Bare Metal vs. Embedded OS: Which is Ideal for Your Project?

https://github.com/OlgerZG/Bare-Metal-vs.-Embedded-OS-Which-is-Ideal-for-Your-Project-

Olger Zambrano
Department of Applied Mathematics and Computer Science
Technical University of Denmark
Kongens Lyngby, Denmark
s233461@dtu.dk

***Abstract*—In this paper, an analysis will be conducted to compare the fundamental differences between bare-metal programming and the use of an embedded operating systems. We will explore the advantages and disadvantages of each approach, as well as provide guidance to help you determine which is the best choice for your specific project requirements. This analysis aims to assist developers, engineers, and decision-makers in making informed choices when developing embedded systems.**

***Index Terms*—**Embedded Software, Performance Comparison, Bare Metal, Embedded OS, FreeRTOS

I. Introduction

Embedded systems development involves different programming approaches, with two prominent approaches, bare metal programming and implement an embedded operating system. In this context, bare-metal programming refers to the practice of writing code directly for the hardware without relying on an intermediary operating system layer, also know as kernel [1]. The choice between these approaches is crucial for your project's development and long-term viability. It not only constrains the initial design, but also defines the system's ability to adapt future requirements and changes in technology. For this reason, making an informed choice becomes important when deciding whether to use an operating system or not.

II. Literature review

This sections aims to summarize the theoretical advantages and disadvantages of each choice, which will contrasted with the practical aspects exposed in section III.

A. Advantages of Bare-Metal Programming

Bare-metal programming offers several advantages that make it a preferred choice for certain embedded systems. Understanding these benefits is crucial for making an informed decision in embedded systems development.

1) Performance Efficiency: One of the primary advantages of bare-metal programming is the potential for superior performance. Since there is no overhead from an operating system, the system's resources are fully dedicated to executing the application code. This results in faster response times and reduced latency, critical factors for real-time applications such as those found in industrial control systems or automotive electronics.

2) Resource Utilization: Embedded systems often operate in resource-constrained environments. Bare-metal programming allows developers to have fine-grained control over system resources, minimizing unnecessary overhead. Without the layers of an operating system, the code can be optimized to run efficiently on the specific hardware, making more effective use of limited resources like memory and processing power.

3) Deterministic Execution: For applications where deterministic behavior is crucial, such as in safety-critical systems or real-time control systems, bare-metal programming provides a high level of predictability. The absence of an operating system scheduler means that the system's behavior is more deterministic, making it easier to analyze and guarantee performance under different conditions [2].

4) Reduced Footprint: Bare-metal applications typically have a smaller footprint compared to those running on an operating system. This is advantageous in scenarios where memory constraints are tight, common in many embedded systems. The absence of unnecessary features and services results in a streamlined application, leaving more space for the core functionality of the system.

B. Disadvantages of Bare-Metal Programming

While bare-metal programming has its merits, it is essential to consider its limitations and the scenarios where it might not be the most suitable choice.

1) Limited Abstraction: Bare-metal programming lacks the abstraction provided by an operating system, making it more challenging to develop complex applications. Tasks such as memory management, multitasking, and device drivers become the responsibility of the developer, potentially increasing development time and introducing a higher risk of errors [3].

2) Portability Challenges: Code developed for a specific hardware configuration may not be easily portable to other platforms. This lack of portability can be a significant drawback if the project requirements change, or if there is a need to reuse code on different embedded systems with varying hardware architectures [3].

3) Development Complexity: Bare-metal programming often involves low-level hardware interactions, requiring in-depth knowledge of the target platform. This can result in more complex development and debugging processes compared to using higher-level abstractions provided by operating systems.

4) Scalability Issues: As projects grow in complexity, scalability becomes a concern. Bare-metal programming might face challenges when scaling up to larger, more intricate systems, where the benefits of abstraction and management provided by an operating system become more apparent.

C. Advantages of Embedded Operating Systems

In contrast to bare-metal programming, using an embedded operating system introduces a layer of abstraction between the application code and the hardware. This abstraction comes with its own set of advantages, making it a suitable choice for certain embedded systems.

1) Abstraction and Simplified Development: One of the primary advantages of using an embedded operating system is the abstraction it provides. Developers can leverage higher-level APIs and services, making it easier to develop complex applications. Tasks such as multitasking, memory management, and device driver handling are abstracted by the operating system, reducing the complexity of the development process [4].

