Constant Bandwidth Server

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

***Abstract*—The constant Bandwidth Server(CBS) is a scheduling algorithm used in real-time systems to ensure temporal isolation and effective resource allocation among tasks. This article focused on the introduction, implementation, and real-world uses of CBS and a comparison between similar algorithms.**

***Index Terms*—CBS, scheduling algorithm, real-time systems**

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(Q_i = 2 In figure 1) 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.

A. Uppaal implementation of the scenario

This simple Uppaal model explains the scenario of assigning additional time in order to execute the task. The variable `x` is assigned as the clock of the system (figure 01). When the system starts the clock value is 0 and it gradually increases until `x=2` or Q_i=2. As soon as the clock counter reaches 2, the synchronization function, `extend_deadline` triggers the function in Figure 02. Then the state of Figure 02 moves from idle to execution state. The task stays in the execution state for 2 clock units which is representing the execution of the task within the specified time frame. After that specified time (2 clock units) the update `x:=0`, resets the clock. This process will repeat until the whole task is finished executing.

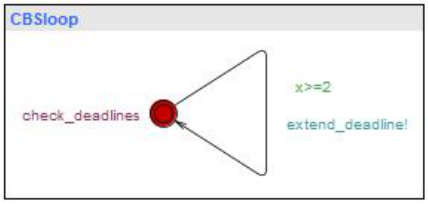

Fig. 1. Assigning time

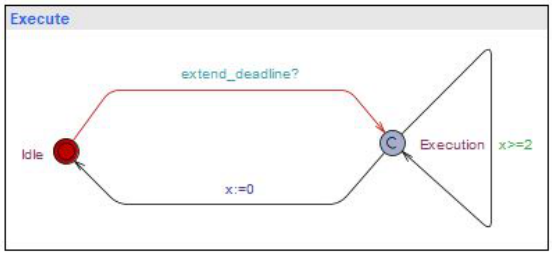

Fig. 2. Execution of a process

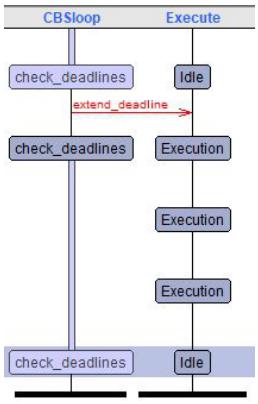

Fig. 3. Simulation of the resource reservation

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 or Deferrable 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.

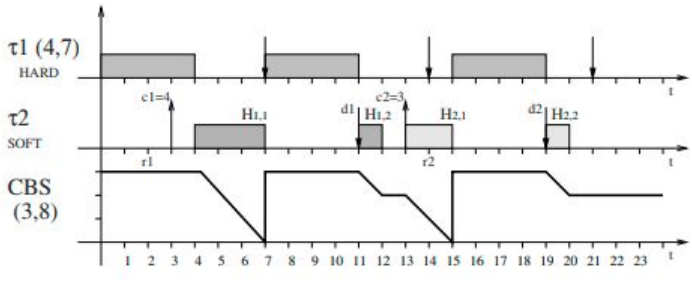

Fig. 4. Temporal protection using CBS mechanism [1]

The reclaiming mechanism is shown in Figure 04. 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]

