University of Stuttgart
Institute for Control Engineering
of Machine Tools and Manufacturing Units
(ISW)

Master’s Thesis

Wait-free synchronization for inter-process communication in real-time systems

submitted by

Devrim Baran Demir

from Hamburg

Degree program	M. Sc. Informatik
Examined by	Prof. Andreas Wortmann
Supervised by	Marc Fischer
Submitted on	March 26, 2025

Declaration of Originality

Master's Thesis of Devrim Baran Demir (M. Sc. Informatik)

Address	Kernerweg 22, 89520 Heidenheim an der Brenz
Student number	3310700
English title	*Wait-free synchronization for inter-process communication in real-time systems*
German title	*Wait-free Synchronization für Interprozesskommunikation in Echtzeitsystemen*

I now declare,

- that I wrote this work independently,
- that no sources other than those stated are used and that all statements taken from other works—directly or figuratively—are marked as such,
- that the work submitted was not the subject of any other examination procedure, either in its entirety or in substantial parts,
- that I have not published the work in whole or in part, and
- that my work does not violate any rights of third parties and that I exempt the University against any claims of third parties.

Stuttgart, March 26, 2025

Kurzfassung

Vorhersehbare und korrekte Interprozesskommunikation (IPC) ist für Echtzeitsysteme von entscheidender Bedeutung, da Verzögerungen, Unvorhersehbarkeit oder inkonsistente Datenstände zu Instabilität und Ausfällen führen können. Traditionelle Synchronisationsmechanismen wie Mutex-Verfahren mit Semaphoren, Monitoren oder Sperren verursachen Blockierungen, die zu Prioritätsinversionen und erhöhten Antwortzeiten führen. Um diese Herausforderungen zu bewältigen, bietet die wartefreie Synchronisation eine Alternative, die den Abschluss von Operationen in einer begrenzten Anzahl von Schritten garantiert und so die Systemreaktionsfähigkeit und -zuverlässigkeit sicherstellt.

Diese Arbeit untersucht die Nutzung von wait-free Datenstrukturen für IPC in Echtzeitsystemen mit Fokus auf deren Implementierung in Rust. Das Eigentumsmodell und die strengen Nebenläufigkeitsgarantien von Rust machen es besonders geeignet für die Entwicklung latenzarmer und hochzuverlässiger Synchronisationsmechanismen. Diese Arbeit analysiert bestehende wait-free Methoden für IPC in Echtzeitsystemen und bewertet ihre Leistung im Vergleich zu herkömmlichen Synchronisationsmethoden.

Stichwörter: Echtzeitsysteme, wait-free Sysnchronisation, lock-free Synchronisation, Interprozesskommunikation, rust

Abstract

Predictable and correct Inter-Process Communication (IPC) is essential for real-time systems, where delays, unpredictability or inconsistent data can lead to instability and failures. Traditional synchronization mechanisms, such as mutex (mutual exclusion) mechanisms like semaphores, monitors, or locks (Also called mutex) introduce blocking, leading to priority inversion and increased response times. To overcome these challenges, wait-free synchronization provides an alternative that guarantees operation completion within a bounded number of steps, ensuring system responsiveness and reliability.

This thesis explores the use of wait-free data structures for IPC in real-time systems, focusing on their implementation in Rust. Rust's ownership model and strict concurrency guarantees make it well-suited for developing low-latency and high-reliability synchronization mechanisms. This work examines existing wait-free techniques for real-time IPC, and evaluates their performance against conventional synchronization methods.

Keywords: real-time systems, wait-free synchronization, lock-free synchronization, inter-process communication, rust

Contents

1. Introduction

1.1. Motivation

In modern manufacturing and automation, control systems must operate under strict timing constraints to function reliably. If a system fails to meet these constraints, unexpected delays can disrupt processes, leading to instability or even hazardous failures in safety-critical environments. For this reason, real-time operating systems and low-level programming languages like C, C++, and Rust are widely used to ensure predictable execution times.

