

Priority Driven Algorithms for Safety Critical Systems

S. Triebert

Delft University of Technology,
EWI, Software Technology

Abstract. Safety critical systems, better known as hard real-time systems, rely processes to meet their requirements. More specifically, they need to meet their deadlines. Because these systems operate in environments in which the fault-tolerance is very close to zero, a scheduling algorithm has to be chosen to schedule these very important processes in a way, that they meet their deadlines and it's predictable whether it's possible to meet their deadlines. The basis is an online hard real-time system. After an initial introduction, two types of scheduling algorithms are discussed, fixed-priority and dynamic-priority driven. Both will be put to the test and will be compared to the requirements of an online safety critical system, which will lead to a final conclusion determining the best suitable priority driven algorithm to be used in an online safety critical system.

1. Introduction

Real-Time Systems are commonly used in safety critical environments. Safety critical environments often require processes to be dealt with within a certain time limit, this because of the fields these systems are used in. One can think of *Air Traffic Control*, *Nuclear Power Plants* or *Airbag systems in cars*. All these systems depend highly on accurate information to be delivered on time. One wouldn't want the airbag not to go off in case of a crash or have a nuclear meltdown, because information wasn't analyzed, processed and delivered on time due to systems that aren't reliable. This is where the real-time systems come in.

If we look at a definition of what a real-time system is we see often something like: "*A real-time system is a system required to complete its work and deliver its services on a timely basis*" [1]. This is exactly what we expect and demand in a safety critical environment. We not only demand that these systems work correctly and responsively, but also want to show that they indeed do work. Since if it can be proven to work in all scenarios, we can safely assume that it will work for these scenarios in real life. For the time being we assume here that all the safety critical scenarios have been tested and verified. These testing scenarios have been made with certain knowledge of the system and *jobs* performed on the system [5][6][7]. Here a *job* is a unit of work that is scheduled and executed by the system. These jobs have certain constraints and properties, also known as the *task set*. *Response times*, *deadlines*, *release times*, *worst case execution times* are critical properties of jobs when you want to predict and execute them because you have to fit them into an

execution schedule. Also the *minimum interrelease time of jobs in a task*, better known as period, plays an important role. The mentioned critical properties determine if a certain *task*, which exists out of a set of related jobs, is feasible. Feasibility of tasks is the most important issue when dealing with safety critical systems, because these systems rely on the correct and responsive execution of safety critical tasks.

To know if tasks, and consequently the jobs within the task, are feasible, one has to know if it can be executed fully and stay within its deadline. If a deadline is met, depends on the *response time*, this measured time between the release time and actual completion of the job. The maximum allowable response time of a job is its *relative deadline*. Consequently the *absolute* deadline of a job is its release time plus its relative deadline. So for a job to meet its timing constraint it has to finish within its pre-determined deadline. In respect with safety critical system, all of the safety critical tasks and job have to complete within their deadlines. Unfortunately none of the systems today have just one task to execute, all systems have multiple tasks. This means that multiple jobs have to be scheduled on one processor. The downside is that safety critical systems depend on information before the deadline of a task is met, but although the release times can be shifted around in a schedule, the deadlines stay in the same place. The information has to be there when a deadline is met, if not, the system will be compromised if a safety critical task hasn't finished.

In respect to these safety critical systems we'll determine whether scheduling done with static or dynamic priority scheduling algorithms will deliver the most reliable and most feasible results.

2. Scheduling algorithms

When we are talking about scheduling algorithms in this paper, we're talking about algorithms on a priority-driven scheduler, also known as an online scheduler. A priority-driven scheduler does not pre-compute a schedule of tasks, meaning that the job which are to be executed, are simply put in a queue and await their turn to be executed based on a certain scheduling algorithm.

There are two kinds of algorithms for priority-driven schedulers, fixed-priority and dynamic-priority based algorithms. In the next two sections we're going to discuss these two algorithms and sketch an idea of how these algorithms generally work.

2.1 Fixed scheduling algorithms

A fixed-priority algorithm handles all the jobs in a task with the same priority. The scheduling algorithm has complete knowledge of the task set and its constraints. These include deadlines, computation times and (future) release times. The priority of periodic tasks is fixed – or static – relative to other tasks. The most commonly known static algorithms are the *Rate-Monotonic* [3] and *Deadline-Monotonic* [2] algorithms. For further reference we will refer to them as RM and respectively DM. The idea behind the two is almost the same. Both assign a priority to a task based on information from the task set.