2) Portability: Embedded operating systems enhance portability by providing a standardized interface between the application and the hardware. Code developed on one platform can be more easily adapted to run on different systems that support the same operating system, facilitating code reuse and reducing development time [3].

3) Rich Feature Set: Embedded operating systems come with a rich set of features and services that can enhance the functionality of an embedded system. This includes support for networking protocols, file systems, and inter-process communication, which can be crucial for applications requiring communication with other devices or systems [4].

4) Scalability: Embedded operating systems are designed to scale with the complexity of the project. As the requirements of the embedded system grow, an operating system can provide the necessary abstractions and management to handle more sophisticated applications and a larger number of tasks [4].

D. Disadvantages of Embedded Operating Systems

While embedded operating systems offer several advantages, it is essential to be aware of their limitations and when bare-metal programming might be a more suitable choice [4].

1) Resource Overhead: One of the significant drawbacks of embedded operating systems is the resource overhead they introduce. The additional layer between the application and the hardware consumes memory and processing power, potentially limiting the use of these systems in resource-constrained environments.

2) Performance Considerations: The abstraction provided by an operating system comes at a cost of increased processing time and potential latency. In scenarios where real-time performance is critical, the overhead introduced by the operating system might be unacceptable.

3) Robustness: Robustness assessments indicate similar error rates between bare-metal and an operating system, with slight variations depending on specific applications. The decision between the two hinges on factors like real-time requirements, given the minimal differences in error rates. Notably, when considering an operating system, there is a pronounced increase in the percentage of critical execution flow errors, such as Startup Failures and Crashes. In contrast, bare-metal exhibits a higher susceptibility to SDC errors. Consequently, bare metal may be deemed more appropriate for systems where ensuring high availability is crucial, and the tolerance for erroneous results is acceptable [2].

4) Dependency on Vendor Support: Embedded operating systems often rely on vendor support for updates, bug fixes, and compatibility with new hardware. Depending on the vendor's commitment to the platform, this dependency could impact the long-term viability of the embedded system.

III. Experimentation

Given the paper's objective to include empirical experience as part of the assessment, a miniature project will be conducted utilizing a ESP32S3 and a FreeRTOS implementation as the Embedded OS. The experiment will involve attempting to measure the microcontroller's performance in determining all prime numbers up to the 5500th prime.

For performance evaluation, the Insertion of Instrumentation Code [5] will be utilized, as recommended in prior studies. This will be implemented by measuring the Real-Time Clock (RTC) both before and after the benchmark, allowing for the calculation of the total execution time.

A. Documentation and community importance

One of the most important aspects not previously discussed, is the important of the documentation. The availability of documentation and examples was more abundant for programming with FreeRTOS, facilitating a smoother development process and a better understanding of the built-in functions.

B. Framework importance

While performing the experiment as previously described, unexpected results appeared. Surprisingly, utilizing an embedded operating system leads to a superior performance. After a long of research, a notable distinction was identified: the Bare Metal framework lacked compiler optimization, including generic optimizations. In the other hand such optimizations were available by default on the FreeRTOS counterpart. After disable compiler optimizations the expected results were obtained.