Many real-time applications involve multiple tasks that must run concurrently and share resources efficiently. Without proper synchronization, problems such as data corruption or race conditions can occur leading to unpredictable behavior. Traditional synchronization methods, such as mutexes and semaphores, are commonly used to manage access to shared resources. However, these blocking mechanisms introduce significant challenges in real-time settings. Since they require threads to wait for resource availability, they can lead to increased response times, potential deadlocks, and priority inversion. These delays are unacceptable in systems that require strict timing guarantees. [1]–[3]

Consider, for instance, a factory setting where a worker presses an emergency stop button to halt machinery. The system must react instantly, as any delay could lead to severe consequences, including equipment damage or injury. However, if the system experiences a deadlock or a priority inversion, it may become unresponsive at a critical moment. This highlights a fundamental issue with blocking synchronization techniques: while they ensure orderly access to shared resources, they can also introduce delays that are unacceptable in real-time environments.

To overcome these limitations, there is an increasing interest in wait-free and lock-free synchronization techniques. A wait-free algorithm guarantees that every operation completes in a finite number of steps, regardless of contention from other threads. This property ensures system responsiveness and predictability, which are essential for real-time applications. By eliminating blocking and contention-based delays, wait-free synchronization prevents priority inversion and ensures that high-priority tasks execute without interference. [1], [4]

Lock-free synchronization, on the other hand, ensures that at least one thread makes progress in a finite number of steps, but it does not guarantee that every thread will complete its operation. While lock-free algorithms tend to be more efficient than wait-free ones, they allow the possibility of thread starvation in high-contention scenarios. This can be problematic in real-time and heterogeneous computing environments, where some threads might be repeatedly delayed due to interference from faster threads. So we see that lock-free synchronization has a slightly different goal then wait-free synchronization, which is why in this work we will only focus on wait-free datastructures to stay in the scope of this work. [4]

Efficient synchronization mechanisms are particularly important in the context of IPC, which plays a crucial role in real-time systems. IPC allows processes to exchange data efficiently, but its performance is heavily influenced by the synchronization techniques used. Traditional IPC mechanisms, which often rely on locks or shared memory protected by semaphores, can introduce significant latency and reduce throughput. Wait-free data structures offer a promising alternative by ensuring that communication operations complete within predictable time bounds. However, selecting appropriate wait-free data structures and evaluating their performance in real-time environments remains a challenge. [5]–[8]

The Rust programming language provides useful features for implementing real-time synchronization mechanisms. Its ownership model and strict type system prevent data races and enforce safe concurrency without requiring traditional locking mechanisms. Additionally, Rust offers fine-grained control over system resources, making it a strong candidate for real-time applications that demand both low latency and high reliability. [9], [10]

The concepts and methods introduced here, including real-time systems, IPC, synchronization techniques, wait-free synchronization, and the rust programming language are explored in greater depth in chapter 2.

1.2. Objective

The primary goal of this research is to find a wait-free data structure can be used to implement a wait-free synchronization for IPC though a shared memory in real-time systems using Rust. To do so, this study aims to:

- Identify and analyze existing wait-free synchronization techniques for IPC through shared memory for real-time systems.
- Compare the performance of wait-free synchronization mechanisms with traditional locking methods in real-time scenarios.
- Choose and develop the wait-free data structure for IPC through shared memory in a real-time setting using Rust that is best suited for the purpose of this thesis.

By addressing these objectives, this work contributes to the field of real-time systems by providing practical a solution for efficient and predictable IPC with rust. The insights gained from this research can help improve the reliability and performance of real-time applications across various domains.

1.3. Structure of the Thesis

section has to be done

2. Background

To establish a clear foundation for the concepts and definitions introduced throughout this thesis, we provide a fundamental overview of the key topics relevant to this research. This includes an introduction to real-time systems, Inter-Process Communication (IPC), and synchronization techniques, with a particular focus on wait-free and lock-free synchronization. Additionally, we examine the Rust programming language, as it serves as the primary development environment for this study. Furthermore, we explore existing synchronization methods in real-time systems to contextualize the motivation and contributions of this work.

