Earliest Deadline First Scheduling Algorithm

Yigitcan Aydin
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
Email: yigitcan.aydin@stud.hshl.de

***Abstract*—Real-time embedded systems has been increasing its usage in the 21st century and it is not showing any signs that development of real-time systems will slow down. Since embedded devices can be found in almost every field in the modern life of the people, e.g. from medical industry to gaming, from aviation to logistics, etc., as a natural result of this, they are getting more complicated and they have to serve in wide range of tasks. Therefore, a careful and detailed research have to be done on how these overwhelming and outnumbered tasks should be handled. Hence, in this paper one of the well know scheduling algorithms, which is earliest deadline first scheduling algorithm, is being discussed as well as some other must-know subjects such as the general definition of the real-time systems, an overview to aperiodic and periodic scheduling algorithms.**

I. Introduction

In order to acquire better understanding of the scheduling algorithms used in embedded systems, especially in real-time systems, first we have to be get familiar with the basic terms and concepts in real-time systems. As a starting point we begin with the definition of the real-time system. A real-time system is one in which the accuracy of system behavior is determined by not only the logical outcomes of operations, but also by the physical time in which these outputs are generated. The set of outcomes in time of a system is referred to as system behavior [1]. Real-time systems are basically categorized under three main terms such as soft, firm and hard real-time systems.

A soft real-time task is one that makes progress after its deadline and still has some use for the system, despite decreasing efficiency [2]. Few examples of soft real-time systems are personal computers, video games, multimedia players, web browsers, etc. When some of the tasks fail to be scheduled, it does not affect the overall system performance and these failures are not critical that they would cause no harm to the system as well as the user.

A real-time activity is defined to be firm if delivering outcomes after the deadline is worthless to the system but not harmful [2]. For example, video conference, voice call systems for daily life, navigation systems can be given as example for firm real-time systems. If the tasks fail to meet their deadlines, it would fail the system but not in disastrous way. When the video conference system breaks down, the connection might be reestablished and continue with the meeting without harming any other hardware components and people around.

When generating outcomes after the deadlines, a real-time work is considered to be hard and might also have disastrous effects for the underlying system [2]. The traffic control systems, fire alarm and fire distinguishing systems, car break systems, etc. are the few example for the hard real-time systems. If, anyhow a simple scheduling or meeting deadline for a task in a hard real-time system occurs, this would cause severe problems such as complete failure in the system along with so many severe health problems of the people whoever is close to the place in which the hard real-time system component fails.

Apart from above explained fundamental real-time system terms, we should also discuss about the desired requirements that real-time systems should have. Tasks have to be scheduled so that they can meet the deadline constraints. Therefore, the outputs must be accurate not just in terms of value as well as in terms of timing. As a result, the system software must provide specialized features and methods for time management and task execution with precise timing limitations and varying degrees of importance [2].

Another important point for a RTS (Real-Time System) is to contain predictability in the system. In case of a task cannot be completed in time, the system must tell the user ahead of time as then alternate actions may be prepared to address the situation [2].

One of the most important features of RTS is to utilize the resources such as memory, power consumption, weight, etc., in an efficient way as much as possible. RTS have to be designed to achieve a robust system in order for the RTS not to collapse when the system is overloaded [2]. This feature leads us to fault-tolerance point where RTS must be designed in a way that it continues its process even if one of the hardware is not functioning anymore. As a last check point for a well-designed RTS is to have maintainability feature so that future system updates can be done.

For a real-time system designer to satisfy the basic requirement for a real-time embedded systems, using suitable and most efficient scheduling algorithm plays a major role. In the following sections we will discuss about general real-time

scheduling concepts such as aperiodic and periodic scheduling and later on our main concern which is the earliest deadline first algorithm, its utilization and some of its advantages and disadvantages.

II. Real-Time Scheduling

Real-time scheduling has been a big concern for the real-time embedded systems designers in order to make sure the task sets scheduled and executed before they miss their deadlines. Therefore, finding an optimal scheduling algorithm is one of the important points to realize especially the hard real-time systems.