The RM algorithm uses the periods of tasks to assign the priority: *the shorter the period, the higher the priority*. Since the rate – of job releases – of a task is the inverse of its period, this implies: *the higher its rate, the higher its priority*. This is also where it derives its name from, Rate-Monotonic. We will give a quick demonstration with the help of Figure 1. Task 1 has an execution time of 1, and a period of 4, so $T_1(4,1)$. Task 2 will be equal to $T_2(5,2)$ and Task 3 to $T_3(20,5)$. When we go thru the algorithm we get:

- T1 has the highest – rate – priority, so is placed at the head of the queue.
- T2 has the second highest rate and will be executed at time 2 after T1 has finished.
- T3 will execute after T1 and T2 have finished. T1 has met its requirements at time 4.
- T3 will be preempted at time 4, because T1 has the highest priority again.
- At time 5, T2 will start – its new period – at time 5, having higher priority than T3.
- At time 7, T3 continues after T1 and T2 finished within their periods.
- At time 8, T1 starts again with highest priority.
- At time 9, T3 starts, because T2 has already finished and so has T1
- At time 10, T2 has higher priority because its period starts again.
- At time 12, T1 starts and finishes within its period
- At time 13, T3 starts and finishes within its period of 20 and isn't preempted by T2 because T2 already finished within its period and hasn't started with its next period, which will start at time 15.
- At time 16, T2 is preempted because T1 has a higher priority.
- And so on.

There have been written several papers [3][8] that go deeper into the exact workings of the RM algorithm.

Another algorithm closely related to the RM algorithm is the DM algorithm. The DM algorithm uses the relative deadline of tasks to assign the priority: *the shorter the relative deadline, the higher the priority*. The DM algorithm equals the RM when the relative deadline of every task is proportional to its period. In other cases it performs better since it can produce a feasible schedule where the RM could fail. Since going a deep analysis of all fixed algorithms goes beyond the scope of this paper, a more in depth view of the DM algorithm can be found in [4].

The strongest point of fixed-priority algorithms is the predictability. One is able to predict the whole process that will be executed. But predictability has a cost, the cost of performance.

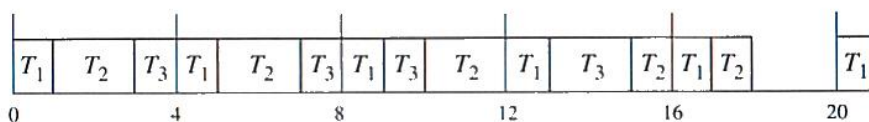


Figure 1. RM Schedule [1]

2.2 Dynamic scheduling algorithms

Dynamic scheduling in contrast to static scheduling does not have complete knowledge of the task set or the timing constraints. It assigns different priorities to individual jobs in each task. The priority changes as jobs are released and completed, so the task's priority changes in respect to other tasks. This is also the dynamic aspect, the shifting priorities of tasks. There are many dynamic algorithms that have proven themselves to work, but since this goes beyond our scope, we'll give insight into how dynamic algorithms generally work with the use of an example algorithm. In this case we will discuss the – well-known – *Earliest-Deadline-First* algorithm into a little bit more detail. For future reference we will refer to the *Earliest-Deadline-First* algorithm as EDF.

As the name already inclines, this method assigns priority to tasks according to their deadlines, more precisely, *it assigning priorities to individual jobs in the tasks according to their absolute deadlines* [1]. We will explain this algorithm with the help of Figure 2. There are two tasks, Task 1 with a deadline of 2 and an execution time of 0.9 – T1(2, 0.9) – and Task 2 with T2 (5, 2.3).

- At time 0, both jobs of T1 and T2 are ready. T1 however has an earlier absolute deadline of 2, so the first job of T1 is executed first, after which T2 starts.
- At time 2, the next job of T1 is released, and its deadline is at time 4, which is earlier than that of T2. The currently running job of T2 is preempted and the job of T1 is executed.
- At time 2.9, the job of T1 finishes, the previous preempted job of T2 is continued.
- At time 4, the next job of T1 is released. Its deadline is 6, the deadline of the current job of T2 is however 5. The job of T2 may continue executing.
- At time 4.1, the job of T2 finishes, the next job of T1 starts to execute.
- And so on.

