Improving Scheduling in Real Time Operating Systems with Earliest Deadline First

Jack Imburgia
Department of Computer Science
Syracuse University
Syracuse, NY USA
joimburg@syr.edu

***Abstract* – Scheduling methodologies in operating systems are often researched when looking for potential improvements in performance and efficiency. With the proliferation of embedded and internet of things (IOT) hardware and systems, the increased complexity of these systems, and the resulting heighted demands for resources, it is critical to maximize efficiencies in processing time, energy usage, and the capacity of throughput.**

Operating systems have traditionally relied on Fixed Priority (FP) scheduling systems. The reasons for are many including ease of implementation and predictability of output. However, a reasonable alternative to this approach is the scheduling method Earliest Deadline First (EDF). In contrast to FP, EDF offers dynamic priority scheduling where tasks are continuously reevaluated and scheduled next according to their next deadline. This can lead to greater CPU utilization which will result increased efficiency.

This paper proposes to discuss the merits of EDF as an alternative to traditional fixed priority scheduling methodologies, demonstrate how EDF works in practice, and implement a simple proof of concept EDF scheduling system.

Keywords—earliest deadline first, scheduling, queue

I. Introduction and Background

An operating system manages hardware, network activity, system I/O, persisting files to disk storage and, in the interest of this paper, scheduling processes. Real Time Operating Systems (RTOS) must behave in a way that seems immediate and responsive to end users. Operating systems used in embedded system devices such as smart phones are an example. Real time operating systems accomplish this through scheduling methodologies that allocate resources so that multiple processes can run concurrently which results in responsive experiences for users. Data is processed as it arrives and ideally there are no buffer delays.

A. RTOS Performance and Accuracy

There are two components to determine the performance of a task in a real time operating system. First, the output of task must compute the desired result correctly. Second, the task must be completed in an expected time frame.

But the accuracy of a task cannot be assessed solely on the output. There are also two additional components to determining the correctness of a real time operating system. First, it must be determined if a process was completed in a time frame that meets its requirements. Second, the difference between the expected and actual timing must be calculated to determine the variability. An important job of a real time operating system is to have a mechanism in place to deal with the worst case amounts of time each task takes to complete

B. Tasks and Scheduling.

Tasks is real time operating systems can generally be generally be put into two categories.

- There are **periodic** tasks that arrive regularly to the scheduling system and will repeat themselves in regular fixed intervals.
- There are **aperiodic** tasks that are processed by the scheduling system at irregular or unknown timeframes. The scheduling system will decide on an order if execution and start and stop tasks accordingly.

The real time operating system also decides the amount of time a process is allocated. Scheduling is typically done in a preemptive manner- the CPU is allocated to a process for a set amount of time. When that time expires its processing ceases, and the CPU is allocated to a different process through a context switch.

C. Traditonal Fixed Priority Scheduling Systems

Scheduling in real time operating systems has been traditionally accomplished through Fixed Priority (FP) systems. The importance of each task is explicitly assigned in the form of a priority. The higher priority of a task, the more important it is deemed and the lower priority the lesser important.

©2020 IEEE

When determining which processes to run at a given time, the highest priority / more important tasks will receive preference when allocating processing time on the CPU. Operations on the ready queue of processes can be done in a constant time. This can result in greater predictability of the timing of execution of processes. This demonstrates preemptive scheduling as the tasks are started and stopped due to their priority.

D. Drawbacks of Fixed Priority

While fixed priority scheduling systems are widely used in real time operating systems, they are not without drawbacks.

- Lower priority tasks will compete with higher priority tasks for CPU time and it is possible that they may not get enough time to complete (or no time at all). This is called starvation.
- If higher priority tasks continue to come into the queue, the lower priority tasks will continue to wait.

II. Analysis: Earliest Deadline First

An alternative to traditional fixed priority scheduling systems used in most real time operating systems is Earliest Deadline First (EDF). This introduces the ability of tasks to have periodic deadlines which are not seen in fixed priority.

A. Dynamic Priority Scheduling Systems

The concept of priority remains in earliest deadline first. However, is not fixed. Instead, it is dynamically determined based on the closest deadline of the tasks in the queue. The highest priority job is computed as being the one with the earliest deadline (hence the name earliest deadline first) and will be scheduled first. The priority of a job is inversely related to the deadline of a task- the closer / lower the deadline, the higher the priority. An objective of earliest deadline first is to schedule tasks so that all tasks meet their deadlines.

