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 24, 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 24, 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 (RTOS) 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. Despite this limitation, lock-free algorithms are often preferred in practice because they avoid blocking and can achieve high throughput. [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]

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 in real-time systems using Rust. To do so, this study aims to:

- Identify and analyze existing wait-free synchronization techniques for real-time IPC.
- 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 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

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 constraints can be classified into hard, soft, and firm real-time requirements. Hard real-time systems have strict timing constraints, and missing a constraint is considered a system failure and may lead to a catastrophic desaster. An example would be an autopilot system in an airplane or an airplane sensor, where in case of a timing constraint this may lead to a crash. On the other hand, soft real-time systems try to stick to the timing constraints as much as possible, but missing a timing constraint is not considered a system failure. However, this may lead to a degration of the system's Quality of Service (QoS) to improve responsiveness. The QoS describes the overall performance of the networc in this context, which includes factors like latency, jitter (variation between expected and the actual timing for a task, so when a task takes longer or shorter then expected [11]), bandwith, throughput and error rates. An example would be an audio and video streaming service, where some lag is acceptable to some extent. Finally what firm real-time systems do is, treating information that is delivered/computed after a timing constraint as invalid. Examples for these kind of systems are fincancial forecast systems, where the information has no use after a certain time. Since we focus on strict timing constraints, we will focus on hard real-time systems in this thesis. [12]

2.2. Inter-Process Communication

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] ni, *Determinism and jitter in a real-time system*, https://www.ni.com/docs/de-DE/bundle/labview-nxg-rt-module-programming-with-rt-target/page/determinism-real-time.html, 2016.

[12] J. Kay, *Introduction to real-time systems*, https://design.ros2.org/articles/realtime_background.html, 2016.

List of Acronyms

IPC Inter-Process Communication

QoS Quality of Service

List of Figures

List of Tables

A. Appendix