# Real-Time System Scheduling using Dynamic Priority Exchange Server

A seminar paper on Butazzo's dynamic priority exchange server working under EDF

Emirkan Sali
*Electronic Engineering Student*
*University of Applied Sciences Hamm-Lippstadt*
Lippstadt, Germany
emirkan.sali@stud.hshl.de

***Abstract*—In the world of real-time systems, there are many scheduling options to ensure that the system meets the appropriate timing constraints and works exactly as intended and predicted. The dynamic priority exchange server is an extension of the priority exchange server based on the Earliest Deadline First algorithm, in which the main function is that the priorities of assigned tasks are interchangeable in their runtime.**

***Index Terms*—Real-time, Scheduling, Dynamic, Server, aperiodic, periodic**

## I. Introduction

Real-time systems have been a crucial part of society since processor-based technology was introduced to the world. Many examples of applications include industrial, automotive, medical, and many more [1]. Every computer-controlled system needs to perform according to its given function, but in many cases, the functions of the systems include time constraints for certain tasks. For these types of applications, real-time systems are required to ensure the execution of given tasks according to the time constraints of the system. If the system reacts too late to a given task, it could have mild or catastrophic consequences depending on the type of system and its use case [1]. Foreseeing and preventing such cases from happening in the first place makes predictability the most crucial attribute for real-time systems.

For any real-time system, a given task can be seen as a process that needs to be executed by the CPU of the system. To ensure predictability and properly handle all tasks within the given time constraints, an operating system with the proper task scheduling algorithm is needed. When a CPU receives a set of tasks from the system, it has to be assigned to process each task in a way that can be determined and controlled [1]. To achieve the proper way of assigning the CPU to the right task at the right time to make sure the system does not fail its time constraints, there are many scheduling algorithms that can be used. In this paper, the main topic will be the Dynamic Priority Exchange server (DPE), proposed by Spuri and Buttazzo, which operates in a system scheduled by the Earliest Deadline First algorithm [1].

In an scheduling algorithm, to describe the behaviour of the algorithm and the given taskset, the following notations are introduced [1]:

- $\Gamma$ is a set of periodic tasks
- $\tau_i$ is one task of a periodic taskset
- $\tau_{i,j}$ is the $j$th instance of task $\tau_i$
- $C_i$ is used to represent the computation time of task $\tau_i$
- $T_i$ is used to represent the period of task $\tau_i$
- $\phi_i$ represents the current phase of task $\tau_i$
- $D_i$ denotes the relative deadline of a task $\tau_i$
- $d_{i,j}$ is the absolute deadline of task $\tau_i$ and its instance $j$

## II. Basics for dynamic priority exchange server

To understand how the dynamic priority exchange server operates, one has to explore the functionality of its basics first. Decomposing DPE, we see:

- DPE is based on the earliest deadline first (EDF) algorithm. [1]
- DPE is a Server in the system, which works in the same way as a periodic task. The polling server serves as its basis. [1]
- DPE can be seen as an extension of the priority exchange server, in which case it is important to understand its functionality first. [1]

### A. Earliest Deadline First algorithm

The Earliest Deadline First algorithm is a dynamic algorithm in which every task's priority is directly based on their current deadline [1]. To be more precise, tasks that have earlier deadlines than others will receive a higher priority and will be executed as such. In this case the deadline of a periodic task can be denoted in the same way as in [1] as

$$d_{i,j} = \phi_i + (j-1)T_i + D_i \tag{1}$$

A Schedulability analysis can be performed using the formula

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{2}$$

The bound 1 in this case denotes the CPU being utilized at 100%. Any value below 1 or equal 1 therefore means that the CPU is able to schedule every task sucessfully under EDF [1]. It is notable that, although EDF is based on deadlines, the schedulability of the algorithm does not depend on its task deadlines. An example by Butazzo [1] of EDF can be seen in figure 1