Before getting deeper in the concept of scheduling there are fundamental term to be focused on. First of all we need to understand the difference between static and dynamic scheduling. Whenever the scheduler makes scheduling choices at compile time, this process is called static scheduling. It creates a routing and scheduling table for the run-time dispatcher [1]. For the static scheduling algorithms it is not possible to change the task priorities after the tasks are scheduled and added to the ready queue. Therefor, it requires comprehensive background information of the task-set features, such as upper limit of the execution times, priority restrictions, and deadlines, in order to do this [1].

On the contrary, dynamic scheduling is that whenever the scheduler chooses one of the latest lineup of available tasks to schedule at run-time. Dynamic scheduling algorithms are capable of modifying task situations [1]. Dynamic scheduling algorithms provide to the CPU the possibility of rearranging the priorities of the tasks so that scheduler can change the order in the ready queue at any time in the run-time of the process.

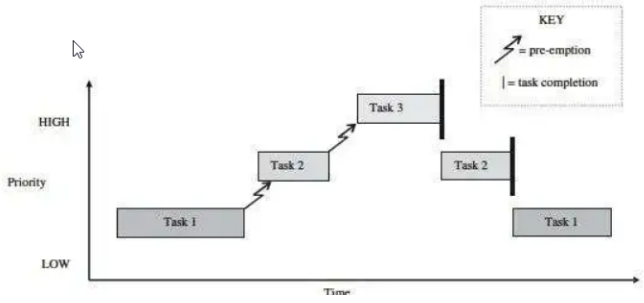

Fig. 1. Preemptive priority scheduling [3]

Another crucial concept to understand is to know the difference between non-preemptive and preemptive scheduling. The presently running task will not be easily stopped until it decides to release the provided resources by itself under non-preemptive scheduling [1]. In this mode of the scheduling, even if a higher priority task arrives, it has to wait for its own time slot assigned by the scheduler. In a circumstance where multiple small activities need to be completed, non-preemptive scheduling makes sense.

Preemptive scheduling on the other hand, where a more critical job demands service, the presently running task could also be preempted, or halted [1]. In other words, in case of occurrence of a new task, the scheduler first suspends the task that is being executed at the arrival time of the new-coming task, and executes the new task which has higher priority regarding to the task's features (see Figure 1).

Apart from the points explained above, we should consider aperiodic and periodic task scheduling algorithms briefly in order to understand the earliest deadline first algorithm. Having a short information about these algorithms will allow us to compare different kind of algorithms to EDF (Earliest Deadline First) to figure out the advantages and disadvantages of EDF more easily.

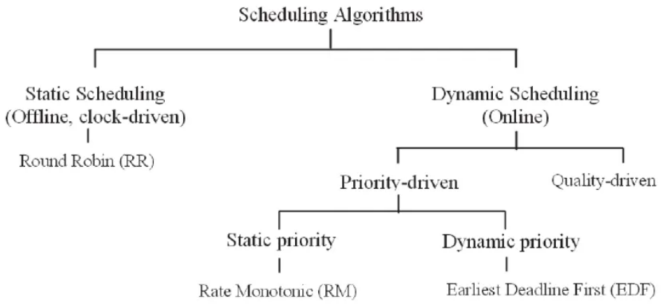

Fig. 2. Algorithms used in RTOS [4]

In the following subsections, we will be discussing about some of the mostly used algorithms in the Real-Time Operating Systems (RTOS) such as Round Robin (RR) Algorithm and Rate Monotonic Algorithm (RMA) along with the Earliest Deadline First (EDF) algorithm (see Figure 2).

A. FIFO and RR Algorithms

FIFO (First in First out) algorithm is one of the most straightforward algorithms in terms of scheduling concerns. As its name suggests that it places the very first task which arrived before other ones into the head of the task queue (see Figure 3). Tasks cannot be preempted during the run-time. Therefore, it may cause convoy problem in the scheduling process if the arriving task influx is too much. This algorithm might be useful and advantageous in some of the cases but for the hard real-time systems it might be a problems. FIFO can be used to implement the queue data structures. LIFO (Last in First out) is the vice versa of FIFO algorithm. As the common usage of LIFO, stack data structures can be given as an example.