Earliest deadline first is an optimal scheduling algorithm. It implicitly has a higher CPU utilization than fixed priority systems. If a task is schedulable, then they will be scheduled.

B. Tasks

Tasks in earliest deadline first are periodic tasks. Each task will repeatedly perform a process over a span of time. The periods are also referred to cycles and the tasks as cyclic tasks.

C. Ready Queue

The tasks that are eligible to be run are maintained in the ready queue. Tasks are with the highest priority tasks at the beginning of the queue. These are the tasks have the closest deadline. The lowest priority tasks are at the end of the queue and have the deadlines that are furthest away. The first task in the list is the highest priority, has the closest deadline, and will be the active task that is running.

Transitioning from one task to another can happen in several ways:

- If a task's cycle time completes, the ready queue is re-sorted according to the timing of the next deadline with the closest being first and the furthest away being last. The first task in the queue will be the active task and begin running. If there are more than one tasks with the highest priority (a tie), the active task will be selected arbitrarily.
- If a task is running and a new task is requested for scheduling with a closer deadline, it therefore has a higher priority than the active task. The active task will be preempted, and the new task will be started.
- If a new task arrives with the same priority / deadline, it will not be preempted and will be inserted into the ready queue at the second position behind the active task.

D. Attributes

In its simplest form, an earliest deadline first task will have three attributes that are needed to compute its priority.

- Cycle time / period – the amount of time each periodic cycle will run approximately
- Deadline – the point in time in which the next cycle must complete by
- Processing time – total computation time needed to complete all cycles and meet all deadliness. The is the worst-case execution time (WCET) to complete a task in totality

E. Advantages of Earliest Deadline First

There are several distinct advantages to earliest deadline first over fixed priority.

- In theory earliest deadline first can utilize the CPU at a higher rate than fixed priority- possibly as high as 100% utilization.
- Earliest deadline first has fewer context switches than fixed priority, reducing overhead, and improving performance.
- There is no need to define priorities for each task- it is implicitly based on its deadline.

F. Disadvantages of Earliest Deadline First

Earliest deadline first is uncommon in real time operating systems for several reasons:

- It can be difficult to implement in hardware because of its complex algorithm.
- While it may outperform fixed priority in theory, earliest deadline first is less predictable due its variable nature of scheduling. Fixed priority's constant time is more predictable, conservative, and often more appealing to operating systems designers because of this.
- Earliest deadline first offers less control because the priority of tasks cannot be explicitly set.
- It is susceptible to domino effect during times of system overload. This occurs when successive tasks miss deadlines, resulting in all tasks missing deadlines. In fixed priority, only low priority tasks will miss deadlines.

III. Research

In researching earliest deadline first, several papers can be cited for their relevance:

C. Mattihalli, "Designing and Implementing of Earliest Deadline First scheduling algorithm on Standard Linux"

An early attempt to demonstrate that earliest deadline first can be implemented in a Unix environment.

R.M. Pathan, "Design of an Efficient Ready Queue for Earliest-Deadline-First (EDF) Scheduler"

A proposal for an alternative implementation of the earliest deadline scheduler ready queue with the hope of making it more popular in industry.

J. Singh, B. Patra, and S.P. Singh, "An Algorithm to Reduce the Time Complexity of Earliest Deadline First Scheduling Algorithm in Real-Time System"

Suggested improvements for standard earliest deadline first algorithms with the goal of reducing complexity.

IV. Example

Table 1 shows three tasks with the relevant attributes needed to perform earliest deadline first scheduling.

Table 1: EDF Tasks

Task	Cycle Time	Deadline	Processing Time
T0	3	7	20
T1	2	4	5
T2	2	8	10

- Task T0 must execute 3 cycles for every 20 periods. The first deadline is at time 7.
- Task T1 must execute 2 cycles for every 5 periods. The first deadline is at time 7.
- Task T2 must execute 3 cycles for every 10 periods. The first deadline is at time 8.

We use the least common multiple (LCM) to determine how often the scheduling pattern can repeat. The LCM of processing time periods 20, 5, and 10 is 20.

