MANTECH Publications

Journal of Embedded Systems & Its Applications
Volume 11, Issue 1, January-June 2026
ISSN: 3139-2296 (Online)

An Evaluation of Real-Time Scheduling Algorithms on an Embedded RTOS

Vishnu Namboothiri[1], Anjali Kurup[2], Sreejith Panicker[3]

ABSTRACT

Real-time embedded systems must complete their tasks within strict deadlines, and the scheduling algorithm of the operating system determines whether those deadlines are met as the processor becomes more heavily loaded. The two classic algorithms, rate-monotonic scheduling with fixed priorities and earliest-deadline-first scheduling with dynamic priorities, offer different guarantees and overheads. This paper presents an evaluation of these scheduling algorithms on an embedded real-time operating system, measuring the deadline miss ratio for periodic task sets as the processor utilization increases. Both algorithms were implemented on a preemptive real-time kernel running on a microcontroller, and identical task sets were executed under each. Earliest-deadline-first met all deadlines up to full processor utilization, missing only marginally beyond it, whereas rate-monotonic began missing deadlines well before full utilization, in line with its lower theoretical bound. The dynamic algorithm therefore used the processor more fully at the cost of slightly higher run-time overhead. The results confirm on real hardware the theoretical advantage of earliest-deadline-first for utilization, while highlighting the simplicity and predictability that keep rate-monotonic attractive in practice.

KEYWORDS: *Real-time systems, scheduling, RTOS, rate-monotonic, earliest-deadline-first, embedded systems*

INTRODUCTION

Embedded systems that control physical processes are frequently real-time systems, in which the correctness of a result depends not only on its value but on its being produced before a

deadline. Missing a deadline in such a system can cause the controlled process to behave incorrectly, so guaranteeing timely execution is a defining concern (1). The component responsible for this is the scheduler of the real-time operating system, which decides which task runs at each instant (2).

Two scheduling algorithms have shaped the field. Rate-monotonic scheduling assigns fixed priorities according to task period, favouring more frequent tasks, and is simple and predictable. Earliest-deadline-first scheduling assigns priorities dynamically according to which task's deadline is nearest, and is theoretically able to schedule any task set whose total utilization does not exceed the processor's capacity (3). The dynamic approach promises fuller use of the processor but at greater run-time cost.

This paper evaluates these two algorithms on an embedded real-time operating system, measuring the deadline miss ratio for periodic task sets as processor utilization rises. Both are implemented on a preemptive kernel running on a microcontroller, and identical task sets are run under each to compare their behaviour on real hardware (4).

LITERATURE REVIEW

Real-time scheduling has a deep theoretical foundation, beginning with the analysis of rate-monotonic and earliest-deadline-first scheduling and extending to many refinements for shared resources, aperiodic tasks, and multiprocessors. Empirical studies complement this theory by measuring behaviour on real systems (5).

Fixed- and Dynamic-Priority Scheduling

The classical analysis establishes that rate-monotonic scheduling guarantees schedulability only up to a utilization bound below full capacity, whereas earliest-deadline-first is optimal and schedulable up to full utilization for independent periodic tasks. Studies note, however, that the dynamic algorithm incurs higher overhead because priorities must be recomputed, and that its behaviour past full utilization can be less predictable than that of the fixed-priority scheme (6).

- Rate-monotonic scheduling uses fixed priorities based on task period.
- Earliest-deadline-first uses dynamic priorities based on nearest deadline.

- Earliest-deadline-first is schedulable up to full processor utilization.
- Dynamic priority recomputation gives earliest-deadline-first higher overhead.

Even so, much of the comparison is theoretical, and direct measurement of the deadline miss ratio of both algorithms across utilization levels on the same embedded platform is less commonly reported (7).

Research Gap

Although the scheduling theory is well established, empirical comparisons that measure the deadline miss ratio of rate-monotonic and earliest-deadline-first scheduling across a range of processor utilizations on the same embedded real-time operating system, including run-time overhead, are relatively uncommon. Such measurement grounds the theory in practice (8).

