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 17, 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 17, 2025

Kurzfassung

Effiziente und vorhersehbare Interprozesskommunikation (IPC) ist für Echtzeitsysteme von entscheidender Bedeutung, da Verzögerungen und Unvorhersehbarkeit zu Instabilität und Ausfällen führen können. Traditionelle Synchronisationsmechanismen wie Mutexe und Semaphore 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 wartefreien Datenstrukturen für IPC in Echtzeitsystemen mit Fokus auf deren Implementierung und Optimierung in Rust. Das Eigentumsmodell und die strengen Nebenläufigkeitsgarantien von Rust machen es besonders geeignet für die Entwicklung latenzarmer und hochzuverlässiger Synchronisationsmechanismen. Die Studie analysiert bestehende wartefreie Techniken, passt sie für IPC in Echtzeitsystemen an und bewertet ihre Leistung im Vergleich zu herkömmlichen Synchronisationsmethoden. Die Ergebnisse tragen zur Entwicklung effizienter und skalierbarer Synchronisationstechniken bei und verbessern die Vorhersagbarkeit von Echtzeitsystemen.

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

Abstract

Efficient and predictable interprocess communication (IPC) is essential for real-time systems, where delays and unpredictability can lead to instability and failures. Traditional synchronization mechanisms, such as mutexes and semaphores, 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 and optimization in Rust. Rust's ownership model and strict concurrency guarantees make it well-suited for developing low-latency and high-reliability synchronization mechanisms. The study examines existing wait-free techniques, adapts them for real-time IPC, and evaluates their performance against conventional synchronization methods. The findings contribute to the development of efficient and scalable synchronization techniques, improving the predictability of real-time systems.

Keywords: real-time systems, wait-free synchronization, lock-free synchronization, interprocess 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]

For example, consider a factory setting where a worker presses an emergency stop button. If the system experiences a deadlock and becomes unresponsive, the failure could result in severe consequences, including potential injury or loss of life. This highlights the critical need for synchronization methods that do not block execution and allow real-time processes to proceed without unexpected delays.

To address these limitations, there is 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]

Inter-Process Communication (IPC) plays a crucial role in real-time systems, allowing processes to exchange data efficiently. The choice of IPC mechanisms directly impacts system performance, as inefficient communication can introduce latency and reduce throughput. Wait-free data structures offer a promising way to improve IPC 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 explore how wait-free data structures can be used to improve interprocess communication in real-time systems using Rust. Specifically, 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, develop and optimize the optimal wait-free data structures for interprocess communication in Rust.
- Evaluate the effectiveness of these implementations through real-time benchmarks and system simulations.

By addressing these objectives, this work contributes to the field of real-time systems by providing practical solutions for efficient and predictable IPC. 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. State of the Art

To understand what is meant by all the terms and definition in the previous introduction and generally in this thesis, we will first look at how current real-time systems are built and how they work. This will include multiple topics such as real-time systems, inter-process communication, synchronization, wait-free synchronization and lock-free synchronization. We will also look at the Rust programming language since that is the language that is in the scope of this thesis.

2.1. Real-Time Systems

2.2. Inter-Process Communication

2.3. Synchronization

2.4. Wait-Free Synchronization

2.5. Lock-Free Synchronization

2.6. Rust Programming Language

3. Examples

In the following some examples for the conversion in LaTeX are listed. For formal requirements, please have a look at the guidelines for the preparation of student theses at the Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW).

3.1. Listings

An introduction for list environments with LaTeX can be found at `https://en.wikibooks.org/wiki/LaTeX/List_Structures`.

An unordered enumeration can look like this:

- Fusce tincidunt consectetur nisl a pretium. Nam sed eleifend nunc. Nulla feugiat nisl ac mauris varius, eu viverra tellus condimentum. Nullam tempus dolor a elementum con-vallis. Nam sagittis, nisi non tempor luctus, enim ex pretium nunc, lacinia suscipit arcu augue id sem.
- Fusce tincidunt consectetur nisl a pretium. Nam sed eleifend nunc. Nulla feugiat nisl ac mauris varius, eu viverra tellus condimentum. Nullam tempus dolor a elementum con-vallis. Nam sagittis, nisi non tempor luctus, enim ex pretium nunc, lacinia suscipit arcu augue id sem.
- Fusce tincidunt consectetur nisl a pretium. Nam sed eleifend nunc. Nulla feugiat nisl ac mauris varius, eu viverra tellus condimentum. Nullam tempus dolor a elementum con-vallis. Nam sagittis, nisi non tempor luctus, enim ex pretium nunc, lacinia suscipit arcu augue id sem.