Table 2 shows the first instance of the scheduling pattern with the tasks in Table 1.

- Time is the point in time
- Each task is represented by two columns:
    - The first displays the name of the task if it is running; it is also shaded when there is a deadline at that time.
    - The second is shaded when the total cycle time for an interval must be executed in

The sequence of tasks occurs as follows:

- T1 has the earliest deadline of 4; cycles 2 times 0-1
- T0 has the next deadline of 7; cycles 3 times 2-4
- T2 has the next deadline of 8; cycles 2 times 5-6
- T1 has the next deadline of 9; cycles 2 times 7-8
- Time 9 has no tasks
- T1 has the next deadline of 14; cycles 2 times 10-12
- T2 has the next deadline of 18; cycles 2 times 12-13
- Time 14 has no tasks
- T1 has the next deadline of 19; cycles 2 times 15-16
- All tasks meet their deadlines for this scheduling pattern

Table 2: EDF Schedule

Time	T0		T1		T2	
0			T1			
1			T1			
2	T0					
3	T0					
4	T0					
5					T2	
6					T2	
7			T1			
8			T1			
9						
10			T1			
11			T1			
12					T2	
13					T2	
14						
15			T1			
16			T1			
17						
18						
19						
20						

Fig. 1 shows the utilization computation for this scheduling pattern:

Fig 1. Utilization computations

$$\left(\frac{3}{20}+\frac{2}{5}+\frac{2}{10}\right)$$

$$\left(\frac{3}{20}+\frac{8}{20}+\frac{4}{20}\right)$$

$$=\frac{15}{20}=\ 85\%$$

V. Proof of Concept Implementation

Earliest deadline first can be implemented relatively simply using C#. Fig. 2 displays a basic data structure to store the task related attributes. The State attribute stores the count of times that task received the CPU during this scheduling pattern.

The source code discussed in the proof of concept is located at https://github.com/jackimburgia/earliest-deadline-first .

Fig 2. Task data structure

```
public class Task
{
      public string Name { get; set; }
      public int CycleTime { get; set; }
      public int Deadline { get; set; }
      public int ProcessingTime { get; set; }

      public int State { get; set; }
}
```

Fig. 3 displays the earliest deadline first algorithm:

- Time will loop while it is less than the LCM of the task processing times on the scheduling pattern
- Tasks are filtered where the potential next start is in the time frame
- Tasks are sorted by the next deadlines and the earliest is attempted to be retrieved
- If a task is returned, cycled through its cycle time and release control

Fig 3. Earliest deadline first algorithm

```
int time = 0;

while (time < lcm)
{
   Task next = tasks
      .Where(t => t.NextStart() <= time)
      .OrderBy(t => t.NextDeadline())
      .FirstOrDefault();

   if (next == null)
   {
      time++;
   }
   else
   {
      for (int x = 0; x < next.CycleTime; x++)
         yield return new ScheduleTime() {
               Time = time + x,
               Task = next };

      time += next.CycleTime;
      next.State++;
   }
```

Fig. 4 displays the calculations for the next deadline of a task and the next start time.

Fig 4. Next deadline and next start calculations

```
public static int NextDeadline(this Task task)
{
   int next = task.Deadline + task.State * task.ProcessingTime;
   return next;
}

public static int NextStart(this Task task)
{
   int next = task.ProcessingTime * task.State;
   return next;
}
```

Fig. 5 displays the Windows user interface where new tasks can be entered.

Fig 5. Task entry UI

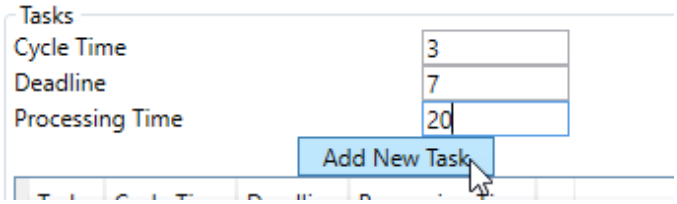


Fig. 6 displays the completed task list for this scheduling pattern.