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 k[th] 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 05

```
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. 5. The CBS algorithm [1]

V. Handling overruns and implementing resource reservations with CBS

Overrun conditions are caused by tasks that execute more than expected or are activated more frequently than expected. This can happen because the task parameter is incorrectly estimated and calculated. If the overruns are not properly handled, task overruns can cause critical issues in real-time systems.

In order to cope with handling overruns, resource reservations use as the solution. It limits the effects of overruns in tasks with variable computation times. In this method, each task is assigned a fraction of the processor bandwidth to satisfy its timing constraints. The kernel must prevent each task from consuming more than the requested amount to protect other tasks (temporal protection) [1].

To implement temporal protection, each task T_i with variable computation time should be handled by a dedicated CBS with bandwidth U_{s_i}, that can not interfere with the rest of the tasks for more than U_{s_i}. Figure 3 shows an example of two tasks (T_1 and T_2) served by the two dedicated CBSs with a bandwidth of U_{s_1} = 0.15 and U_{s_2} = 0.1, a group of two tasks (T3, T4) is handled by a single CBS with the bandwidth U_{s_3} = 0.25, and three hard periodic tasks (T_5, T_6, T_7) are directly scheduled by the EDF, without server intercession, because their execution times are not subject to larger variations. The example the total processor bandwidth is shared among the task as shown in Figure 07.

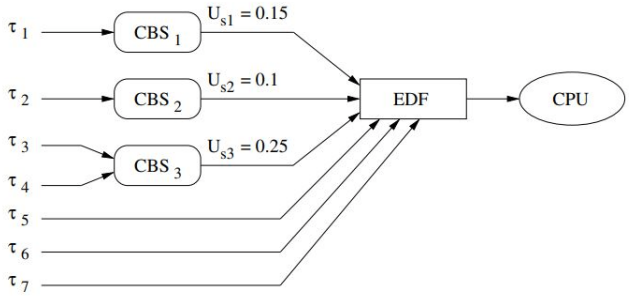

Fig. 6. Temporal protection using CBS mechanism [1]

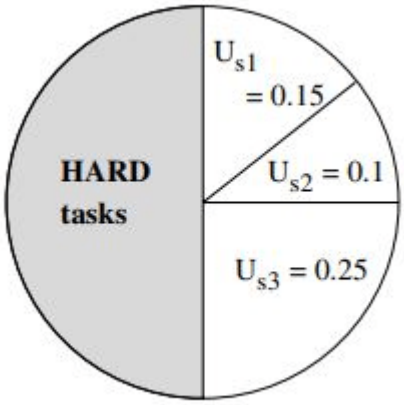

Fig. 7. Bandwidth allocation for set of tasks [1]

The properties of the CBS guarantee that the set of hard periodic tasks (with utilization Up) are schedulable by EDF only if,

$$U_p + U_{s_1} + U_{s_2} + U_{s_3} \leq 1$$

If the above condition holds the set of hard periodic tasks always use fifty percent of the processor independent of the execution time of the other tasks. [1] The CBS can be updated to enforce hard reservations by postponing the budget replenishment to the server deadline.

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

Self-suspending tasks are a specific type of real-time tasks that can voluntarily suspend their execution during their execution time due to waiting for some activity to complete, e.g. an accelerator to return data. However, this situation can cause substantial performance/schedulability degradation.

A. Examples of the self-suspending task system

- Hardware acceleration by using co-processors and computation offloading.
  In so many embedded systems, the selected portions of programs are executed on dedicated hardware co-processors in order to satisfy performance requirements.
  There are two types of ways to utilize hardware co-processors. One way is busy-waiting, the software task does not give up its privileges on the processor and has to wait by spinning on the processor until the co-processor finishes the requested work(figure 08).

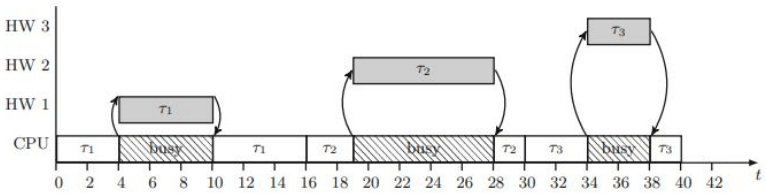

Fig. 8. Busy Waiting [1]

The next strategy is to suspend the software task. This allows freeing the processor so the processor can be used by the other ready tasks. Therefore, even single-CPU systems can execute tasks simultaneously. This arrangement is called limited parallelism, which improves the performance of the CPU by effectively utilizing the processor and the co-processors [2].

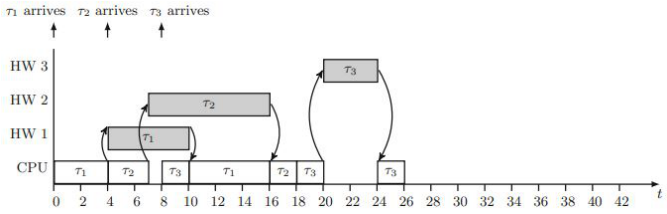

Fig. 9. self-suspending task [1]

So, when a self-suspending task is assigned, the CBS algorithm allocates a budget to the task and the task can use the budget to execute. The task can return the budget to the CBS algorithm if it wants to suspend itself.
A use of CBS in self-suspending tasks is the Hard Constant bandwidth sever(H-CBS)algorithm, a hybrid of the CBS algorithm and the Earliest Deadline First (EDF). The following sections describe the H-CBS in detail using its budget-based formulation and fundamental properties. The rules of an H-CBS server with period `P` and maximum budget `Q` (bandwidth α = `Q/P`) are summarized below.
