Evaluate Efficiency of different scheduling algorithms using FreeRTOS

Lihao Guo
Electrical & Computer Engineering
University of Arizona
leolihao@email.arizona.edu

Zhaohui Yang
Electrical & Computer Engineering
University of Arizona
zhy@email.arizona.edu

Zhuangzhuang Chen
Electrical & Computer Engineering
University of Arizona
zhuangzhuangchen@email.arizona.edu

Del Ellis Spangler
Electrical & Computer Engineering
University of Arizona
dspangler@email.arizona.edu

Kyle Boyer
Electrical & Computer Engineering
University of Arizona
kairu@email.arizona.edu

***Abstract*—We compared two scheduling algorithms, EDF and DMS, running on FreeRTOS using a third-party scheduling library for this project. We tested 150 task sets across 30 different scenarios consisting of various combinations of total utilization and the number of periodic tasks. We found that DMS consistently outperformed EDF on Average Response Time and the number of runs completed with no late tasks across a broad range of utilization values and missed deadlines by a narrower margin in most other cases.**

***Index Terms*—scheduling algorithms, FreeRTOS, EDF, DMS**

I. Introduction

There are two types of scheduling modes applied in embedded systems: static scheduling and dynamic scheduling [1]. Static scheduling determines priority and timing in advance, and dynamic scheduling determines priority at execution time. For applications where the tasks are to be executed depending on conditions at runtime, dynamic scheduling can provide a more efficient solution, as it does not require prior knowledge of the actual workload.

Given any particular hardware, the capacity to perform tasks within their deadlines depends on the characteristics of the tasks themselves and the scheduling algorithm. Therefore, exploring dynamic scheduling algorithms that can take full advantage of hardware performance with acceptable computational overhead is a necessity [2].

This project aims to take two scheduling methods, EDF and DMS, which claim to be optimal solutions and compare their performance under various load conditions. The motivation is to look at the effect of task delay and determine how much the choice of algorithm matters. As the structure of the rest paper will follow: Section II describes the basic concept of EDF and DMS. Then Section III describes our experimental setup. The next section is the result of our experiments and an analysis of the results. The final part is the conclusion and the future work.

In order to test other algorithms without modifying the kernel, we are using a library called ESFree [3] that provides user-land support for task scheduling. While this is less efficient than implementing the algorithms in the kernel, this solution requires much less development time and less in-depth knowledge of the kernel. In addition, eliminating the need to modify the kernel allows this to be used on any platform with FreeRTOS support without further modification.

II. Background

A. Deadline Monotonic Scheduling (DMS)

Deadline-Monotonic Scheduling (DMS) is an optimal fixed-priority scheduling algorithm [4], meaning that priorities are designated for each task based on their relative deadlines. Tasks with shorter deadlines are assigned a higher priority. DMS is one kind of preemptive scheduling algorithm - once a new task comes, a running task might be preempted if the new task's deadline is near than it. DMS is widely applied on the single-core processor in which deadline periods of tasks are not equal to (usually shorter than) the period of tasks [3]. Although its simplification, DMS does demonstrate robust effectiveness. For example, Ref. shows that if the CPU utilization is less than 69%, DMS gives the highest priority to the periodic task with the shortest deadline and guarantees the schedulability of tasks [5].

B. Earliest Deadline First (EDF)

Earliest Deadline First (EDF) is an optimal dynamic priority scheduling policy if the CPU utilization is less than 100%, which gives the highest priority to the tasks with the earliest deadline [6]. It is a dynamic priority scheduling algorithm, usually implemented by a priority queue, i.e., heap structure. From head to tail of the priority queue, deadlines of tasks are farther from the current moment. Thus the processor only needs to process the head task in each context switch. The most calculation overhead lies in maintaining the priority queue all the time.

C. Efficient Earliest Deadline First (Efficient EDF)

In the ESFree library, an efficient-version EDF algorithm is implemented [7]. The efficient EDF module is notified when

tasks are ready, blocked, or suspended, which has two priorities for tasks: running a priority and not running a priority. All tasks start with not running as a priority, and switched in that task with the earliest absolute deadline in the ready or running state has run a priority in the following situation: 1) a task becomes blocked; 2) a task suspends itself; 3) a task becomes ready when no other tasks running; 4) a task with an earlier absolute deadline that current task becomes ready. It reduces the calculation overhead by specifically considering individual tasks and the individual statuses of tasks.

D. Related Work

A. Toma. et al. performed an experiment that compared the performance of multiple scheduling algorithms and found that the algorithms predicted to produce the best performance did not always do so under real-world conditions [8]. S. Senviel. et al. implemented an improved version of EDF, which seeks to skirt some of the limitations of the original algorithm [9].