This algorithm already shows how the scheduling is done without knowing about when tasks – with its jobs – are to be released in the future. The parameters of each job become known after the job has been released.

A drawback of these dynamic methods is that there is no easy way to determine which tasks miss their deadlines. Also dynamic resource scheduling will result in problems, meaning not many systems offer a *resource sufficient* – where recourses can be guaranteed – environment. Dynamic algorithms that can handle these available resource problems, like the *Spring* algorithm [9][10], can result in highly underutilized systems, based on a pessimistic estimation of workload. This is also where the unpredictability comes in with all dynamic methods. There is no efficient way to determine in dynamic systems whether all the deadlines can be met and what the kind of overload is expected, so whether there is a way to get the tasks feasibly scheduled.

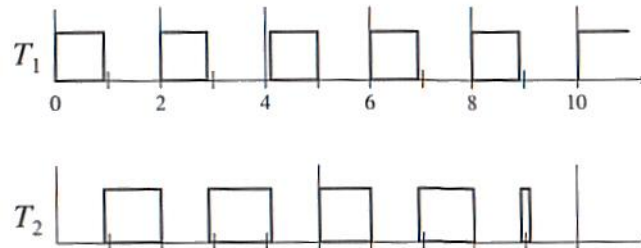


Figure 2. EDF Schedule [1]

3. Safety Critical Systems

Safety critical systems, better known as *hard real-time systems*, require for all tasks and processes to meet their requirements. In contrast to soft real-time systems, none of the deadline can't be met, meaning all of the deadlines have to be met. Since modern day safety critical systems are consisting of many components and numerous input variables, there are many different requirements for tasks and jobs within those tasks. Creating a system that supports *offline* – predefined schedules – scheduling is a difficult problem. To create such a – offline – system, it has to be deterministic.

Looking at the current state of many hard real-time systems, which have a growing number of resources, and already consist of many components and resources, determining if such a system is deterministic is quite hard. Many component and resources haven't got a fixed set of functions, which can be used and are required to schedule offline. Subsequently, although such systems have the advantage of knowing in advance if it's feasible and how to get it feasible, it isn't flexible, it creates a static schedule. This doesn't allow for processes that react on certain outcomes of tasks. There can't be tasks released without knowledge of future release times in offline systems.

For the reasons above an *online* – priority driven – hard real-time system seems a good option, although one has to take in account that whenever jobs are nonpreemptable, *no optimal online scheduling algorithm exists* [1]. It has however the ability to make scheduling decisions without the knowledge about tasks released in the future. This would make scheduling flexible and adaptable, which is ideal for future unpredictable workloads on systems. But the question would be how such an online system, which doesn't have a static – known feasible – schedule, could be implemented best. As described earlier, there are fixed and dynamic-priority driven algorithms, available for online systems. Trying to determine the best scheduling type for *online hard real-time systems*, fixed or dynamic, remains to be done.

4. Fixed and dynamic algorithms put into practice.

To put the two scheduling methods to the test, we assign a priority to a task in a RTOS – Real-Time Operating System –, in this case a real-time version of Linux,

RTAI. Interrupts don't pose a big problem on this platform since the RTAI system has a patented solution [12] to overcome these issues. Assigning a priority to a task, we can directly see the benefits of a fixed-priority algorithm. The task with the highest priority will be executed, unless there is a task with a higher priority, which then gets its share of the processing unit. Also the task with the highest priority will have the least deviation in executing time. Only pure hardware related issues as pipelining and internal caching of instructions within the processing unit can make this executing time deviate. Here also the prediction comes into play. The order in which the tasks are executed is always in the order of their priority or as earlier mentioned, *the priority of tasks is fixed relative to other tasks*. The main drawback would be that the dynamic algorithms outperform these fixed ones [1]. Also utilization of the system resources wouldn't be used to the maximum.

Testing this with a dynamic-priority driven algorithm is more difficult. Not many – real-time – platforms have this dynamic-priority algorithm in place by default. There have been projects, which have added and created dynamic-priority driven algorithms, have done a survey [11] about it or have extended it [13][14]. Most of these projects added extra awareness of certain variables, which the system uses. They for example add extra awareness using a control-loop, or extending certain dynamic algorithms trying to overcome existing problems like predictability of online feasibility. But with the currently available dynamic algorithms, the main drawback is still the unpredictability and the often-serious problems that occur when a deadline is missed, this especially is a problem of the EDF dynamic algorithm. In return, the algorithms are outperforming the fixed algorithms and can thus perform and schedule quite effectively, in the best-case scenario [1].