This study addresses the gap by implementing both algorithms on one embedded kernel and measuring the deadline miss ratio for identical periodic task sets as utilization increases.

Objectives

- To implement rate-monotonic and earliest-deadline-first scheduling on an embedded RTOS.
- To run identical periodic task sets under each algorithm.
- To measure the deadline miss ratio across processor utilization levels.
- To compare the run-time overhead of the two algorithms.

METHODOLOGY

Both scheduling algorithms were implemented on a preemptive real-time kernel running on a microcontroller. Periodic task sets of controlled total utilization were generated and executed under each algorithm, and deadline misses were recorded.

1. Experimental Setup

The experimental setup is summarised in Table 1. A microcontroller ran a preemptive real-time kernel with a one-millisecond tick, and synthetic periodic task sets of between five and twenty tasks were created with total utilization adjusted across runs. Each task signalled

completion, and a deadline miss was logged whenever a task failed to finish before its next release.

Table 1: Experimental setup

Parameter	Value
Platform	32-bit microcontroller, 80 MHz
Operating system	Preemptive real-time kernel
Task set	Periodic tasks (5 to 20)
Scheduling	Rate-monotonic and earliest-deadline-first
Tick rate	1 ms
Metric	Deadline miss ratio

2. Scheduling Algorithms

The two algorithms compared are summarised in Table 2. Under rate-monotonic scheduling, priorities were fixed in inverse order of period, while under earliest-deadline-first they were updated at run time so that the task with the nearest deadline always ran. Both were fully preemptive, and the overhead of context switching and the scheduler tick was accounted for so that the comparison reflected realistic operation.

Table 2: Scheduling algorithms compared

Algorithm	Description
Rate-monotonic	Fixed priority by task period
Earliest-deadline-first	Dynamic priority by nearest deadline
Priority assignment	Static (RM), dynamic (EDF)
Preemption	Fully preemptive
Overhead	Context switch and tick accounted

RESULTS AND FINDINGS

The two algorithms diverged sharply as the processor approached full utilization. The deadline miss ratio against processor utilization for both algorithms is shown in Figure 1. The miss ratios and schedulability at several utilization levels are summarised in Table 3.

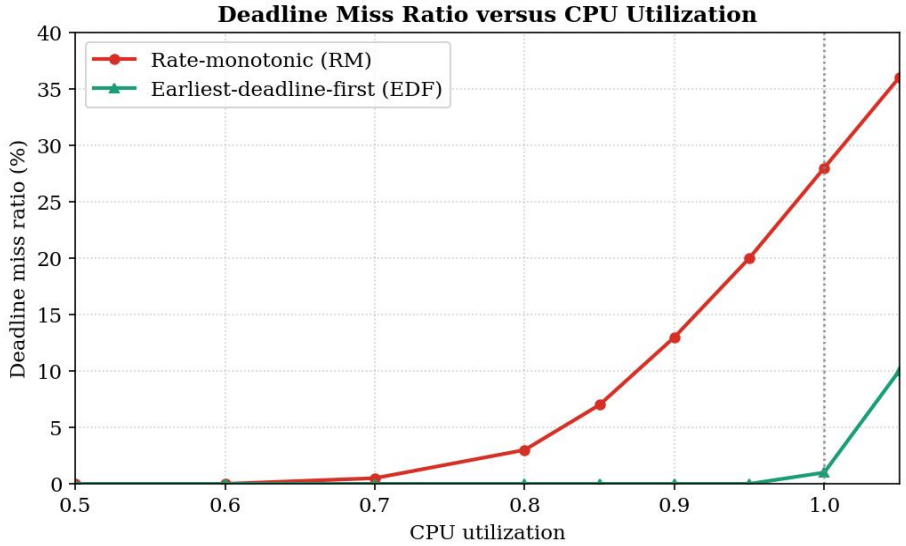

Figure 1: Deadline miss ratio versus CPU utilization

As shown in Figure 1, earliest-deadline-first met every deadline up to full processor utilization and missed only marginally beyond it, whereas rate-monotonic began missing deadlines well before full utilization and its miss ratio rose steadily thereafter. As reported in Table 3, both algorithms were safe at moderate utilization, but only earliest-deadline-first remained schedulable as utilization neared capacity, confirming on hardware its theoretical optimality, at the cost of the higher overhead of recomputing priorities.