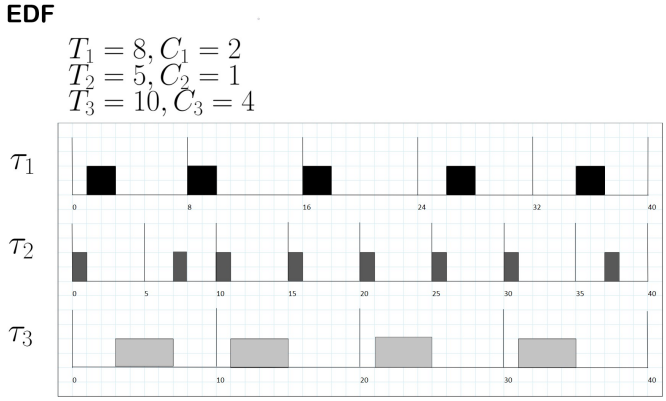


Fig. 1. Example of an EDF algorithm that has scheduled 3 tasks with different periods and computation times

In the given example the periods of the task are the same as the deadlines. Three tasks $\tau_1$, $\tau_2$ and $\tau_3$ are given with periods $T_1$=8, $T_2$=5, $T_3$=10 and worst case computation times $C_1$=2, $C_2$=1, $C_3$=4 respectively. At $t$=0 we can see that the task with the shortest period gets executed first, because its deadline is way earlier than the other 2 tasks. After the execution of $\tau_1$, every other task gets executed in a decreasing order of the deadline. The dynamic mode of operation is visible at $t$=35. Although $\tau_2$ has a smaller period than $\tau_1$, it gets executed after $\tau_1$. At this point in time, all three tasks' deadlines are the same, after changing during the time period of the example, so the task which is in the front row of the queue gets executed first, i.e in this case $\tau_1$.

A Schedulability analysis for the example of figure 1 is given by inserting the values into the formula of equation 2, which yields:

$$\frac{2}{8} + \frac{1}{5} + \frac{4}{10} \leq 1 \tag{3}$$

Summing up the values on the left side, we have:

$$0.85 \leq 1 \tag{4}$$

Which means, EDF can schedule all tasks in the system without any problems.
All in all, EDF is an optimal algorithm for usage in single processor systems with no energy limitations and only independent hard deadline jobs, because it will successfully schedule any set of periodic tasks when it is possible at all to do so [2].

### *B. Polling Server*

In many scheduling algorithms the response time for aperiodic requests is not the same as for periodic request, since these requests can arrive at an unpredictable time. A lot of aperiodic requests do not have a tight deadline unlike periodic tasks and are therefore sometimes handled with low priority, [1]. In background Scheduling for example, aperiodic tasks that arrive, are always handled by the First Come First Serve (FCFS) principle, but they are only handled at times, where the CPU has no periodic tasks ready to execute [1].

To improve the response time for aperiodic requests of the system, a server can be introduced. This server runs in the same way as an periodic task and can therefore be scheduled using an algorithm that is optimized in handling periodic tasks [1]. A server also has a computation time $C_s$, called server capacity and a period $T_s$ like any other periodic task [1]. The polling server is one algorithm that uses a server approach to schedule a set of tasks [1]. In this approach, the server is a periodic task with capacity $C_s$ and period $T_s$ handling aperiodic requests in an system that schedules using Rate Monotic Scheduling (RM) i.e. an algorithm that gives higher priorities to tasks with shorter periods and schedules them based on that [1]. In an example shown by Butazzo [1] one can gain an understanding of how the polling server handles its requests in figure 2

