A Rusty Research Kernel: CS4999 Independent Study Proposal

Jackie LIU[1]

[1]Cornell University
jl2627@cornell.edu

Abstract – A research kernel targeting RISC architecture written purely in the Rust programming language, without a C runtime, answering the hypothesis that Rust is an excellent language for building OS's on different platforms, old and new.

1 Background and Motivation

C and C++ have a long and storied history in systems development. However, issues such as memory unsafety plague their otherwise successful use. Rust minimally sacrifices performance and uses a borrowing system which guarantees memory safety, making it a viable alternative. In the same vein, while CISC architectures have been dominant in microprocessor design, RISC architectures take different design decisions for today's low-power, high memory devices. New technologies such as Rust and RISC-V seek to remedy many of the issues that have arisen from the previous paradigms of systems development, and new software should be developed with these new paradigms in mind to reflect evolving technology and engineering practices.

2 Relevance and related works

Creating a research Rust kernel targeting RISC architectures will allow us to explore the frontier and feasibility of doing so, helping with future implementations. There is a body of work that has been done around this design area, of which the Mimiker MIPS microkernel and Redox, Theseus, intermezzOS, and orange_slice Rust kernels/OSes I will draw inspiration from. There has also been some illustrative rhetorical debates surrounding the Rust language in particular for rewriting systems software[3], demonstrating its suitability through a variety of desirable language features[1].

3 Methods and Approach

I will focus on two main challenges in different stages to this project. First, in delivering a runnable kernel as close to the metal as possible (not emulating OS/hardware features inside an existing OS), and second in implementing some novel OS primitives on top. I will attempt to build on technical characteristics such as safety and performance, in addition to more novel OS concepts that I will explore through the aforementioned implemented Rust OS's.

For example, I could explore and build on the notion of OS modularity and static invariant verification in the Theseus OS[2]. The key hypothesis of the Theseus project was whether a modular and "compiler verified" OS would increase its fault recovery through reducing the notion of state spill, or fate-sharing between software modules.

I will refer to course material for CS4410 and CS3410 for theoretical knowledge, and Rust documentation and online tutorials for building hobby OS's for the basic implementation. The code will be entirely my own and I must cite non-trivial code written by others otherwise.

4 Timeline

1. In one or two semesters: A very basic RISC Rust kernel running on QEMU, Raspberry Pi, etc to build a technical foundation
2. Beyond: Pull requests in the Theseus or Redox repos or documented novel features in the basic kernel

5 Deliverables

1. A GitHub repository with runnable code targetting RISC architecture that the bootloader will select to run
2. Monthly progress reports (alpha, beta, etc.)
3. Lengthier technical reports upon major milestones documenting what was implemented
4. A written demo and short live demo

6 Evaluation

Similar to a project based course scaled to the appropriate number of credits.

References

[1] A case for oxidation: The rust programming language, Jun 2018.

[2] Kevin Boos, Namitha Liyanage, Ramla Ijaz, and Lin Zhong. Theseus: an experiment in operating system structure and state management. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1–19. USENIX Association, Nov. 2020.

[3] MarakanaTechTV. Is it time to rewrite the operating system in rust?, Feb 2019.