Use the `enumerate` environment for ordered lists.

1. Fusce tincidunt consectetur nisl a pretium. Nam sed eleifend nunc. Nulla feugiat nisl ac mauris varius, eu viverra tellus condimentum. Nullam tempus dolor a elementum con-vallis. Nam sagittis, nisi non tempor luctus, enim ex pretium nunc, lacinia suscipit arcu augue id sem.
2. Fusce tincidunt consectetur nisl a pretium. Nam sed eleifend nunc. Nulla feugiat nisl ac mauris varius, eu viverra tellus condimentum. Nullam tempus dolor a elementum con-vallis. Nam sagittis, nisi non tempor luctus, enim ex pretium nunc, lacinia suscipit arcu augue id sem.

3. Fusce tincidunt consectetur nisl a pretium. Nam sed eleifend nunc. Nulla feugiat nisl ac mauris varius, eu viverra tellus condimentum. Nullam tempus dolor a elementum con-vallis. Nam sagittis, nisi non tempor luctus, enim ex pretium nunc, lacinia suscipit arcu augue id sem.

Descriptions are set with the `description` environment.

Mosquito Fusce tincidunt consectetur nisl a pretium. Nam sed eleifend nunc. Nulla feugiat nisl ac mauris varius, eu viverra tellus condimentum. Nullam tempus dolor a elementum con-vallis. Nam sagittis, nisi non tempor luctus, enim ex pretium nunc, lacinia suscipit arcu augue id sem.

Emu Fusce tincidunt consectetur nisl a pretium. Nam sed eleifend nunc. Nulla feugiat nisl ac mauris varius, eu viverra tellus condimentum. Nullam tempus dolor a elementum con-vallis. Nam sagittis, nisi non tempor luctus, enim ex pretium nunc, lacinia suscipit arcu augue id sem.

Armadillo Fusce tincidunt consectetur nisl a pretium. Nam sed eleifend nunc. Nulla feugiat nisl ac mauris varius, eu viverra tellus condimentum. Nullam tempus dolor a elementum con-vallis. Nam sagittis, nisi non tempor luctus, enim ex pretium nunc, lacinia suscipit arcu augue id sem.

3.2. Cite

Cite with the `\cite{}` command **Cochran2005**, **Cubitt2013**. You can mention authors and titles as well with `\citeauthor{}` and `\citetitle{}`: E.g., **Feuersaenger2014** developed `pgfplots` and describes it in the documentation **Feuersaenger2014**.

Use tools for literature management as JabRef or Citavi.

3.3. Typing math

An introduction for typing math with LaTeX can be found at `https://de.wikibooks.org/wiki/LaTeX-Kompendium:_F%C3%BCr_Mathematiker`. The Wikipedia page for typesetting math is worth a visit as well (`https://de.wikipedia.org/wiki/Formelsatz`).

You can use `\( ...\)` to typeset fomulas in text e.g., $\sqrt{a^2}$) or $\begin{bmatrix} a & b & c \end{bmatrix}^{\mathrm{T}}$ (`\bmat` and `\T` are self-defined macros from `settings.tex`).

Multiline math environments can e.g. be set with the `align` environment. A & character helps with the vertical alignment.

$$\begin{aligned} c^2 &= a^2 + b^2 \\ \Leftrightarrow \quad c &= \pm\sqrt{a^2+b^2} \qquad (3.1) \end{aligned}$$

Only number equations that will be referenced later, such as Equation 3.1. `\nonumber` suppresses the generation of an equation number.

3.4. Tables

Tasbles are set in a `table` environment, as shown in Table 3.1. They must always be referenced and explained in the previous text. The package `booktabs` faciliates the consistent use of horizontal line strenghts.

Article		
Animal	Description	Price (€)
Mosquito	per gramm	13.65
	per piece	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

Table 3.1.: Example table using the booktabs package

3.5. Graphics

Simple graphics can be integrated with `\includegraphics`. Always use a `figure` environment for graphics. Reference graphics before they appear in the text, such as Figure 3.1, and assign meaningful captions to them.

Universität Stuttgart
Institut für Steuerungstechnik
der Werkzeugmaschinen und
Fertigungseinrichtungen (ISW)

Figure 3.1.: The logo of the ISW at the University of Stuttgart

Some graphics with TikZ and PGFplots are shown on Figure 3.2, Figure 3.3a, Figure 3.3b and Figure 3.4 which might be helpful or inspirational for your thesis.