FreeRTOS is well suited to deeply embedded real-time systems implemented in architectures equipped with microcontrollers, which generally support tasks with real-time requirements. Unfortunately, FreeRTOS does not have native support for EDF or DMS. Instead, it uses Fixed-Priority-Scheduling (FPS), configured to offer round-robin scheduling per priority level. In addition, FreeRTOS normally handles scheduling in the real-time kernel, and much of the code is responsible for implementing that written functionality in assembly [10].

K. ROBIN. et al. indicated that they had used Digilent Zed-Board(ARM Cortex-A9), Digilent Nexys 4 DDR(Xilinx MicroBlaze), and STMicroelectronics STM32 Nucleo-F401RE (ARM Cortex-M4F) to evaluate the performance of the algorithms through ESFree [3]. They mentioned that the ESFree measured average context switch overhead in two test cases, one with ten periodic tasks and the other with 30 periodic ones. They found that the worst-case context switch time for FreeRTOS and ESFree is when a higher priority task has finished its function and the CPU time goes to a lower priority task. In that scenario, the context switch overhead of fixed-priority scheduling policies such as RMS and DMS in ESFree exceed the overhead of native FreeRTOS by only 2-8%.

The documentation for ESFree also notes that in certain cases, it may be preferable to use the default FPS algorithm over a theoretically faster one to avoid the overhead of performing scheduling outside the kernel. Since we will only be comparing schedulers available through ESFRee, this overhead should not affect the experiment's outcome.

III. Experimental Setup

Our setup consists of FreeRTOS running on a POSIX environment, with ESFree providing access to EDF and DMS user-land implementations. This stack then runs tasks designed to do a particular number of ticks worth of work. The parameters for the tasks, including utilization, period, and worst-case execution time, are generated programmatically from the scenario description.

To generate each task set, we used an algorithm called UUniFast, which takes a desired total utilization and several tasks and returns a set of uniformly distributed utilization values that sum to the specified total. The principle of the algorithm is to sample the sum of the utilization value of $n-1$ tasks first and then set the utilization value of tasks to the difference between the total utilization value and the sampling value. We selected task periods from a gaussian distribution centered on 500 ticks. Moreover, we specifically decided on 500 to reduce the number of very small duration tasks, thus reducing the effect of measurement overhead on the results.

To complete our experiment, we designed to run each task with 10000 ticks, and we tracked the average response time, the total number of tasks executed, maximum lateness, and the number of late tasks for each run. Each run lasted 10000 ticks. For each run, we tracked the average. If the number of late tasks is zero, the application does not miss the task; otherwise, it misses the task number.

After we obtain all results from the experiment, we could calculate the success ratio by following the formula:

$$Success\ Ratio = \frac{\#\ task\ sets\ with\ no\ missed\ deadline}{\#\ generated\ task\ sets}$$

We ran each algorithm 150 times. We tested them each at 6 different utilization values ranging from 0.75 to 1, in increments of 0.05. We tested 5 different numbers of tasks for each utilization value, ranging from 5 to 25 in increments of 5. Each combination of utilization and task count ran 5 times to compensate for the randomness in our task generation.

IV. Results & Analysis

Table I is our result from the experiment. The first column is the utilization value, the second column is the number of tasks, and the third column is the success ratio. We can see that even at lower utilization (below 0.8), both algorithms occasionally had late tasks, which were not schedulable. For EDF, when utilization increases, the success ratio gradually decreases demonstratively. For DMS, when utilization increases, the success ratio does not significantly decrease.

TABLE I
SUCCESS RATIO VS. UTILIZATION FOR DMS AND EDF

Scheduler	Utilization	Success Ratio
DMS	0.75	0.992
	0.80	0.992
	0.85	0.993
	0.90	0.990
	0.95	0.990
	1.00	0.988
EDF	0.75	0.991
	0.80	0.988
	0.85	0.986
	0.90	0.983
	0.95	0.984
	1.00	0.983

Table II shows the success ratio as a function of the number of tasks has run. For DMS, when the number of tasks

increases, the success ratio increases. For EDF, when number task increases, the success ratio increases significantly.

TABLE II
SUCCESS RATIO VS. NUMBER OF TASKS FOR DMS AND EDF

Scheduler	# of Tasks	Success Ratio
DMS	5	0.987
	10	0.990
	15	0.991
	20	0.993
	25	0.993
EDF	5	0.979
	10	0.986
	15	0.985
	20	0.988
	25	0.991

As we compared those tables from the experimental result, It is easy to find that DMS's performance is better and more robust. The DMS does not change when the number of tasks or utilization changes.