When we look at the RR (Round Robin) algorithm, it is the version of FIFO that we have the possibility of using

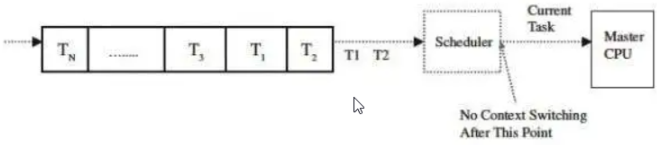

Fig. 3. FIFO algorithm working principle [3]

preemptive scheduling mode. RR algorithm is a cyclic queue algorithm so that the tasks are being executed in a circular manner.

B. Rate Monotonic Algorithm

RMA (Rate Monotonic Algorithm) is an algorithm that uses the static scheduling with the preemption mode. As we discussed before it collects all the required data so that it can schedule the tasks during the compile time. The highest priority is assigned to the task that has the shortest period. Once the highest priority task's period comes, then this task is executed by preempting the already being executed task [5]. The main purpose of this algorithm is to give more CPU usage to the more frequently used tasks.

III. Earliest Deadline First Scheduling

Earliest deadline first (EDF) algorithm is one of the dynamic scheduling algorithms that begins with the scheduling by choosing the task which has the closest time interval until its deadline. Since EDF is a preemptive algorithm, whenever new tasks arrive to the queue or a task which is in execution finishes its job, the priorities of the complete queue are reassigned. Therefore, it is considered as a part of the dynamic scheduling algorithms.

EDF is particularly used in periodic task scheduling in real-time systems. However, it can be also used for aperiodic tasks as well.

Since it is an optimal scheduling algorithm, any other algorithms can schedule a given feasible task set if EDF is able to schedule it. Therefore, in order to determine if a given task set is schedulable or not, we must first calculate the utilization factor of the task set. Utilization factor U is the some of the fractions given by $\frac{C_i}{T_i}$, where C is the completion time and T is the period (in our case it is equal to the deadline) of the individual tasks (see Equation 1).

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

[6]

Utilization factor emphasizes the CPU usage. In our equation, less than or equal to 1 means that we are able use max 100% of the CPU power. This ratio in RMA is approximately around 70% which is less than or equal to 0.7.

As long as we have a feasible task set with $U \leq 1$ we are able to schedule it with EDF. What if we had $U > 1$? In this case, EDF algorithm cannot promise us anymore that all the tasks in the task set will meet their deadlines. If such situation occurs, especially in a real-time system, that might cause hazardous problems in the system.

EDF algorithm sometimes may cause domino effect if one of tasks misses its deadline, it may severely affect the other tasks' schedulability. If we have the such case in our system, other incoming tasks that will be executed start missing their deadlines and after some point none of them are able to meet their deadlines. This possible behavior of the EDF makes it difficult to predict the execution times and the deadlines of the tasks.

Because of the difficulties in predicting the scheduling behavior in EDF, other scheduling algorithms, such as rate monotonic and fixed priority algorithms [micro], more frequently take place in task scheduling in real-time systems.

In order to understand the earliest deadline first algorithm, we can take a look at the following task set and try to analyze this set whether it is schedulable and if it is, how can we schedule it properly.

First of all, let us consider different tasks with the arrival times $A_i = \{0, 2, 4\}$, the completion times $C_i = \{1, 3, 2\}$ and the periods (in our case the periods are equal to the deadlines of the tasks) $T_i = \{4, 8, 6\}$.

Now we can start by calculating the utilization factor U which gives us the prior knowledge whether we can schedule the above given set of the tasks or not. By using the equation 1;

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1,$$

$$U = \sum_{i=1}^{3} \frac{C_i}{T_i} \leq 1,$$

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3},$$

$$U = \frac{1}{4} + \frac{3}{8} + \frac{2}{6},$$

$$U = \frac{23}{24} = 0,958 \leq 1.$$

Since the utilization factor is 0,958, which means that in order to schedule the given task set 95,8% CPU power will be used, less than 1 we are able to schedule this set of the tasks.