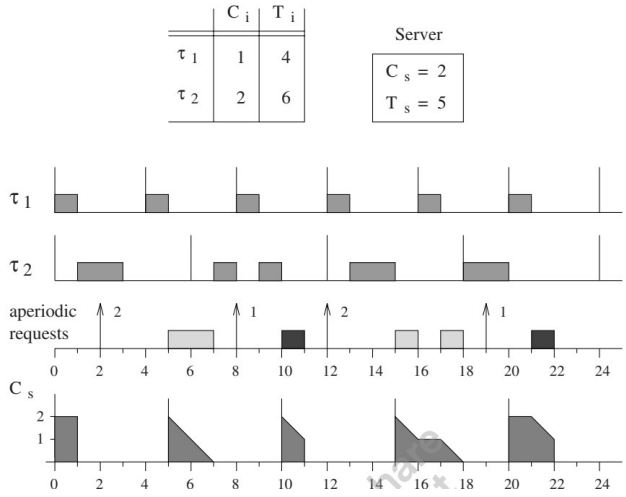


Fig. 2. Example of polling Server approach from [1] for a set of two periodic tasks and aperiodic requests

The polling server is a low priority server and will suspend its capacity if there are no aperiodic requests in the time it is active [3] [1]. any aperiodic request will have to wait until the next server period when its capacity is replenished again [1]. In figure 2 one can see how the server capacity is consumed whenever an aperiodic task is ready to execute, denoted by the arrows and their computation time next to them. If the server has no aperiodic tasks to handle, periodic requests will be handled instead and the server suspends its capacity until the next period [3] [1].

In the case of the polling server, it has a low priority and does not reserve any server capacity during its period, unlike

e.g. the Deferrable Server, which can operate at a higher priority than some given tasks and also preserves its capacity when there is no aperiodic request at the start of its period [1].

### C. Priority exchange Server

The most similar algorithm to DPE is the Priority Exchange (PE) Server proposed by Lehoczky, Sha, and Strosnider [1]. It works in a similar way to the normal polling server in that it is a periodic task, but it is usually a high priority with drastic differences in capacity preservation [1]. Instead of preserving its capacity for itself during the period, PE exchanges its capacity with the execution time of a lower priority task for potentially longer than just one period [1].It means that the capacity of the server basically gets stored in other periodic tasks and it can be transferred back to the server whenever an aperiodic request arrives. The only way in which the server capacity is lost is when the processor sits idle due to lack of periodic tasks [3]. Like any other server, the capacity is set to the normal value at the beginning of every period, but the exchanged capacity will not get lost when other periodic tasks are running. An example of PE by Butazzo [1] is shown in figure 3

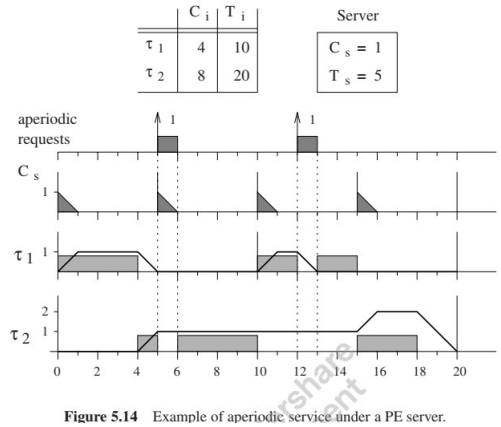


**Figure 5.14** Example of aperiodic service under a PE server.

Fig. 3. Example of Priority exchange approach for a set of two periodic tasks and aperiodic requests from [1]

at t = 0 one can see how the server capacity is consumed and transferred to $\tau_1$ by observing the behaviour of the blank line on the diagram. at t = 4, $\tau_2$ arrives and the server capacity is exchanged with the task and consumed from $\tau_1$. at t = 5 an aperiodic request with computation time = 1 arrives and gets handled by the server using its capacity, which has been replenished at the same time. Note that the capacity stored in $\tau_2$ still remains. At t = 10 the server capacity is replenished again and is exchanged with $\tau_1$, because no aperiodic requests are ready to execute at that time and $\tau_2$ is not ready to execute again. the saved capacity of the currenttly running task $\tau_1$ is consumed at t = 12, when an aperiodic task with computatio time = 1 arrives. after execution, $\tau_1$ continues executing without the exchanged capacity of the server. Lastly at t = 15 the server replenishes its capacity again and exchanges it with the current running task $\tau_2$, giving it a capacity of 2 at that point. However, this capacity is lost after execution of $\tau_2$ is finished, because no other tasks are ready to execute.