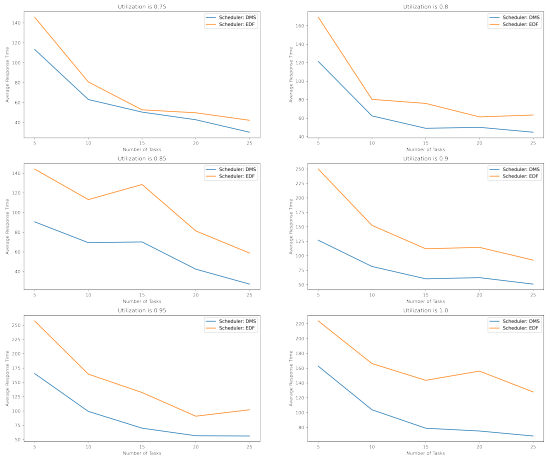

Fig. 1. Average Response Time

Figure 1 shows that DMS also has a lower average response time across all task sets in this experiment. We can see that DMS is better not only in completed task rate but also τ (utilization). As we can see, when the number of tasks increases to 15, τ tends to be stable.

Figure 2 shows that DMS also has a lower average response time across all task sets by the scheduler. In the DMS case, τ is universally smaller than EDF, which coincides with the conclusion aforementioned that DMS performs more stably on utilization.

As can be seen in Figure 3, even when neither algorithm can successfully meet all deadlines, DMS generally gets closer than EDF in most cases.

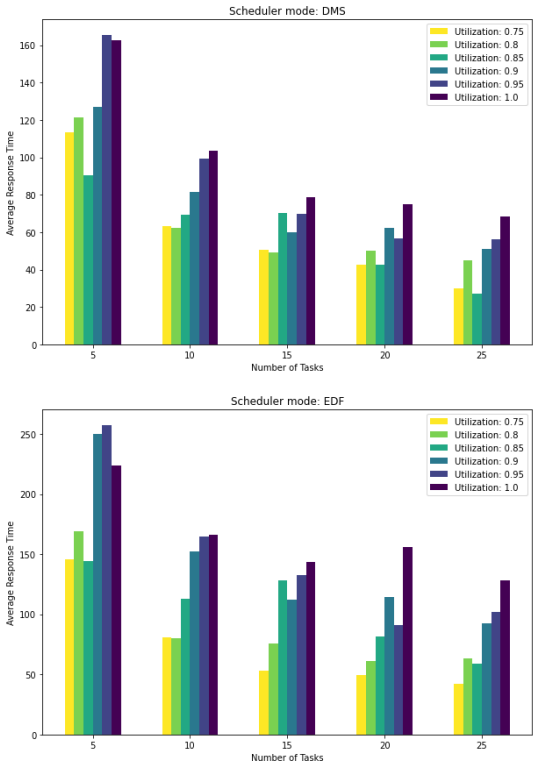

Fig. 2. Average response time τ (by scheduler)

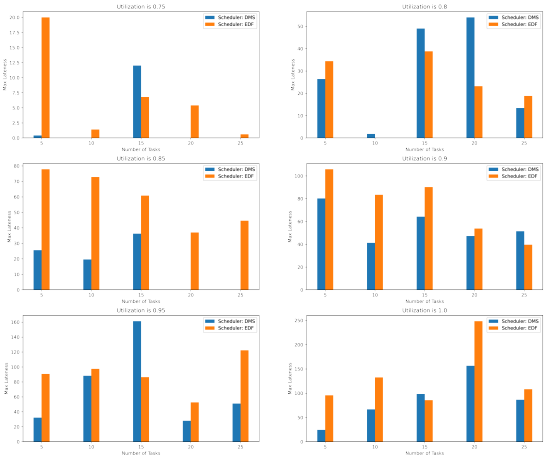

Fig. 3. Maximum Lateness

Figure 4 shows that in the case of DMS, r is universally smaller than EDF, which coincides with the conclusion mentioned above that DMS performs more stably on utilization.

Figure 5 shows that the variance of max lateness is evidently smaller in the EDF case than that for DMS, which indicates that EDF is more stable although its overall max lateness is larger DMS.

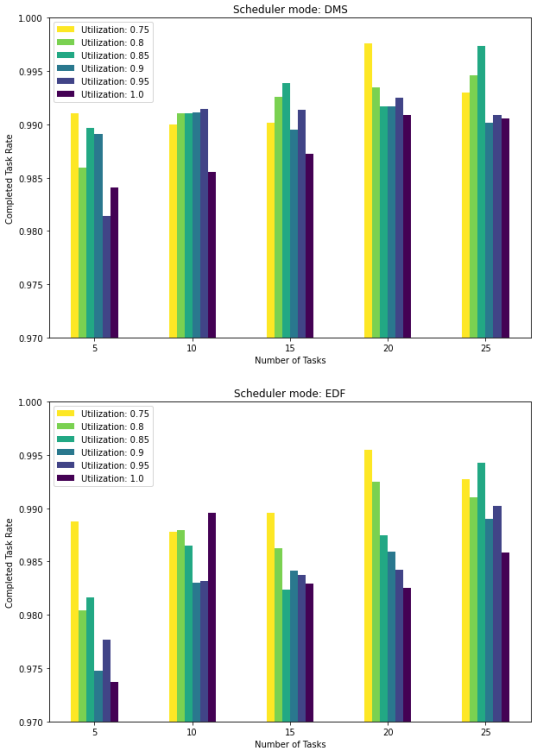

Fig. 4. Completed task rate r (by scheduler)