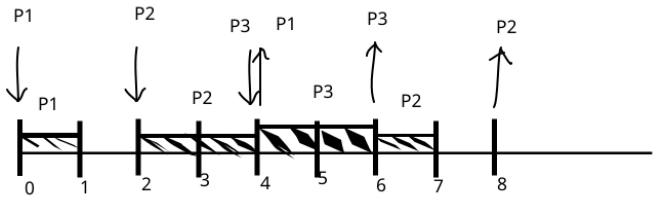

Fig. 4. Earliest deadline first scheduling example
(Downward arrow: arriving task; Upward arrow: deadline)

If the tasks had same arrival time, the priority sequence would be $P1 > P3 > P2$ according to their deadlines. In our case, the task 1 is arriving at t=0 and its completion time is 1. Therefore, it is scheduled between t=0 and t=1. Between t=1 and t=2, since there are no arriving tasks, processor is in the idle mode. At t=2 the task 2 is arriving and is scheduled until the task 3 arrives at t=4. Since the task 3 has higher priority than the task 2 according to EDF algorithm, at t=4 the task 2 execution is preempted and execution of the task 3 begins until t=6. From t=6 on there are no more higher priority tasks remain and scheduler returns executing the task 2 until t=7 (see Figure 5).

```
float cpu_util(task *t1, int n)
{
    int i = 0;
    float cu = 0;//cpu utilization factor
    while (i < n)
    {
        //U = sum(C_i / T_i)
        cu = cu + (float)t1->T[execution] / (float)t1->T[deadline];
        t1++;
        i++;
    }
    return cu;
}
```

Fig. 5. Calculation of utilization factor in C [7]

If we wanted to schedule this set of tasks with RMA, we would not be able to do this because the required utilization factor in RMA should be less than 0.69. Since U for the given set of tasks is 0.958 which is much more greater than 0.69, RMA cannot guarantee us that it can schedule our tasks.

If we assume that all the above mentioned tasks' arrival time were the same, then we could expect the following output regarding to the C++ code and its result: (see Figure 6).

IV. Conclusion

Although the EDF algorithm is one of the optimal algorithms, similarly RMA, it is not preferred to be used

```
Process 1:-
==> Execution time: 1
==> Deadline: 4

Process 2:-
==> Execution time: 3
==> Deadline: 8

Process 3:-
==> Execution time: 2
==> Deadline: 6

Scheduling:-

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
P[1]: |####|    |    |    |    |    |####|    |
P[2]: |    |    |    |####|####|####|    |    |
P[3]: |    |####|####|    |    |    |    |####|
```

Fig. 6. EDF result with the same arrival times [8]

in some of the real-time system applications. This is mainly because of the predictability problems of the earliest deadline algorithm. However, if someone wants to get a better understanding in the field of embedded systems and especially in the real-time operating systems, it is better to understand first the optimal algorithms such as earliest deadline first and rate monotonic scheduling algorithms.

References

[1] H. Kopetz, *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Springer US, 2011.

[2] G. Buttazzo, *Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications*. Springer US, 2011.

[3] EDN, "Embedded operating systems – part 2: Process scheduling," https://www.edn.com/embedded-operating-systems-part-2-process-scheduling/, April 2013, (Accessed on 06/05/2022).

[4] Microcontrollerslab, "Earliest deadline first (edf) scheduling algorithm," https://microcontrollerslab.com/earliest-deadline-first-scheduling/, (Accessed on 06/05/2022).

[5] L. Thiele, "Embedded systems, aperiodic and periodic scheduling," https://lectures.tik.ee.ethz.ch/es/slides/6$_{R}ealTimeScheduling.pdf$, $(Accessedon06/0$

[6] EmbeddedArtistry, "Earliest deadline first scheduling [edf]," https://embeddedartistry.com/fieldmanual-terms/earliest-deadline-first-scheduling/, (Accessed on 06/05/2022).

[7] jabezwinston, "Earliest$_{d}eadline_{f}irst,''$ $https$: $//github.com/jabezwinston/Earliest_{D}eadline_{F}irst$, $(Accessedon25/05/2022)$.

[8] mohammaduzair9, "Scheduling-algorithms-os/schedule.cpp," https://github.com/mohammaduzair9/Scheduling-Algorithms-OS/blob/master/schedule.cpp, (Accessed on 25/05/2022).