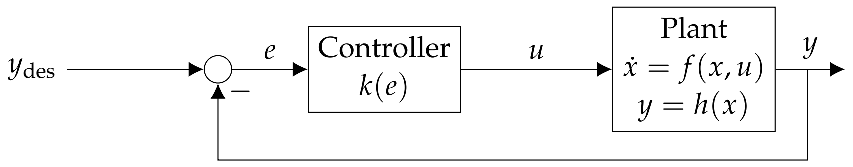
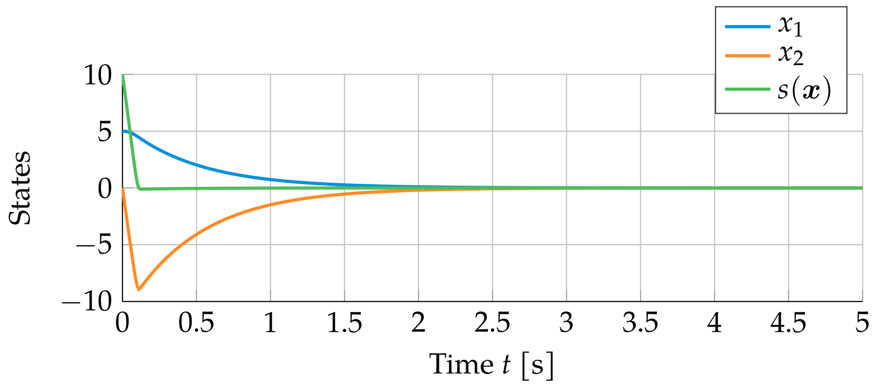
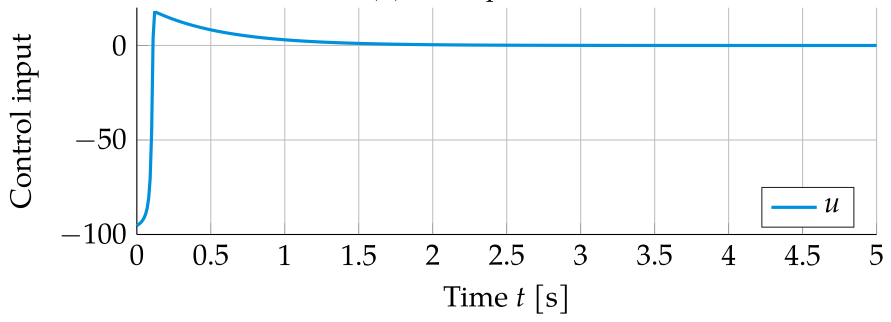


Figure 3.2.: A simple block diagram



(a) States plotted over time



(b) Input $u(\cdot)$ plotted over time

Figure 3.3.: Example for exports from MATLAB with `matlab2tikz`

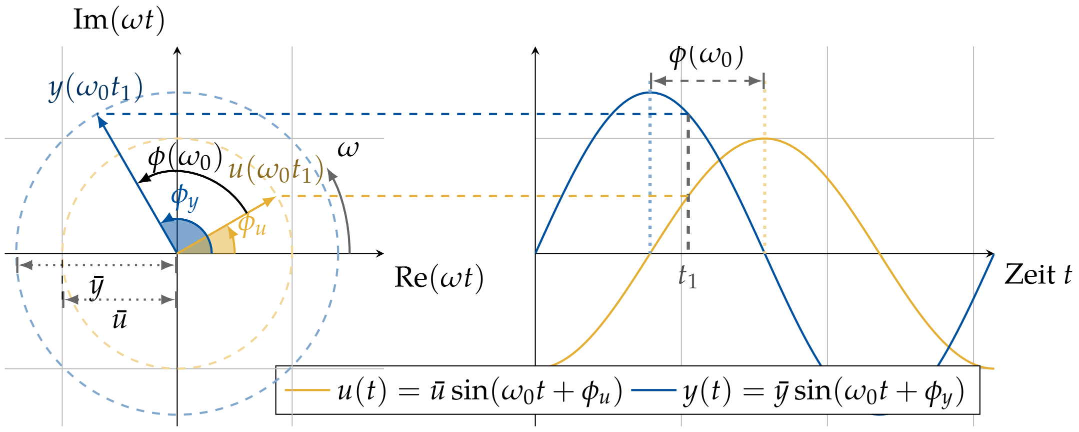


Figure 3.4.: Analytical calculation of a phasor diagram with TikZ **Tantau2013**

3.6. Code

The `listings` package, for example, is suitable for code excerpts. Make sure that you only include necessary code, Listing 3.1 is a bad example and should not be included.

```
#include <iostream>

using namespace std;

int main(void){
    cout << "Hello world." << endl;
    return 0;
}
```

Listing 3.1: This code does not provide any insights and should not be included.