## III. Dynamic priority exchange server

DPE works as an extension of PE in a way that it uses its base functionality, but refines the priority assignment of periodic tasks by using EDF to schedule them. When no aperiodic requests arrive, the algorithm will schedule the periodic tasks in the same way as explained with the example of EDF in figure 1. That makes DPE a dynamic approach for PE, because the priorities of a taskset $\Gamma$ change dynamically during execution, depending on their approaching deadline with respect to the passed time [1].

The algorithm can be described as follows:

- Like any other server it has a period $T_s$ and a capacity $T_s$ [1].
- Each period, the servers aperiodic capacity is replenished to $C_S^d$ with $d$ being the deadline of the current server period [1].
- All periodic tasks have an aperiodic capacity $C_{S_i}^d$, initally set to 0 [1].
- all aperiodic capacities get assigned priorities depending on their current deadline [1].

an example of DPE is visualized in figure 4

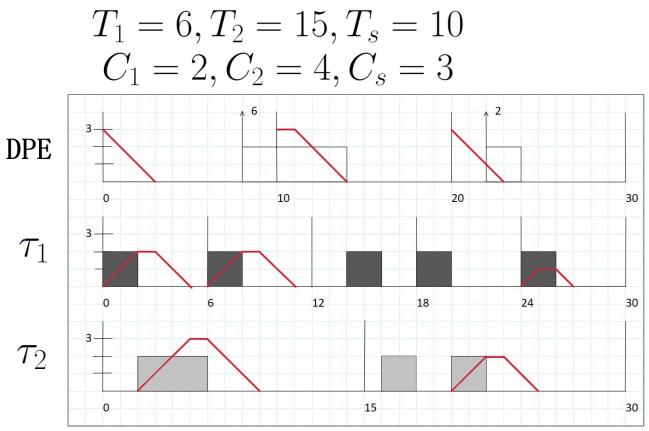


Fig. 4. Example visualization of the dynamic priority exchange server algorithm running under EDF

In the same way as in the EDF example, the deadlines in this example are the same as the periods of the tasks.
At time $t$ = 0 the aperiodic capacities $C_1^6$ and $C_2^{15}$ for the tasks $\tau_1$ and $\tau_2$ respectively are set to 0 and the Server Capacity is initially set to $C_s^{10} = 3$. No aperiodic requests are pending at that point in time, so the capacity of the server is consumed in three time units and the periodic tasks $\tau_1$ and $\tau_2$ are executed according to EDF. Synchronously to the consumption of the server capacity, the aperiodic Capacities of $\tau_1$ and $\tau_2$ increase to $C_1^6 = 2$ and $C_2^{15} = 1$. This can be seen by observing the change of the red line in figure 4.

At time $t$ = 6, $C_2^{15}$ is at 3 and is about to be consumed, but a new period for $\tau_1$ starts and since no aperiodic request is pending at that time, it gets executed. During execution 2 units of $C_2^{15}$ are consumed and accumulated in $C_1^6$. At $t$ = 8, an aperiodic request, $J_1$, of six units of time arrives in the system. Since the capacity of the DPE has been preserved, it starts executing immediately upon its arrival. Usually the 3 units of server capacity would not be sufficient runtime to execute the full aperiodic task, but at $t$ = 10, the server capacity gets replenished to 3 and is fully consumed by aperiodic request $J_1$. Any other new aperiodic request in the next 6 time units will have to wait for the next period of the DPE, because its capacity is fully consumed for its 2nd period in the example of figure 4.

