Project Proposal

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

1. Background and Motivation

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.

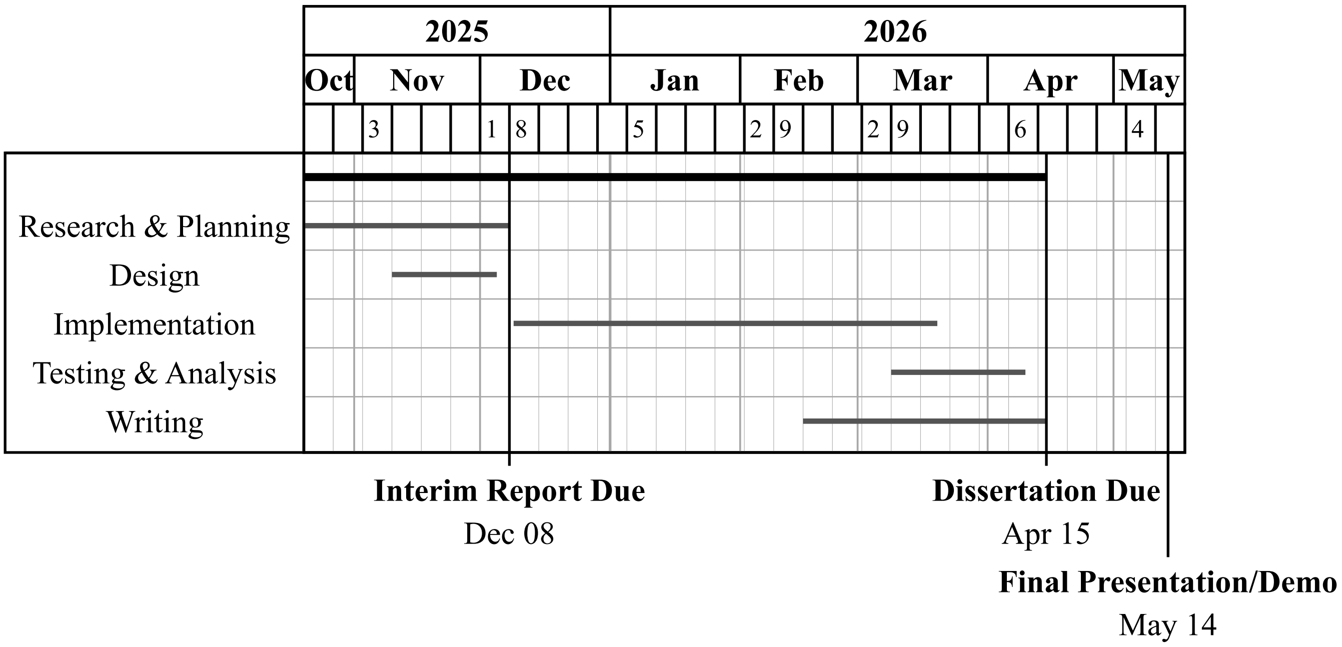
2. Aims and Objectives

The aim of this project is to develop an analysis software that assesses the performance of Rust against alternative languages in Linux kernel modules. Crucially, results evaluate and identify key maintainability critera, such as memory safety and abstraction.

- The key objectives for this project are to:
  1. Research and conduct a thorough evaluation of existing methods for performance benchmarking and code analysis in systems-level programming, across various language types.
  2. Develop a user-friendly software suite, such as a command-line interface, that automates the compilation and execution of modules written in Rust and alternatives.
  3. Implement a robust analysis engine within the suite that uses empirical data on kernel maintances indicators, in order to facilitate an evaluation of key maintainability criteria.
  4. Automatically process and present the collected data in a clear, comparative report in order to convey the performance and safety trade-offs between the languages.

3. Work Plan

The project will have clear phases for research, design, implementation and testing to ensure consistent progress. The timeline allocates sufficient time for each key task while accounting for other academic commitments. Regular meetings with the project supervisor will be held to ensure the project's progress aligns with requirements and that any changes can be implemented early.



4. 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.