(STILL IN PROGRESS)**

***Index Terms*—modification, CBS, TBS**

I. Introduction

In real-time systems, a CBS(Constant Bandwidth Server) is a scheduling algorithm which is used to ensure temporal isolation between tasks that share system resources, such as the CPU. CBS is a scheduling policy that allocates a server to a task and ensures that the task is not exceeding the assigned bandwidth. The CBS assigns a certain amount of time, calls as the budget, to a task for each server period. CBS also assigns a fixed priority to each task[1]

II. Resource reservation in dynamic real-time systems

Resource reservation is a method used in Real-Time systems to assure that the task is executed within its allocated time limits. This method allocates a fraction of the processor bandwidth to each task considering each timing requirement. However, the kernel then prevents each task from consuming more than its allocated bandwidth to protect other tasks in the system.

A way to implement resource reservation is to reserve a task-specific amount of CPU time at every interval. If a task is assigned a pair (Q_i and P_i), Where Q_i is the amount of CPU time allocated to a task and P_i is the interval in which the task is executed. The task can execute for the Q_i unit for each P_i time, But if the task is not finished, It assigns another time Q_i at the beginning of the next time period and schedules it as a real-time task and the budget expires.

III. Efficient reclaiming in reservation-based real-time systems with variable execution times

In Real Time systems with periodic and aperiodic tasks efficiently utilizing the spare time is ensured using the DPS server to reclaim spare time from the periodic task and add it to the corresponding aperiodic capacity, Which allows for immediate service of aperiodic requests.

The reclaiming mechanism is shown in Figure 01. And it illustrates how the sphere time can be replied to from the periodic tasks and added to the aperiodic capacity. At the completion time of the first to periodic instance the corresponding aperiodic capacity of implemented by an amount equal to the spare time saved. This mechanism allows the first aperiodic request to receive immediate service for all seven units of time required, which can replicate as completing at the time `t=11` instead of `t=12`. However, reclaiming the spare time for periodic tasks as aperiodic capacities does not affect the schedulability of the system. Because any sphere time is already allocated to a priority level corresponding to its deadline when the task has been guaranteed. Therefore, the spare time can be safely used if requested within the same deadline.[1]

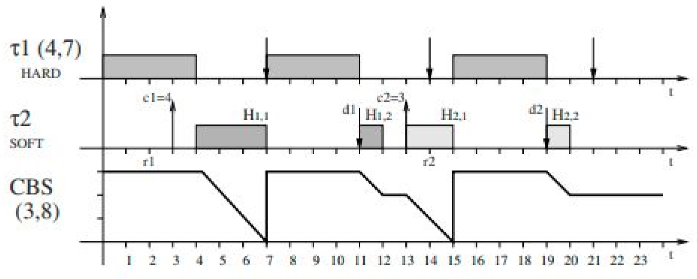


Fig. 1. An example of CBS scheduling

IV. The CBS algorithm and its implementation

When describing the CBS algorithm, let a_k and d_k be the release time and the deadline of the kth chunk generated by the server, and let `c` and `n` be the actual server budget number of the pending requests in the server queue. These variables are initialized as follows:

d_o = 0 , c = o, n = 0, k = 0.

Following the above notations the server behaviour can be described by the algorithm shown below in Figure.[2]

V. Handling overruns and implementing resource reservations with CBS

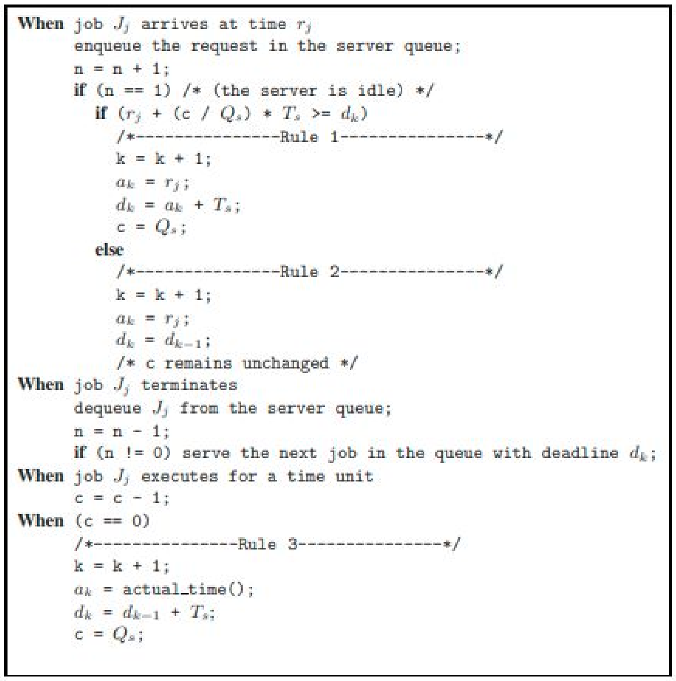
STILL IN PROGRESS

VI. The use of CBS for serving self-suspending tasks

STILL IN PROGRESS

VII. Comparison of CBS with other service mechanisms

STILL IN PROGRESS



```
When job $J_j$ arrives at time $r_j$
      enqueue the request in the server queue;
      n = n + 1;
      if (n == 1) /* (the server is idle) */
         if ($r_j$ + (c / $Q_s$) * $T_s$ >= $d_k$)
            /*---------------Rule 1---------------*/
            k = k + 1;
            $a_k$ = $r_j$;
            $d_k$ = $a_k$ + $T_s$;
            c = $Q_s$;
         else
            /*---------------Rule 2---------------*/
            k = k + 1;
            $a_k$ = $r_j$;
            $d_k$ = $d_{k-1}$;
            /* c remains unchanged */
When job $J_j$ terminates
      dequeue $J_j$ from the server queue;
      n = n - 1;
      if (n != 0) serve the next job in the queue with deadline $d_k$;
When job $J_j$ executes for a time unit
      c = c - 1;
When (c == 0)
      /*---------------Rule 3---------------*/
      k = k + 1;
      $a_k$ = actual_time();
      $d_k$ = $d_{k-1}$ + $T_s$;
      c = $Q_s$;
```

Fig. 2. The CBS algorithm

A. comparison with Total Bandwidth Server(TBS)

STILL IN PROGRESS

VIII. Figures and Tables

TABLE I
TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

References

[1] Buttazzo, Giorgio, "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications - Second Edition, Springer 2005", pp.189-200 , January 2011.