3.7. Abbreviations

For abbreviations you can use the package `acronym`.

The command `\ac{}` introduces the abbreviation at the first use, for example ISW and several Speicherprogrammierbare Steuerungen (SPS). Then the abbreviations ISW and SPS will be used automatically. The definition of the abbreviations can be done in a separate file, the example document includes `chapters/Acronyms`.

We recommend resetting the acronym package after the abstract with `\acreset`, so they will be reintroduced in the introduction chapter.

4. Tooling

4.1. Recommended Editors

There are several options for working on your LaTeX document. All of them have individual strong suits and drawbacks. For locally installed editors it is highly recommended to use version control (see section 4.3).

4.1.1. Overleaf

By far the easiest and fastest setup is provided by `overleaf.com`. After creating an account, you can search for the ISW thesis template and start working on your thesis right away. Overleaf allows simultaneous editing of by multiple persons and provides integrated version control and document compilation. However, sometimes it is not possible to have your work hosted on public servers due to Non-Disclosure Agreements (NDAs). Ask your supervisor if you're allowed to use overleaf!

4.1.2. Visual Studio Code

VSCode is also a valid choice for editing latex documents. Just install the `LaTex Workshop` extension. You should also consider installing the `LTex` extension for integrating LanguageTool (see section 4.2).

4.1.3. TexStudio

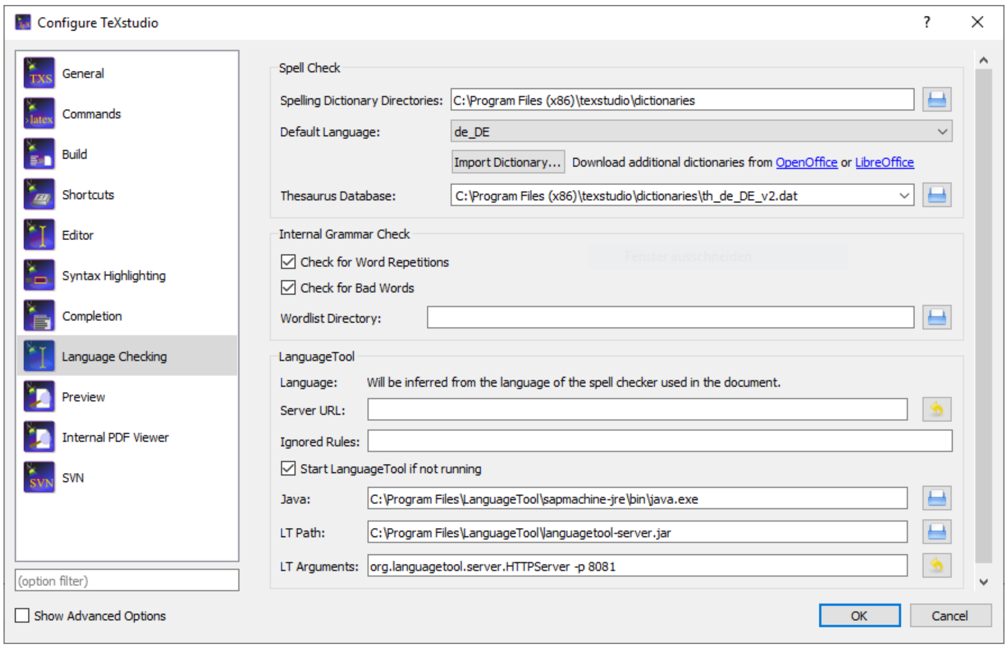
All ISW pool computers come with the TeXstudio[1] latex editor and the MiKTeX[2] LaTeX distribution pre-installed. If they are missing from your ISW-machine, you can install them using the OPSI software-on-demand utility. Remember to select both for installation.

In TeXstudio, remember to set `lualatex` as the standard compiler and `biber` as the bibtex backend.

4.2. Checking of grammar, spell and style

As with all word-processing tasks, it is highly recommended to use at least some sort of spell-checking software. For this purpose TexStudio provides integration with Lan-

[1] `https://www.texstudio.org/`
[2] `https://miktex.org/`



Spell Check

Spelling Dictionary Directories: C:\Program Files (x86)\texstudio\dictionaries

Default Language: de_DE

Import Dictionary... Download additional dictionaries from OpenOffice or LibreOffice

Thesaurus Database: C:\Program Files (x86)\texstudio\dictionaries\th_de_DE_v2.dat

Internal Grammar Check

[✓] Check for Word Repetitions

[✓] Check for Bad Words

Wordlist Directory:

LanguageTool

Language: Will be inferred from the language of the spell checker used in the document.