At any time t, the server is characterized by an absolute deadline *d(t)* and a remaining budget *q(t)*. When a task is executed *q(t)* is decreased accordingly. The H-CBS has three states: Idle, Ready and Suspended. [3]

- H-CBS algorithm definition

Rule 1: Each server is Idle with $q = 0$ and $d = 0$.

Rule 2: When the server is Idle or a job receives at time t, a replenishment time is computed as

t_r = d(t) – q(t)/α
if t is smaller than t_r, the server becomes Suspended and it remains suspended until time t_r, At time t_r, the sever becomes Ready, the budget is replenished to Q and d $\leftarrow$ t_r + P.
otherwise, if the server becomes Ready, the budget is immediately replenished to Q and d $\leftarrow$ t + P.

Rule 3: When q = 0, the server becomes suspended and suspended until time d. At time d, the server becomes Ready, the budget becomes Q and the deadline is postponed to d $\leftarrow$ d + P.

Rule 4: : When the server does not have any pending work it turns to the Idle state, holding the current values of both budget q and deadline d.

Theorem 01:
$\sum_{i=1}^{n} \frac{Q_i}{P_i} \leq 1$

The H-CBS server can be used to achieve temporal isolation among a set of real-time tasks. [3]

Theorem 02:
Given sets of n non-self-suspending tasks having an implicit deadline, associates each task to an H-CBS server Si. For each H-CBS server Si, define Q_i = C_i and P_i = T_i = D_i . This set of tasks is executed each upon a dedicated H-CBS and is schedulable with EDF if and only if the test of the previous Equation holds. [3]

The advantage of associating each task to an H-CBS server is enabling the protection from over-run tasks of the system. In this way, when each task is executed upon a reservation server, all the overruns are protected by the budget exhaustion mechanism that stops the execution of the task. This way, only the task that is experiencing an overrun is affected in terms of stimulability, guaranteeing the deadlines of other tasks. [3]

VII. Comparison of CBS with other service mechanisms

A. Variable bandwidth Server (VBS)

VBS is an extension of a constant-bandwidth server where the throughput and latency of process execution cannot be controlled to remain constant across different workloads but also vary in time as long as the resulting bandwidth stays below the given bandwidth cap. [4]

1) Period Adjustment:
VBS provides dynamic period adjustment while the CBS provides constant bandwidth for each process. In comparison, VBS provides better resource utilization and adaptation to dynamic workload demands.
2) Responsiveness:
VBS provides responsiveness to the changing resource demands and allocates bandwidth dynamically. This process allows to obtain more resources when needed and release resources when they are not required. While CBS is using a constant bandwidth allocation and may not be responsive to sudden changes in process requirements or workloads.
3) Flexibility:
VBS provides great flexibility when it comes to resource allocation by allowing dynamic bandwidth allocation in changing workload requirements. While CBS provides deterministic and predictable resource allocation.

B. Best-Effort Bandwidth Server (BEBS)

The BEBS server is an aperiodic server that can be integrated into the real-time system. The algorithm adjusts its period dynamically based on the runtime of tasks and it recognizes each best-effort task is not needed equal responsiveness and adapts its scheduling accordingly. [5]

1) Period Adjustment:
BEBS dynamically adjusts its period based on the behaviour of the assigned task while CBS allocates a fixed bandwidth to each best-effort task and maintains it over time.
2) Responsiveness:
BEBS aims to provide better responsiveness for best-effort tasks by combining a timeshare strategy with an aperiodic server. While CBS treats all best-effort tasks quality and does not consider the levels of responsiveness.
3) Flexibility:
BEBS algorithm is designed to be integrated into a real-time scheduler allowing the simultaneous operation of soft real-time, hard real-time and best-effort tasks. While the CBS is more focused on providing CPU bandwidth reservations continues media applications and is less flexible when it comes to mixed time-constrained workloads.

C. Total Bandwidth server (TBS)

1) Period Adjustment:
TBS allows for dynamic period adjustments based on workload demands and available resources.
2) Responsiveness:
TBS dynamically allocates bandwidth based on the workload and requirements and releases unused resources improving the system's responsiveness and workload variations.
3) Flexibility:
TBS allows dynamic bandwidth allocation based on the workload demands improving the flexibility of the system.

VIII. Advantages and Disadvantages of Constant Bandwidth Server

1) Advantages:

1) CBS allows a guaranteed bandwidth regardless of the system load and other tasks.
2) CBS allocates a fixed priority to each task from higher priority to lower priority ensuring that each critical task's deadline is met.
3) CBS provides precise predictions. The timing and execution are predicted more accurately allowing strict timing requirements.
4) CBS allows temporal isolation for tasks. This behaviour prevents tasks from exceeding allocated bandwidth. [3] [4]

2) Disadvantages:

1) Limited flexibility, CBS is not designed to dynamically changing environments. It is designed for real-time systems where the timing is critical.
2) CBS is not originally designed to cope with overruns. This can affect system performance.
3) CBS lacks adaptability, which means once the bandwidth allocation is set any actual requirements spike will not be considered.

IX. Future Directions

The future of CBS depends on its applications and use in real-time systems. CBS is great for guaranteeing temporal isolation and strict deadlines. Also, there are some areas that the CBS can improve in the future.

X. Conclusion

This paper provides a proper introduction to the CBS with the implementation of the algorithm itself. Then the real-world uses are discussed along with advantages and disadvantages. Then the future perspective of the CBS is discussed along with the possible improvements.

As a conclusion, CBS is a Scheduling algorithm used in real-time systems all over the world. CBS ensures temporal isolation and maintains system integrity while ensuring that each task is executed within its allocated time frame. This means CBS ensures reliable operations of critical tasks guaranteeing the quality of service. CBS is an essential part of real-time systems across multiple industries.

References

[1] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications - Second Edition, Springer 2005*, 2011, using this resource, got an overall idea of real-time systems and especially about constant bandwidth servers in its subsections. This resource was really helpful when getting to know about real-time systems and their operations. Furthermore, this resource was used as the main resource for this paper. The figures in this book are self-explainable Therefore all the external figures are taken from this book.

[2] C. J.-J. N. G. H. W.-H. C. K. e. a., "A review of self-suspending tasks in realtime systems," 2018, this resource helped with understanding self-suspending tasks in real-time systems and their uses and how a self-suspending task is handled in a real-time system.

[3] A. B. M. M. A. Biondi, "Resource reservation for real-time self-suspending tasks: Theory and practice," 2015, this resource also provided information regarding self-suspending tasks and how the resource reservation works on real-time systems.

[4] C. M. K. H. P. H. R. a. A. S. S. s. Craciunas, "Programmable temporal isolation," this resource helped with understanding how temporal isolation is achieved in real-time systems and how it is implemented.

[5] S. Banachowski, T. B. Brandt, and S. A. Anderson, "Integrating best-effort scheduling into a real-time system," 2006, this resource is used to understand the best-effort scheduling and how it is implemented in real-time systems. Used in the section where the comparison happened between CBS and Best-effort scheduling.