

Interim Report

Project Title: An Analysis Suite for Rust's Advantages in Linux
Name: Fahad Khan
Student Id: 20454859
Email Address: efyfk3@nottingham.ac.uk
Programme: BSc Computer Science
Module: COMP3003

Contents

1	Introduction	2
1.1	Context	2
1.2	Related Work	3
1.2.1	Research Gap	3
1.3	Aims and Objectives	3
2	Technical Background	4
2.1	C Family Memory Management	4
2.2	Alternative Memory Strategies	4
2.2.1	Garbage Collectors	4
2.3	Rust Memory Management	4
3	Project Description	4
3.1	Benchmarking Approach	4
3.1.1	Optimisation Tools	4
4	Project Description	4
4.1	Design	4
4.2	Methodology	4
5	Progress	4
6	Bibliography	5

1 Introduction

very quickly get out of the way the point and what will be achieved.

that paragraph on exactly how optimisation is difficult. how something on many machines, optimisation needs to be perfect. how developing memory-safe easily can really help. (write this out, argue that ease of memory writing is worth considering in overall benchmark)

1.1 Context

copied over from proposal, but be careful to be clearer about the program

Linux is an operating system that runs on billions of computers worldwide, with its underlying kernel worked on by thousands of individuals. Unlike its prominent market competitors, the ongoing development effort is open-source, allowing any individual to view and contribute to the source code. Adopters continue to further advantages such as hardware versatility, security and all-encompassing software utility [1]. For example, the kernel modularises processor-specific code, allowing it to be compiled and ported to any processor. This allows Linux to be run across a great variety of network architectures and embedded systems, which take advantage of its powerful system capability [2]. Linux's growing capital and market share have increased interest and contribution from large corporations, such as Google and Intel, becoming dependants [3]. Given the scale of development and sheer prevalence of Linux distributions worldwide, questions are naturally raised about the safety and maintainability of the kernel.

As demonstrated by the July 2024 CrowdStrike incident, where a small update with a memory issue to kernel-level software caused global Windows outages, without safety guarantees a pivotal error may result in catastrophic propagation across an operating system. Therefore, it may be important to re-evaluate the foundation on which Linux is built. The Linux kernel is written almost entirely in C, a programming language pervasive in operating system programming due to providing developers with low-level memory control and minimal runtime overhead. Optimisation is difficult yet incredibly rewarding, with precise manipulation of memory addresses enabling efficient data structure operations [4]. However, C's lack of modern safety features risks developer errors in memory management such as buffer overflows or dangling pointers, compromising system security and stability. This creates incentive to explore safer alternatives for the kernel while maintaining efficiency.

Rust now becomes the main contender for Linux development, a modern systems programming language designed to provide memory safety guarantees without sacrificing the low-level performance essential for kernel development [5]. Due to the continuing frustration with memory and concurrency issues within Linux, the value of preventing the various harms endemic to C grew into the "Rust for Linux" movement, culminating in the official addition of Rust kernel modules to the codebase in 2022 [6]. This ongoing effort has raised significant contention over the second language's practical efficiency and the prospective maintenance burden, forming the central focus of the project.

To carefully address these concerns, this project will produce a software suite to assess Rust's practical advantages within the Linux kernel. An automated framework will benchmark performance by executing a comparative suite that tests Rust modules provided by the user against other languages and variables. Extensive related work has focused on quantitative performance benchmarks between Rust and C, with findings largely indicating that Rust introduces performance overhead and does not eliminate all bug classes [7], [8]. There may be a research gap in these assessments of Rust's advantages, as purely quantifying overhead is insufficient to capture the full trade-off, and the project will address this by including evaluation of high-level abstractions and built-in safety guarantees.

1.2 Related Work

-> focus on this, need to consolidate research here

1.2.1 Research Gap

1.3 Aims and Objectives

can be the same, but more indepth for the project (individual aspects more clear)

2 Technical Background

-> quick overview on memory addresses and registers then c, rust, c++ here. garbage collectors and performance

2.1 C Family Memory Management

2.2 Alternative Memory Strategies

2.2.1 Garbage Collectors

2.3 Rust Memory Management

3 Project Description

3.1 Benchmarking Approach

3.1.1 Optimisation Tools

4 Project Description

4.1 Design

4.2 Methodology

5 Progress

6 Bibliography

- [1] E. Siever, *Linux in a Nutshell*. O'Reilly, 2005.
- [2] D. Abbott, *Linux for embedded and real-time applications*. Newnes, an imprint of Elsevier, 2018.
- [3] J. West and J. Dedrick, “Open source standardization: The rise of linux in the network era,” *Knowledge, Technology & Policy*, vol. 14, pp. 88–112, 2001, doi: 10.1007/pl00022278.
- [4] L. O. Andersen, “Program analysis and specialization for the C programming language,” 1994.
- [5] N. D. Matsakis and F. S. Klock, “The rust language,” Association for Computing Machinery, Portland, Oregon, USA, 2014. doi: 10.1145/2663171.2663188.
- [6] H. Li, L. Guo, Y. Yang, S. Wang, and M. Xu, “An Empirical Study of Rust-for-Linux: The Success, Dissatisfaction, and Compromise,” in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, Santa Clara, CA: USENIX Association, Jul. 2024, pp. 425–443. [Online]. Available: <https://www.usenix.org/conference/atc24/presentation/li-hongyu>
- [7] S.-F. Chen and Y.-S. Wu, “Linux Kernel Module Development with Rust,” 2022, doi: 10.1109/dsc54232.2022.9888822.
- [8] F. Garber, “Rust in the Linux Kernel: Analyzing Rust Implementations of Device Drivers,” *Tuwien.at*, 2025, doi: 10.347s.2025.127963.