Another aperiodic request, $J_2$, of 2 time units enters the system at $t$ = 22. The last remaining unit of capacity of the server is consumed to execute $J_2$ as well as one unit from $C_2^{15}$. After execution of $J_2$ is complete, the remaining unit of aperiodic capacity in $C_2^{15}$ is consumed and accumulated in $C_1^6$. At $t$ = 26, there are no aperiodic or periodic requests left to execute in all remaining periods, so the accumulated unit of aperiodic capacity in $C_1^6$ is consumed and therefore lost.

### A. Schedulability analysis of DPE

Determining the schedulability for a set of periodic tasks scheduled together with DPE is comparable to determining the schedulability of any EDF algorithm [1]. This is in consequence of the server behaving like any other periodic task, with the only difference being, that it exchanges priorities with periodic tasks in the system [1]. This difference, however, is not affecting the schedulability of the system it is in [1]. Therefore the schedulability can be shown by using the Liu and Layland condition [1]:

$$U_p + U_s \leq 1 \tag{5}$$

$U_p$ is the utilization factor of the periodic tasks and $U_s$ for the DPE Server [1].

### B. UPPAAl Model of a Dynamic Priority Exchange Server

The functionality of DPE can be vaguely visualized and simulated using UPPAAl. A UPPAAl model of a DPE example with 2 tasks $\tau_1$ and $\tau_2$ is shown in figure 5

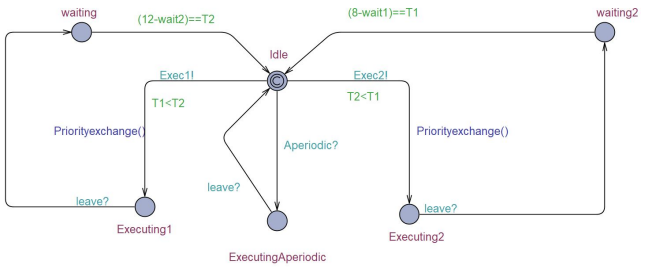


Fig. 5. Uppal model of a DPE Example with two tasks

The task entities and the behaviour of the server is saved in separate templates of the model. When no aperiodic tasks arrive, the DPE model compares the deadlines of both tasks to decide which task is allowed to execute on the CPU. When a task gets executed in the UPPAAL model, just like in DPE a priority exchange occurs with the task. The server capacity is exchanged with the task and is stored in an integer variable. If an aperiodic request arrives, this integer value is taken into consideration to determine the preserved capacity and therefore allowed runtime of the server to execute the aperiodic request.

## IV. Conclusion

In total, DPE is a very useful server to utilize the advantages and schedulability of the EDF algorithm, while adding measures to handle incoming aperiodic requests in a uniprocessor system. In a system with high occurances of aperiodic request, DPE can tend to them immediately due to its high priority, but would require more capacity in case of more aperiodic requests entering during a short number of periods. In a system with low occurances of aperiodic request, DPE preserves its capacity by exchanging its priority with periodic tasks in the system and can therefore handle aperiodic requests with higher worst case computation times, given the algorithm is schedulable.

## References

[1] G. C. Buttazzo, *Hard real-time computing systems predictable scheduling algorithms and applications*. MTM, 2013, relevancy for paper topic: Main book by Butazzo, the proposer of the DPE algorithm, therefore very important for explaining the topic.

[2] M. Chetto and R. E. Osta, "Main results on earliest deadline first scheduling for energy neutral sensors," in *2023 15th International Conference on Computer and Automation Engineering (ICCAE)*, 2023, pp. 167–171, relevancy for topic: Explanation of EDF and its importancy with an application example.

[3] I. Lee. (2010) Real-time scheduling (part 2). Acessed: 4th of june, 2023, Relevancy for paper: Slides on Real-time scheduling algorithms. [Online]. Available: https://www.seas.upenn.edu/ lee/10cis541/lecs/lec-RT-sched-part2-v2-1x2.pdf

### Statement of authorship

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such.

Lippstadt, June 5, 2023 .....