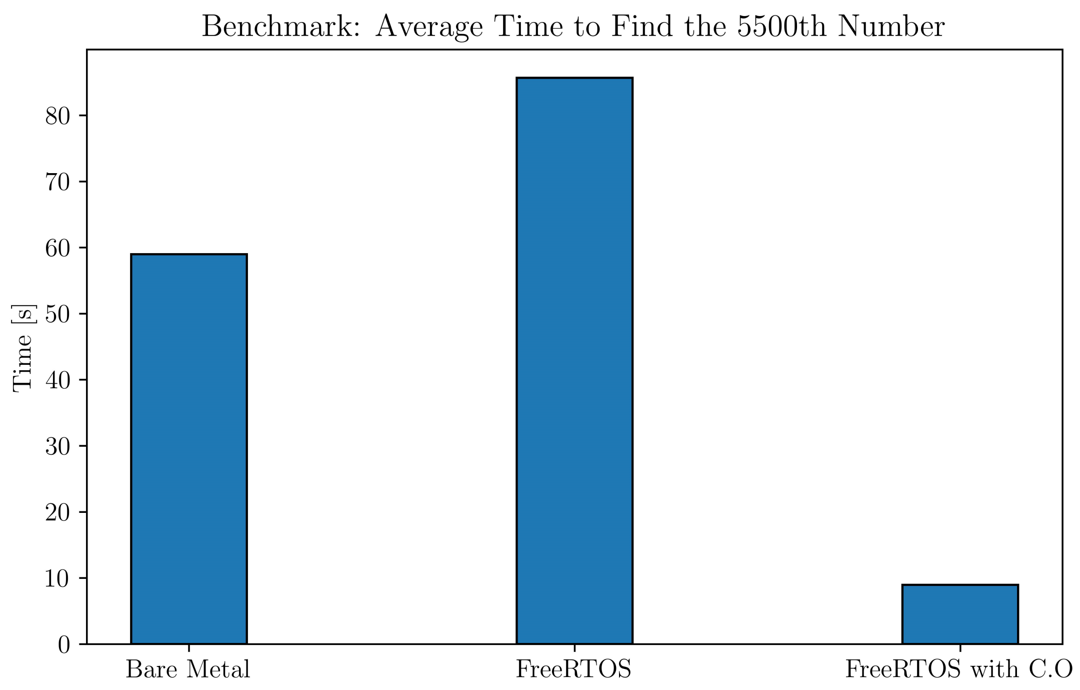

Fig. 1. Experimetation results.

1) Real performance difference: As Fig. 1 illustrates, FreeRTOS without optimizations shows a 1.45 times longer execution time compared to the bare metal counterpart. However, when optimizations were enabled, bare metal displayed a 6.59 longer execution time compared to the FreeRTOS counterpart. This insight underscores the importance of frameworks. Although manual optimization is feasible for bare metal code, an additional challenge arises in this framework: Is the programmer able outperform the compiler optimizations?

This raises a crucial consideration. In scenarios where the programmer cannot surpass compiler optimizations, employing an operating system might lead to superior performance. Thus, when utilizing this framework, the choice between bare metal and an operating system becomes harder, as now the trade-off between manual optimization potential and the benefits of built-in optimizations is something to be consider.

It is important to note that this consideration is specific to this framework. Nevertheless, this example serves as a highlighter of the importance of frameworks in influencing system performance and development. When considering framework options, this aspects requires for careful evaluation and consideration.

IV. Conclusion

Please note that the intention of this paper is not to be in favor of one approach over the other, but rather to empower you with the knowledge and insights necessary to make an informed decision based on your project's unique requirements and constraints. As embedded systems continue to play a crucial role in modern technology, this paper aims to contribute to the ongoing discourse on selecting the most suitable approach for your project.

The choice between bare-metal programming and using an embedded operating system depends on various factors, including project requirements, performance considerations, and development constraints. To chose the right approach you need to acquire comprehensive insights into this subject and exploring through mini-projects on the specific platform to learn the empirical advantages of each implementation.

V. Methodology

In the order to improve the language and the clarity of this paper, ChatGPT has been used. The specific application of ChatGPT was focused on rephrasing to improve readability and coherence. While the core ideas and concepts presented in this paper are based on my original understanding and research.

References

[1] "The embedded rust book - no std." [Online]. Available: https://docs.rust-embedded.org/book/intro/no-std.html

[2] C. D. Sio, S. Azimi, and L. Sterpone, "Evaluating reliability against see of embedded systems: A comparison of rtos and bare-metal approaches," pp. 26–2714, 2023. [Online]. Available: http://creativecommons.org/licenses/by/4.0/

[3] Y. Li, M. Potkonjak, and W. Wolf, "Real-time operating systems for embedded computing," *Proceedings International Conference on Computer Design VLSI in Computers and Processors*, pp. 388–392. [Online]. Available: http://ieeexplore.ieee.org/document/628899/

[4] Q. Teng, H. Wang, and X. Chen, "A hal for component-based embedded operating systems," *Proceedings - International Computer Software and Applications Conference*, vol. 2, pp. 23–24, 2005.

[5] R. Patel and A. Rajawat, "A survey of embedded software profiling methodologies," *International Journal of Embedded Systems and Applications (IJESA)*, vol. 1, 2011.