2.1. Real-Time Systems

In real-time systems the correctness of the system does not only depend on the logical results of computations, but also on timing constraints. These systems can be classified into hard or soft and real-time systems. Hard real-time systems have strict timing constraints, and missing a constraint is considered a system failure and may lead to a catastrophic desaster. The sysetem must guarantee that every timing constraint has to be met. An use case would be industrial automation where all the machines and robotic modules have to communicate with each other as quick as possible to ensure no blockage of the manufacturing line. [11]

On the other hand, soft real-time systems try to stick to the timing constraints as much as possible, but missing some timing constraints is not considered a system failure. Infrastructure wise soft real-time systems are similar to hard real-time systems, since it is still considered important to meet these timing constraints. An example would be a multimedia system where it would be considered fine if sometimes frames are dropped to guarantee the video stream. [11]

Sometimes these two systems appear in combination, where some functions have hard real-time constraints and some have soft real-time constraints. Krishna K. gives a good example in his paper where he describes that for the apollo 11 mission some components for the landing processes had soft real-time behavior and the rest still functioned with hard real-time constraints. [11]

Traditionally these processes are managed by schedulers to meet these timing constraints. The scheduler decides which task to execute next and when to execute it. The problem is that with schedulers we would have blocked processes that wait. [11]

Since in this work we focus on wait-free synchronization schedulers are not needed and so will not be discussed further in this work. Also since the focus lies on hard real-time systems, the term real-time system will be used synonymously with the terminology hard real-time systems.

2.2. Inter-Process Communication

Now the processes used in a real-time system also have to share information with each other so the system can function. So some kind of IPC is needed. IPC allows processes to share information with each other using different kind of methods. We will mainly focus on one method explained later. In general IPC is needed in all computing systems, because processes often need to work together (e.g. a producer process passes data to a consumer process). Lets take the brake-by-wire technology as example. Brake-by-wire is a technology for driverless cars where some mechanical and hydraulic components from the braking systems are replaced by wires to transmit braking signals, since there is no driver anymore to press on the braking pedal [12]. This of course requires different processes to share information together. In the context of this thesis this kind of communication requires strict timing constraints as stated as before, since any kind of delay or blockage would lead to fatal consequences. [13], [14]

To achieve IPC different kind of mechanisms are used. The focus for this work lies on shared memory, so that is also the mechanism we will look into.

2.2.1. Shared Memory

To achieve any kind of information sharing between processes, these processes will need to have access to the same data regularly. With a shared memory segment, multiple processes can have access to the same memory location. So all processes which are part of the IPC can read and write to this common memory space avoiding unnecessary data copys. With that processes exchange information by directly manipulating memory. This kind of IPC is particular useful for real-time applications, which handle large volumes of data or are required to quickly transfer data between sensors and control tasks. [14]–[16]

The problem with this is that the system somehow has to manage how the processes access the shared memory. This is mostly done by using different kind of synchronization techniques. Without any synchronization mechanism, data corruption or inconsistent data can occur. [14], [15]

2.3. Synchronization

2.4. Wait-Free Synchronization

2.5. Lock-Free Synchronization

2.6. Rust Programming Language

2.7. State of the Art

3. Related Work

4. Methodology

5. Choosing optimal wait-free data structure

6. Results

7. Implementation

8. Conclusion and Future Work

Bibliography

[1] M. Herlihy, "Wait-free synchronization", *ACM Trans. Program. Lang. Syst.*, vol. 13, no. 1, pp. 124–149, Jan. 1991. DOI: 10.1145/114005.102808. [Online]. Available: https://doi.org/10.1145/114005.102808.

[2] B. B. Brandenburg, *Multiprocessor real-time locking protocols: A systematic review*, 2019. arXiv: 1909.09600 [cs.DC]. [Online]. Available: https://arxiv.org/abs/1909.09600.

