Reducing the gap between theory and practice in real-time systems with MARS

Giann Spilere Nandi, David Pereira, José Proença, Eduardo Tovar, Luís Nogueira
CISTER, Polytechnic Institute of Porto, Porto, Portugal
{giann, drp, pro, emt, lmn}@isep.ipp.pt

***Abstract*—A significant number of dependable systems rely on scheduling algorithms to achieve temporal correctness. Despite their relevance in real-world applications, only a narrow subset of the works in the literature of real-time systems are readily available to be reproduced in real-world hardware platforms. This lack of support not only hinders the reproducibility of research results, but also reduces the opportunity for new platform-specific research directions to emerge. In this work we discuss the use and development of an open-source tool named MARS capable of porting various scheduling tests and algorithms to hardware platforms used in distributed real-time dependable systems.**

I. Introduction

Guaranteeing temporal correctness is one of the cornerstones of dependable systems. To achieve that, domains like avionics, automotive, healthcare, and industrial automation employ scheduling algorithms to ensure that systems respond to events within strict time constraints [1]. The fundamental aspects used by these algorithms have been extensively researched and are widely discussed in the literature of real-time systems, which include an ample variety of works that propose scheduling algorithms and schedulability analysis tests for several distinct abstract system configurations [12], [4].

Despite their relevance and direct impact in real-world applications, not every work in the literature describing scheduling algorithms is accompanied by an equivalent implementation in a real-world platform. For instance, as far as the authors know, despite the significant amount of work published in the field of mode-change protocols in the past two decades [11], [5], [2], no public library implementing these concepts is available for the community to experiment and reproduce their results in hardware platforms. This gap between the advances in the theoretical domain and their readiness to be validated in real-world scenarios not only hinders the reproducibility of many studies, but also obstructs research opportunities that could uncover platform-specific problems and optimizations.

As part of the development of an open-source project named MARS[1], we are implementing a compiler for a domain-specific language that, among other functionalities, performs the schedulability analysis of a user-defined set of real-time tasks. We achieve that by combining custom implementations of scheduling analysis algorithms available in the literature and integrating third-party schedulability analysis tools.

[1]https://bitbucket.org/mars-language/marscomp/

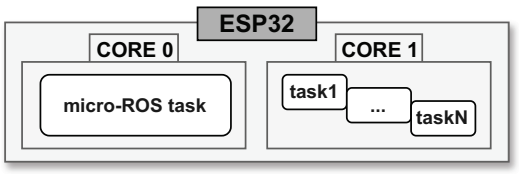

Fig. 1. Illustration of how MARS allocates the symmetrical cores of the ESP32 microcontroller to run soft real-time micro-ROS task on CORE 0 and other hard real-time tasks on CORE 1.

The remainder of this work briefly presents implementation details of MARS and discuss how its toolset could be extended to help reduce the gap between theory and practice of real-time systems in the context of distributed extremely resource-constrained devices used in dependable real-world applications.

II. Proposed Work and Initial Results

MARS is an open-source tool and domain-specific language designed to ease the generation and safe instrumentation of runtime monitors into real-time distributed resource-constrained devices. Due to space constraints, this document will not go into details about the syntax and semantics of the domain-specific language. Instead, the focus will be on how MARS can be used and extended to perform schedulability analysis of various configurations of embedded real-time systems.

Designed to be compatible with a wide variety of use cases and aligned with current industry and academic trends [1], MARS integrates tools that allow developers to generate C code that implements the behavior of distributed nodes executing on single-core and multi-core systems running on top of a real-time operating system. More specifically, MARS leverages i) the publish-subscribe communication scheme implemented by micro-ROS [14], which allows extremely resource-constrained devices to participate in ROS2[2] networks; ii) the API provided by ESP-IDF 4.3[3] to port applications to our currently supported hardware platform: the ESP32 microcontroller[4]; and finally iii) the FreeRTOS API, which

[2]https://www.ros.org

[3]https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/freertos.html

[4]Development kits based on ESP32 are powered by an Xtensa 32-bit LX6 symmetrical dual-core processor, up to 16MB of integrated memory, 512KB SRAM, and integrated WiFi and Bluetooth Low Energy.

2833-292X/24/$31.00 ©2024 IEEE
DOI 10.1109/DSN-S60304.2024.00021

provides, on a tiny footprint, a library with basic functions to control the ESP32's processor scheduling.

Figure 1 illustrates how MARS currently utilizes the two symmetrical cores of the ESP32. As of now, *CORE 0* is used exclusively for micro-ROS and its internal soft-real time user-defined callback execution order[5]. These include functionalities like managing the wifi communication, managing periodic user-implemented functions, running runtime monitors, publishing data on topics, and listening to new data on the topics the node currently subscribes to. On *CORE 1*, MARS allows developers to specify a set of hard real-time tasks, composed of periodic and sporadic monitors and user-implemented functions. This separation of concerns allows ESP32 nodes to participate in distributed ROS2 applications under soft real-time constraints while also allowing developers to execute local hard real-time tasks respecting a given scheduling policy (e.g., rate-monotonic).

Figure 2 illustrates, at a high level of abstraction, how MARS performs its scheduling analysis. First, a set of real-time tasks and their scheduling parameters (e.g., task priority, worst-case execution time, deadline) are fed as input to MARS, which proceeds to analyze it according to a user-selected scheduling policy. Finally, depending on the results, MARS either confirms the schedulability of the task set and generates the respective C code to be flashed on the ESP32, or informs that the system is not schedulable and aborts the process.

So far, the development efforts have been focused on the implementation and integration of single-core scheduling tests, starting with the classic analysis for single-core fixed-priority preemptive tasks [3]. We have also integrated the single-core fixed-priority non-preemptive schedulability analysis tool proposed in [10], but the use of non-preemptive tasks in the context of micro-ROS/ESP-IDF applications still requires further validation.