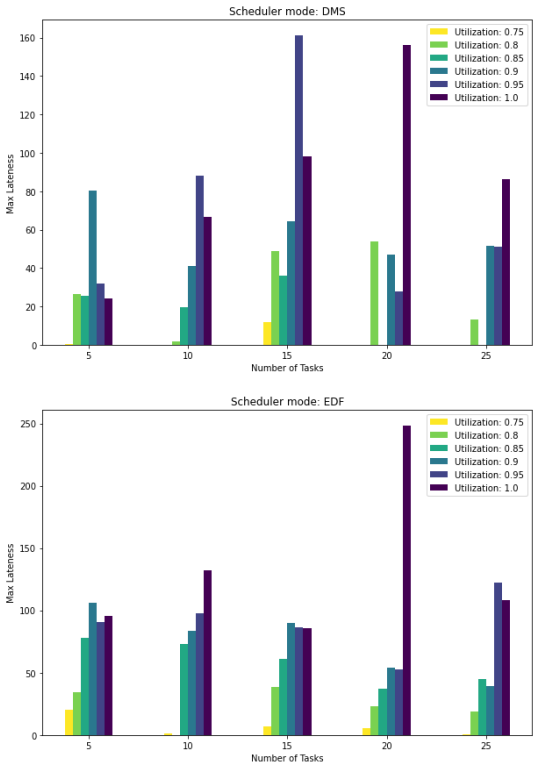

Fig. 5. Max lateness r (by scheduler)

V. Conclusions & Future Work

We successfully tested EDF and DMS under various workload settings. We found that DMS evidently outperforms every crucial metric we concentrated on, i.e., the average response time, the scheduled tasks success ratio, the max lateness time, and more. Particularly, DMS performs a more stable scheduling effect other than universally better metric. Compared with EDF, DMS is a simple static single-processor scheduling algorithm and simpler when implemented; Thus, it achieves more stable performance even when utilization and the number of tasks change significantly. Our work quantitatively illustrated specific differences between two representative prototypes of the real-time scheduling algorithm, hopefully guiding significance in designing practical real-time embedded systems.

For future works, it would be interesting to perform this test with a larger variety of algorithms and extend both the range of utilization values tested to see how they perform at very low utilization. Another potential improvement to our implementation would be determining task periods from domain-specific datasets rather than using a random variable under the Gaussian distribution. Better yet, we hope to explore whether there are derivative EDF models (e.g., Efficient EDF) that can compensate for the disadvantages of the native EDF and achieve the comparative performance of DMS.

References

[1] G. Oliveira and G. Lima, "Evaluation of scheduling algorithms for embedded freertos-based systems," in *2020 X Brazilian Symposium on Computing Systems Engineering (SBESC)*, 2020, pp. 1–8.

[2] R. Belagali, S. Kulkarni, V. Hegde, and G. Mishra, "Implementation and validation of dynamic scheduler based on lst on freertos," pp. 325–330, 2016.

[3] R. KASE, "Efficient scheduling library for freertos - diva-portal.org," 2016. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:1085303/FULLTEXT01.pdf

[4] N. C. Audsley, A. Burns, M. Richardson, and A. Wellings, "Deadline monotonic scheduling," 1990.

[5] G. Buttazzo, "Hard real-time computing systems: predictable scheduling algorithms and applications."

[6] D. Faggioli, M. Trimarchi, F. Checconi, M. Bertogna, and A. Mancina, "An implementation of the earliest deadline first algorithm in linux," in *Proceedings of the 2009 ACM symposium on Applied Computing*, 2009, pp. 1984–1989.

[7] D. Thakor and A. Shah, "D_edf: An efficient scheduling algorithm for real-time multiprocessor system," in *2011 World Congress on Information and Communication Technologies*. IEEE, 2011, pp. 1044–1049.

[8] V. M. Anas Toma and J.-J. Chen, "implementation and evaluation of multi-mode real-time." [Online]. Available: https://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2018-toma-ospert.pdf

[9] Q. L.-S. Steve Senviel, "real-time scheduling for embedded systems using enhanced edf," in *real-time scheduling for embedded systems using enhanced edf*, 2018. [Online]. Available: https://www.cse.scu.edu/fwang1/studentProjects/Schedule_embedded RealTimeEDF_10s.pdf

[10] "Priority based preemptive RTOS scheduler." [Online]. Available: https://www.freertos.org/implementation/a00005.html