5. Summary and Conclusions

After having chosen for an *online* – hard real-time system – safety critical system to be the basis, the choice that remains is, what kind of algorithm should be used, fixed or dynamic-priority driven based. Both algorithms have their pros and cons. However when taking into account what a safety critical system stands for, the pros and cons of both algorithms can easily be translated into requirements of safety critical systems.

Although the dynamic algorithms can perform better and have a better utilization of resources available, fixed algorithms, provide in any case better predictability. Looking at where safety critical systems come from, predictability is something which a running online hard real-time systems needs. Since it needs to make those deadlines, needs to know and monitor whether certain deadlines can be made, fixed-priority algorithms are better to use. Also high priority tasks run the risk in dynamic-priority algorithms that they will be missed, but not dropped fast enough, which could lead to cascading effects of many tasks not meeting their deadlines.

To conclude, fixed-priority algorithms for now are the best available options when choosing for an online safety critical system. Although research has shown that also dynamic algorithms are getting more predictable [13], fixed-priority algorithms have the preference. Safety critical systems determine on deadlines to be met and fixed-

priority algorithms are more predictable over dynamic-priority algorithms concerning meeting deadlines.

Acknowledgements

This paper has been written as a final assessment for the course *IN4024 – Introduction To Real-Time Systems*, taught by Dr. H.G. Gross associated with the Delft University of Technology. This course brought the world of real-time systems to my attention and deepened my interest in it. Experiments conducted for the course and paper, were conducted at the HLO facility on the Delft University complex.

References

- [1] Liu, J.W.S., *Real-Time Systems*.
Prentice-Hall, Inc.
2000
- [2] Leung, J.Y.T., and Whitehead, J., *On the complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks*.
Perf. Eval. (Netherlands), 2, pp 237-250
1982
- [3] Liu, C.L. and Layland, J.W., *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*.
JACM, Vol. 20, No. 1, pp. 46-61
1973
- [4] Audsley, N.C., Burns, A., Richardson, M.F. and Wellings, A.J., *Hard Real-Time Scheduling: The deadline-monotonic approach*.
Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems, pp. 133-137
1991
- [5] Sandell, D., Ermedahl, A., Gustafsson, J. and Lisper, B., *Static Timing Analysis of Real-Time Operating System Code*.
Dept. of Computer Science and Engineering.
- [6] Dianati, M., Song, I. And Treiber, M., *Introduction to Genetic Algorithms and Evolution Strategies*.
University of Waterloo, Ontario, Canada.
- [7] Whitley, D., *Overview of Evolutionary Algorithms*.
Computer Science Department, Colorado State University.
- [8] Lehoczky, J.P., Sha, L. and Ding, Y., *The Rate-Monotonic Scheduling Algorithm – Exact Characterization and Average Case Behavior*.
IEEE Real-Time Systems Symposium.
1989
- [9] Ramamritham, K. and Standkovic, J.A., *Dynamic task scheduling in distributed hard real-time systems*.
IEEE Software, Vol. 1, No.3.
July 1984

- [10] Zhao, W., Ramamritham, K. and Standkovic, J.A., *Preemptive Scheduling Under Time and Resource Constraints*.
IEEE Transactions on Computers 36(8).
1987
- [11] Deng, Z., Lui, J.W.S. and Sun, J., *Dynamic Scheduling of Hard Real-Time Applications in Open System Environment*.
Department of Computer Science, University of Illinois at Urbana-Champaign.
- [12] Yodaiken, V., *Adding Real-Time Support To General Purpose Operating Systems*.
U.S. Patent No. 5,995,745
November 30, 1999
- [13] Jansen, P.G., Mullender, S.J., Havinga, P.J.M. and Scholten, H., *Lightweight EDF Scheduling with Deadline Inheritance*.
University of Twente.
May 9, 2003
- [14] Stankovic, J.A., Lu, C., Son, S.H. and Tao, G., *The Case for Feedback Control Real-Time Scheduling*.
Department of Computer Science, University of Virginia