Further development efforts for MARS will consist of integrating other well-known scheduling algorithms, prioritizing those not natively supported by FreeRTOS (e.g., earliest deadline first, mode-change protocols), and either extending or integrating previous related efforts and tools in the literature [6], [9], [7], [13]. To further contextualize, while there have been efforts for simulating and verifying the schedulability of task sets (most notably Cheddar[13]), these efforts are not directly linked to toolchains capable of generating code ready to be deployed in distributed extremely resource-constrained devices.

Future research directions also include investigating the worst-case execution time of the micro-ROS task and how it scales when adding publishers, subscribers, or other micro-ROS functionalities. This would provide a better understanding of how to safely extend our efforts to include *CORE 0* and *CORE 1* in the scheduling process of hard reak-time tasks. Furthermore, updating to the newest APIs of ESP-IDF and FreeRTOS would allow a finer control over the hardware and system scheudling. Finally, we also want to equip the compiler with functionalities aimed at easing the chore of defining scheduling parameters, like the flexible parameter assignment proposed in [8], which could help developers to safely fine-tune the temporal parameters of their task sets.

[5]We point the reader to the micro-ROS documentation for more details https://micro.ros.org/docs/concepts/client_library/execution_management/

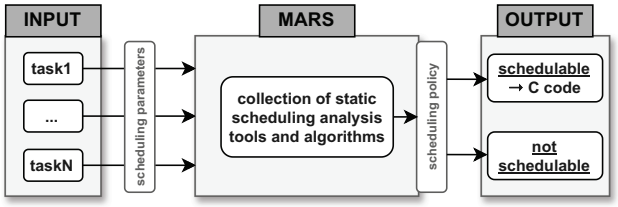

Fig. 2. High level illustration of the execution flow adopted by MARS to analyze a set of tasks and their respective scheduling parameters, evaluate their schedulability according to a user-selected scheduling policy, and the output generated according to the analysis results.

Although the current development efforts of MARS are focused on the ESP32 platform and single-core analysis, it being open-source should be interpreted as an invitation for anyone to modify its source code and integrate it into their desired target platforms and toolchains.

We hope that MARS can serve as a common framework for developers to i) reproduce works in the literature in the context of the embedded platforms of their choice; ii) apply a wider variety of scheduling algorithms and tests in their practical use-case scenarios; and iii) further explore research opportunities, potentially uncovering platform-specific problems that could lead to safety violations in safety-critical applications, or even discovering new optimization possibilities.

III. Conclusion

In this work we discussed the importance of reducing the gap between theory and practice in the context of real-time dependable systems. We have also briefly introduced a tool named MARS, which implements a collection of scheduling algorithms and tests for applications running soft and hard real-time tasks on an ESP32 platform. Due to the open-source nature of MARS, we hope that others extended its source-code to support additional hardware platforms and implement more real-time concepts potentially useful for the area of dependable distributed systems running on extremely resource-constrained devices

Acknowledgment

This work was partially supported by project Route 25 (ref. TRB/2022/00061-C645463824-00000063), funded by the EU/Next Generation, within call n.º 02/C05-i01/2022 of the Recovery and Resilience Plan (RRP); and by grant 2022.13599.BD, financed by FCT through the European Social Fund (ESF) and the Regional Operational Programme (ROP) Norte 2020. Disclaimer: this document reflects only the authors' view and the Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. A comprehensive survey of industry practice in real-time systems. *Real-Time Systems*, 58(3):358–398, November 2021.

[2] Hyeongboo Baek, Kang G. Shin, and Jinkyu Lee. Response-time analysis for multi-mode tasks in real-time multiprocessor systems. *IEEE Access*, 8:86111–86129, 2020.

[3] Robert I. Davis. A review of fixed priority and edf scheduling for hard real-time uniprocessor systems. *SIGBED Rev.*, 11(1):8–19, feb 2014.

[4] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4), oct 2011.

[5] Wen-Hung Huang and Jian-Jia Chen. Techniques for schedulability analysis in mode change systems under fixed-priority scheduling. In *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 176–186, 2015.

[6] Rafia Inam, Mikael Sjödin, and Reinder J. Bril. Mode-change mechanisms support for hierarchical freertos implementation. In *:*, pages Article number 6648010–, 2013.

[7] Robin Kase. Efficient scheduling library for freertos, 2016.

[8] Mitra Nasri. On flexible and robust parameter assignment for periodic real-time components. *SIGBED Rev.*, 14(3):8–15, nov 2017.

[9] Francisco E. Páez, José M. Urriza, Ricardo Cayssials, and Javier D. Orozco. Freertos user mode scheduler for mixed critical systems. In *2015 Sixth Argentine Conference on Embedded Systems (CASE)*, pages 37–42, 2015.

[10] Sayra Ranjha, Geoffrey Nelissen, and Mitra Nasri. Partial-order reduction for schedule-abstraction-based response-time analyses of non-preemptive tasks. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 121–132, 2022.

[11] Jorge Real and Alfons Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, March 2004.

[12] Mehrin Rouhifar and Reza Ravanmehr. A survey on scheduling approaches for hard real-time systems. *International Journal of Computer Applications*, 131(17):41–48, 2015.

[13] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. *ACM SIGAda Ada Letters*, XXIV(4):1–8, November 2004.

[14] Jan Staschulat, Ingo Lütkebohle, and Ralph Lange. The rclc executor: Domain-specific deterministic scheduling mechanisms for ros applications on microcontrollers: work-in-progress. In *2020 International Conference on Embedded Software (EMSOFT)*, pages 18–19, 2020.