Fig 6. Task list

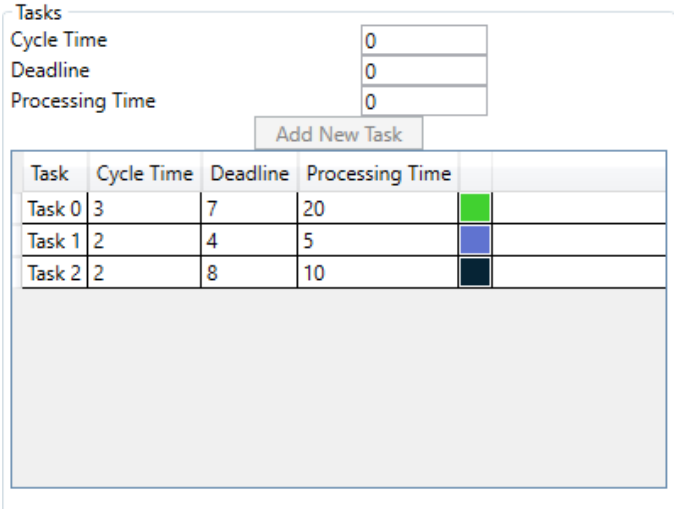
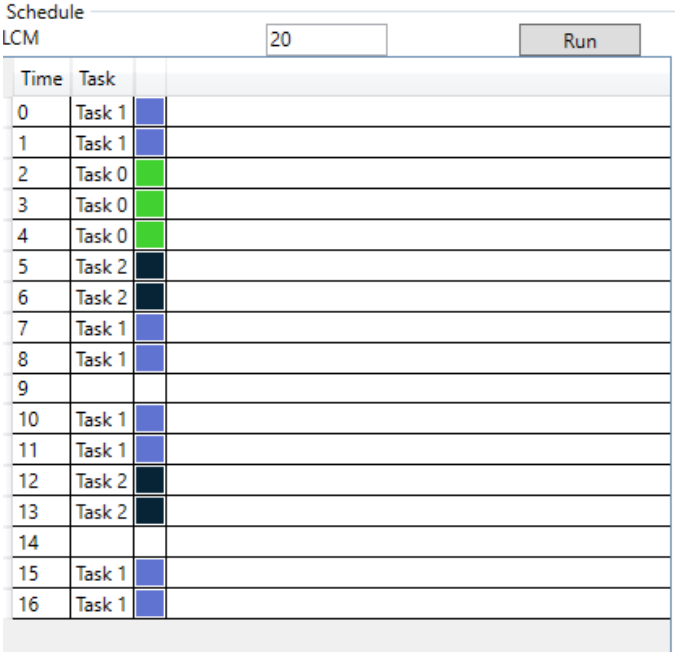


Fig. 7 displays the tasks that are performed and when they are performed in this scheduling pattern.

Fig 7.



VI. Conclusion

Earliest Deadline First offers many benefits that make it a viable and reasonable approach to scheduling in real time operating systems. The increased CPU utilization will lead to a better experience for end users, less power consumption, and free up resources or other activities. While not presently popular in commercial operating systems mostly due to the complexities of the its implementation, its benefits are worth the effort of the difficult implementation.

References

[1] X. Zhao, Y. Wei, and W. Li, "The Improved Earliest Deadline First with Virtual Deadlines Mixed-Criticality Scheduling Algorithm", 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications, pp. 444-448.

[2] C. Mattihalli, "Designing and Implementing of Earliest Deadline First scheduling algorithm on Standard Linux", 2010 IEEE/ACM International Conference on Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing, pp. 901-906.

[3] J. Singh, B. Patra, and S.P. Singh, "An Algorithm to Reduce the Time Complexity of Earliest Deadline First Scheduling Algorithm in Real-Time System", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No.2, February 2011, pp. 31-37.

[4] R.M. Pathan, "Design of an Efficient Ready Queue for Earliest-Deadline-First (EDF) Scheduler", 2016 Design, Automation & Test in Europe Conference & Exhibition, pp. 293-296.

[5] G. Lipari, "Earliest Deadline First", Scuola Superiore Sant'Anna, http://retis.sssup.it/~lipari/courses/rtos/lucidi/edf.pdf, pp. 1-11.

[6] "Earliest deadline first scheduling", https://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[7] Manoj MVR, “Earliest Deadline First (EDF)”, https://www.youtube.com/watch?v=ejPXTOcMRPA

[8] elzoughby, “Earliest-Deadline-First scheduling”, https://github.com/elzoughby/EDF-scheduling