Server URL:

Ignored Rules:

[✓] Start LanguageTool if not running

Java: C:\Program Files\LanguageTool\sapmachine-jre\bin\java.exe

LT Path: C:\Program Files\LanguageTool\languagetool-server.jar

LT Arguments: org.languagetool.server.HTTPServer -p 8081

Figure 4.1.

guageTool[3]. LanguageTool is a java based application, so make sure to install a Java Runtime Environment. On ISW pool-computers you can again install LanguageTool using OPSI software-on-demand. The necessary settings for ISW pool-computers are shown in Figure 4.1. Further information on the integration of LanguageTool with TexStudio can be found at the LanguageTool Wiki [4].

4.3. Version control

Whenever working on a document, it is desirable to have some sort of version control. For this task, ISW provides a gitlab server found at `https://git.isw.uni-stuttgart.de/`. Once you are logged in to gitlab, you can create your own repository for tracking your thesis files. A `.gitignore` file is necessary to not track all changes to automatically generated files. You may use the `.gitignore` provided by this template. Again, if `git` is not installed on your ISW-machine, you can install it using OPSI software-on-demand.

You can achieve a rudimentary integration with TeXstudio by defining macros such as the git commit macro shown in Listing 4.1. Note that macros can also be called by using shortcuts.

```
%SCRIPT
dialog = new UniversalInputDialog()
```

[3]`https://languagetool.org`
[4]`http://wiki.languagetool.org/checking-la-tex-with-languagetool#toc2`

```
dialog.setWindowTitle("Git commit")
dialog.add("", "Message", "comment")
dialog.add(false, "Commit all files","allfiles")
if (dialog.exec() != null) {
comment = dialog.get("comment")
if ((dialog.get("allfiles")) == true){
buildManager.runCommand(
"git commit -a -m \"" + comment + "\"", editor.fileName())
}else{
buildManager.runCommand(
"git commit " + editor.fileName() + " -m \"" + comment +
"\"", editor.fileName())
}
}
```

Listing 4.1: `git commit` macro

5. Conclusions

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] M. F. Dolz, D. del Rio Astorga, J. Fernández, J. D. García, F. García-Carballeira, M. Danelutto, and M. Torquati, "Embedding semantics of the single-producer/single-consumer lock-free queue into a race detection tool", in *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores*, ser. PMAM'16, Barcelona, Spain: Association for Computing Machinery, 2016, pp. 20–29. DOI: 10.1145/2883404.2883406. [Online]. Available: https://doi.org/10.1145/2883404.2883406.

[12] M. Torquati, *Single-producer/single-consumer queues on shared cache multi-core systems*, 2010. arXiv: 1012.1824 [cs.DS]. [Online]. Available: https://arxiv.org/abs/1012.1824.

[13] A. Gidenstam, H. Sundell, and P. Tsigas, "Cache-aware lock-free queues for multiple producers/consumers and weak memory consistency", in *Proceedings of the 14th International Conference on Principles of Distributed Systems*, ser. OPODIS'10, Tozeur, Tunisia: Springer-Verlag, 2010, pp. 302–317.

[14] B. B. Brandenburg and J. H. Anderson, "Spin-based reader-writer synchronization for multiprocessor real-time systems", *Real-Time Syst.*, vol. 46, no. 1, pp. 25–87, Sep. 2010. DOI: 10.1007/s11241-010-9097-2. [Online]. Available: https://doi.org/10.1007/s11241-010-9097-2.

List of Acronyms

ISW	Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart
SPS	Speicherprogrammierbare Steuerung
NDA	Non-Disclosure Agreement
IPC	Inter-Process Communication

List of Figures

List of Tables

List of Symbols

This section is optional. Ask your supervisor whether it is required for your thesis. If you have more than 10 formulas involved it probably is.

There are two ways to build a list of symbols:

- If you just want to get it done, then use a `longtable` and fill your symbols in, see table below.
- If you want it fancy, then package `glossaries` (maybe `glossaries-extra`) may be your way to go. Be warned that although it automates symbol handling (e.g. sorting and referencing of symbols), it comes with some administrative overhead. You can find a discussion on different ways to achieve this on https://tex.stackexchange.com/a/366282.

Symbol	Unit	Description
ψ	rad	Heading angle of hamster
$\dot{x}$	m/s	Linear velocity of hamster
$\ddot{x}_0$	m/s^2	Initial acceleration of hamster

A. Example appendix chapter