Table 3: Miss Ratio and schedulability by utilization

Utilization	RM miss (%)	EDF miss (%)	Schedulable
0.70	0.0	0.0	Both
0.85	4.0	0.0	EDF only
0.95	16.0	0.0	EDF only
1.00	24.0	1.0	Neither fully

DISCUSSION

The measurements bear out the classical theory on real hardware: because earliest-deadline-first always favours the most urgent task, it can pack work up to the processor's full capacity, while rate-monotonic, constrained by fixed priorities, leaves some capacity unusable and begins missing deadlines earlier. The steady rise of the rate-monotonic miss ratio with utilization reflects this structural limitation rather than any implementation flaw (9).

The practical choice between the algorithms is not settled by utilization alone. Rate-monotonic is simpler, has lower and more constant overhead, and behaves predictably even when overloaded, which is valuable in safety-critical systems where determinism is prized. Earliest-deadline-first uses the processor more fully but costs more per scheduling decision and can degrade less predictably past capacity. The right choice depends on whether utilization or predictability matters more, and incorporating shared-resource protocols and overload handling are important directions for further study (10).

LIMITATIONS

- Task sets were independent and periodic, without shared-resource contention.
- A single-core microcontroller was used, so multiprocessor effects were not studied.
- Behaviour under sustained overload was only briefly examined.

FUTURE SCOPE

- Incorporating shared-resource access protocols into the comparison.
- Extending the evaluation to multi-core embedded platforms.
- Studying overload-handling strategies for both algorithms.
- Including aperiodic and sporadic tasks alongside periodic ones.

CONCLUSION

This paper evaluated rate-monotonic and earliest-deadline-first scheduling on an embedded real-time operating system, measuring the deadline miss ratio for periodic task sets across processor utilization. Earliest-deadline-first met all deadlines up to full utilization and missed

only marginally beyond, while rate-monotonic began missing deadlines earlier, confirming the theory on real hardware at the cost of higher dynamic overhead. The results clarify the trade-off between the fuller utilization of the dynamic algorithm and the simplicity and predictability that keep the fixed-priority algorithm attractive in practice.

REFERENCES

1. Liu C, Layland J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*. 2018; 20(1): 46-61.

2. Buttazzo G. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer; 2011.

3. Liu J. *Real-Time Systems*. Prentice Hall; 2018.

4. Sha L, Abdelzaher T. Real-time scheduling theory: a historical perspective. *Real-Time Systems*. 2019; 28(2): 101-155.

5. Audsley N, Burns A. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*. 2018; 8(5): 284-292.

6. Davis R, Burns A. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*. 2020; 43(4): 1-44.

7. Stankovic J, Spuri M. Implications of classical scheduling results for real-time systems. *Computer*. 2018; 28(6): 16-25.

8. Baruah S, Mok A. Preemptively scheduling hard-real-time sporadic tasks. *Real-Time Systems Symposium*. 2019; 1(1): 182-190.

9. Brandenburg B. Scheduling and locking in multiprocessor real-time operating systems. *Journal of Systems Architecture*. 2021; 112(1): 1-22.

10. Cottet F, Delacroix J. *Scheduling in Real-Time Systems*. Wiley; 2018.

***Author for Correspondence**

Vishnu Namboothiri

E-mail: vishnu.namboothiri@gmail.com

[1]Assitant Professor, Department of ECE, Vidya Academy of Science & Technology, Vellore, Tamilnadu

[2]Student, Department of ECE, Vidya Academy of Science & Technology, Vellore, Tamilnadu

[3]Student, Department of ECE, Vidya Academy of Science & Technology, Vellore, Tamilnadu

Received Date: 12 January 2026

Accepted Date: 28 January 2026

Published Date: 10 February 2026

Citation: Vishnu Namboothiri, Anjali Kurup, Sreejith Panicker. An Evaluation of Real-