[3] O. Kode and T. Oyemade, *Analysis of synchronization mechanisms in operating systems*, Sep. 2024. DOI: 10.48550/arXiv.2409.11271.

[4] A. Kogan and E. Petrank, "A methodology for creating fast wait-free data structures", *SIGPLAN Not.*, vol. 47, no. 8, pp. 141–150, Feb. 2012. DOI: 10.1145/2370036.2145835. [Online]. Available: https://doi.org/10.1145/2370036.2145835.

[5] S. Timnat and E. Petrank, "A practical wait-free simulation for lock-free data structures", *SIGPLAN Not.*, vol. 49, no. 8, pp. 357–368, Feb. 2014. DOI: 10.1145/2692916.2555261. [Online]. Available: https://doi.org/10.1145/2692916.2555261.

[6] M. M. Michael and M. L. Scott, "Simple, fast, and practical non-blocking and blocking concurrent queue algorithms", in *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '96, Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 267–275. DOI: 10.1145/248052.248106. [Online]. Available: https://doi.org/10.1145/248052.248106.

[7] H. Huang, P. Pillai, and K. G. Shin, "Improving Wait-Free algorithms for interprocess communication in embedded Real-Time systems", in *2002 USENIX Annual Technical Conference (USENIX ATC 02)*, Monterey, CA: USENIX Association, Jun. 2002. [Online]. Available: https://www.usenix.org/conference/2002-usenix-annual-technical-conference/improving-wait-free-algorithms-interprocess.

[8] A. Pellegrini and F. Quaglia, *On the relevance of wait-free coordination algorithms in shared-memory hpc:the global virtual time case*, 2020. arXiv: 2004.10033 [cs.DC]. [Online]. Available: https://arxiv.org/abs/2004.10033.

[9] B. Xu, B. Chu, H. Fan, and Y. Feng, "An analysis of the rust programming practice for memory safety assurance", in *Web Information Systems and Applications: 20th International Conference, WISA 2023, Chengdu, China, September 15–17, 2023, Proceedings*, Chengdu, China: Springer-Verlag, 2023, pp. 440–451. DOI: 10.1007/978-981-99-6222-8_37. [Online]. Available: https://doi.org/10.1007/978-981-99-6222-8_37.

[10] A. Sharma, S. Sharma, S. Torres-Arias, and A. Machiry, *Rust for embedded systems: Current state, challenges and open problems (extended report)*, 2024. arXiv: 2311.05063 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2311.05063.

[11] K. Kavi, R. Akl, and A. Hurson, "Real-time systems: An introduction and the state-of-the-art", in Mar. 2009. DOI: 10.1002/9780470050118.ecse344.

[12] X. Hua, J. Zeng, H. Li, J. Huang, M. Luo, X. Feng, H. Xiong, and W. Wu, "A review of automobile brake-by-wire control technology", *Processes*, vol. 11, p. 994, Mar. 2023. DOI: 10.3390/pr11040994.

[13] geeksforgeeks, *Inter process communication (ipc)*, https://www.geeksforgeeks.org/inter-process-communication-ipc/, 2025.

[14] A. Venkataraman and K. K. Jagadeesha, "Evaluation of inter-process communication mechanisms", 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:6899525.

[15] K. C. Wang, "Process management in embedded systems", in *Embedded and Real-Time Operating Systems*. Cham: Springer International Publishing, 2023, pp. 115–168. DOI: 10.1007/978-3-031-28701-5_5. [Online]. Available: https://doi.org/10.1007/978-3-031-28701-5_5.

[16] S. Mogare, A. Mahamune, D. Sathe, H. Bhangare, M. Deshmukh, and A. Ingale, "Message passing vs shared memory-a survey of trade off in ipc", *International Research Journal of Modernization in Engineering Technology and Science (IRJMETS)*, vol. 06, Nov. 2024.

List of Acronyms

IPC Inter-Process Communication

List of Figures

List of Tables

A. Appendix