

**Developing Device Drivers in Rust**

Kyle Christie

B00415210

School of Computing, Engineering

and Physical Sciences

BSc (Honours) Computing Science

University of the West of Scotland

Supervisor: Paul Keir

Moderator: Stephen Devine

# 1. Background

## Device Drivers

Device drivers are a vital component of Operating Systems which allow for the control of peripheral devices while interacting with underlying hardware. Drivers also provide facilities which can be used to extend an Operating System via file systems, network protocol, anti-virus capability and more (Ball et al, 2006). Described as the "software layer that lies between applications and physical devices" (Corbet et al, 2005), drivers are clearly a necessity within an Operating System however they suffer from a range of issues with dangerous consequences.

Drivers continue to be programmed with the C programming language which was first developed at Bell Labs between 1969 and 1973, alongside early development of Unix (Ritchie, M.D, 1993). It was designed as a "system implementation language for the nascent Unix operating system" (Ritchie, M. D, 1993). Languages such as C, C++ and Assembly have the potential to be memory unsafe (Gaynor, 2019) which can then lead to critical vulnerabilities as observed by several organisations over the years (Thomas and Gaynor, 2019).

Memory safety is an attribute found within various programming languages with the aim of preventing the developer from introducing certain bugs which strongly relate to memory management (Prossimo, 2022). Memory safety issues usually lead to security problems with common vulnerabilities being out-of-bounds reads, out-of-bounds writes and use-after-frees (Gaynor, 2019).

Furthermore, Linux drivers have seen little to no change within the last two decades. Evidence pointing to this can be found in Linux Device Drivers 3, a book written for Linux Kernel 2.6 (Corbet et al, 2005), where its code examples can compile and successfully run on more recent kernel versions with little to no change. Further evidence supports this point as even online tutorials from 2014 (Karthik M, 2014) continue to compile and run on recent kernel versions. Such examples were built and executed on a small collection of Linux distributions that utilise more recent kernel versions, specifically 4.19.0-17-amd64, 5.15 .0-52-generic and 5.15.67-v7l+. It is therefore clear that device drivers have not evolved in any significant way. Code which targets Linux kernel version from over a decade ago continues to run on more recent versions.

## Input from Industry

During the project, prominent or relevant figures within Game Development and Software Engineering were contacted. These include Jonathan Blow (developer of Braid, The Witness and the Jai programming language) Dave Plummer (former Microsoft engineer, now entrepreneur, best known for his work in creating Windows Task Manager among other projects), Alex Gaynor (Software Resilience Engineer working on the Rust-For-Linux project), Miguel Ojeda (Software Engineer working on the Rust-For-Linux project) and Asahi Lina (Software Engineer working on a Linux GPU driver in Rust for the Apple M1).

Blow fed back that drivers are a good problem to consider though felt that the project was not ambitious enough. He spoke of a concept called an 'Exo-kernel model' which he feels would fix a lot of device driver problems as drivers are simply considered 'normal' code. He also feels that the field is held back as the basic idea of what an operating system should be has already been figured out, Blow seems to think there is little innovation in the field. Alongside his feedback, he also provided me with a USENIX ATC Keynote; 'It's Time for Operating Systems to Rediscover Hardware'.

Plummer explained that he had mostly worked on cache software and filesystems but had never really worked on drivers unless it was necessary. He recommended a textbook, 'Windows 7 Device Driver' by Ronald D. Reeves Ph.D. and gave his best wishes.

Gaynor praised the project, saying "Building a USB mouse driver with Rust for Linux sounds like a great project", and outlined the approach used by the RustForLinux community when writing Drivers which is as follows;

1. Check if RustForLinux already has existing APIs for the relevant kernel subsystem.
2. If not, they design a safe Rust API that exposes the original kernel APIs.
3. It is then possible to use these new abstractiosn to write the original driver.

Gaynor mentioned a pull request for the Rust-For-Linux repository which aims to add support for USB device drivers. After a short correspondence, Ojeda provided some advice and useful resources. He recommended getting used to reading C code within kernel. With regards to the Rust driver, his recommendation was to write a C version which can then be referenced for the Rust driver, this will also help in learning kernel APIs and verify whether issues are a result of Rust support or otherwise.

## Project goal

The aim of this project is to try and overcome the previously highlighted issues by developing a Linux device driver in Rust. Not only will it replace C, Rust and its features should prevent issues with memory safety. Rust is a relatively young language with several benefits and features that aim to improve memory safety. It continues to spread through industry as it was recently incorporated into the Linux Kernel from version 6.1 (Vaughan-Nichols, 2022) and there have been public calls from developers for Rust to be utilised more. An example of this being Microsoft Azure CTO, Mark Russinovich, urging the industry (regarding to C and C++) 'For the sake of security and reliability, the industry should declare those languages as deprecated.' (Claburn, 2022).