

x-realism 调研报告

项目简介

使用 Rust 编程语言实现一个操作系统内核

项目背景

Rust 调研

Rust VS C/C++

C/C++ 诞生在硬件极为昂贵的时代，所以追求性能，其过于灵活，最大的问题就是安全性问题，很容易出现漏洞，包括但不限于以下

- **释放后使用/双重释放错误**：由于需要手动管理内存，导致需要在 `free` 时小心翼翼
- **悬空指针**：可能导致空指针引用等不安全行为
- **缓冲区溢出错误**：这是造成大量漏洞和导致被攻击的原因
- **数据竞争**：数据的不安全并发修改
- **未初始化的变量**：会引发一系列未定义行为

在编写、调试程序时通常需要花费大量的时间来解决内存或数据竞争问题，而人肉 code review 大大降低了效率，也给后续的维护造成了极大的挑战，而下文会提及 Rust 是如何实现安全的。

Rust VS GC'ed languages

随着硬件成本的降低，Java 等语言用性能(GC)来换安全性，但是 GC 的劣势也很明显。

- **代价昂贵**：无论是何种类型的 GC，其维护代价都不低。
- **内存开销**：运行时需要动态回收，降低性能
- **非确定性**：不知道何时会暂停进行回收，取决于所用内存
- **难以优化**：无法自行优化缓存，因为 GC 不知道程序将如何使用内存，其优化方式未必最优。

In our production environments, we have seen unexplainable large STW pauses (> 5 seconds) in our mission-critical Java applications.

Rust 优势

Rust 是一门强调**安全**、**并发**、**高效**的系统编程语言。无 GC，实现内存安全机制、无数据竞争的并发机制、无运行时开销的抽象机制，它声称解决了传统 C 语言和 C++ 语言几十年来饱受诟病的内存安全问题，同时还保持了很高的运行效率、很深的底层控制、很广的应用范围，在系统编程领域具有强劲的竞争力和广阔的应用前景。

高效

Rust 无 GC，无 VM，无解释器，具有极小的运行时开销，能充分高效利用 CPU 和内存等系统资源。

It is an explicit goal of Rust to be as fast as C++ for most things. Given that Rust is built on top of LLVM, any performance improvements in it also help Rust become faster.

以下为几门语言的性能对比

Language	User	System	Total	Slower than (C++)	Language version	Source code
C++ (<i>optimized with -O2</i>)	0.899	0.053	0.951	–	g++ 6.1.1	link
Rust	0.898	0.129	1.026	7%	1.12.0	link
Java 8 (<i>non-std lib</i>)	1.090	0.006	1.096	15%	1.8.0_102	link
Go	2.622	0.083	2.705	184%	1.7.1	link
C++ (<i>not optimized</i>)	2.921	0.054	2.975	212%	g++ 6.1.1	link
Python 3.5	17.950	0.126	18.077	1800%	3.5.2	link
Python 2.7	25.219	0.114	25.333	2562%	2.7.12	link

安全

Rust 设计上是内存安全的。它不允许**空指针**、**悬空指针**或**数据竞争**。其丰富的**类型系统**和**所有权模型**保证了内存安全和线程安全，使得能够在编译时消除许多类别的错误。也就是说，一段能跑起来的代码大概率是安全的。具体特性如下

- **内存管理**：采用 **RAII**(resource acquisition is initialization) 模式；有引用(`&`)，此类指针不涉及运行时引用计数。安全性在编译时进行验证，防止未定义行为。
- **所有权模型**：变量与存放在内存某块的值绑定
 - 每个值只能绑定到一个变量，此时该变量拥有值的**所有权**；
 - 变量离开作用域，它负责回收位置和销毁值。
- **智能指针**：通过智能指针 `Box<T>` 来控制存放在**堆**内存中的类型为 `T` 的值；Rust 的智能指针功能丰富，许多开箱即用
- **生命周期**：通过生命周期注释，保证生命周期一致，杜绝**悬空指针**。
- **借用规则**：可以借用变量控制的值得到**不可变引用**(`&T`)和**可变引用**(`&mut T`)
 - 在一个引用存在的全程，被它借用的值不能销毁
 - 一个值同时只能有一个可变引用，但可以有多个不可变引用
- **移动语义**：内置的静态分析器不允许移动后使用，由 borrow checker 进行检查。
- **类型安全**：对一些基本类型的行为进行了限制，较少甚至消除**语义不明确**行为。
- **并发编程模型**：简单来说就是编译器阻止了一切可能的**数据竞争**。
- **错误处理**：使用 `Option<T>` 解决空指针问题；针对可恢复和不可恢复错误有不同处理。

生产力

Rust 有内容详尽的**文档**以及开放、友好、高效的**开源社区**。并且有一流的开发**工具链**。

- 集成的包管理工具 cargo。
- 编译器能提供有效的错误提示和修正信息，减少了 debug 的时间。
- 自动格式化程序 clippy 规定了代码格式，减少了团队磨合统一标准的时间。
- 支持单元测试，不用引入测试框架。

本项目 Rust 适配性

本项目的目标是写一个内核，属于系统编程范畴，可能有如下相关需求

- 对硬件的控制
- 对系统底层的控制
- 对 CPU 和内存的高效利用
- 对运算性能的要求
- 对系统安全和内存安全的要求

综合前面的调研结论，首先考虑到当前主流的系统编程语言 C/C++ 在安全性方面的问题，而 Rust 又可看做传统 C/C++ + 内存安全，符合系统编程相关要求；其次，相较于传统 C/C++，Rust 算是一门年轻的语言，考虑到其潜力和和的发展趋势，以及项目的未来维护，最终敲定 Rust 作为本项目的开发语言。

项目细节

我们计划实现如下特性中的部分

- 微内核
 - 共享内存
- 硬件中断
- 段页式的内存管理
- 多任务并发
 - 实时操作系统
 - 协程 coroutine

微内核分析

宏内核对比



知乎 @5A59

(宏内核，使用函数调用在各个模块间通信)



微内核安全性好、可移植性好、灵活性高、容易 debug，并且抽象性也更好。但同时也存在着性能会相应降低的问题

微内核可以作为主要参考，因为可以快速做出原型，也方便功能迭代，参考 <https://github.com/redox-os/redox> 中的内核部分，或者 Minix 项目

可以灵活拓展更多功能：

- 网络部分 TCP/IP 协议栈
- 高效文件系统支持
- 图形渲染

展望

- 微内核存在不同进程之间通讯频繁切换上下文造成的性能损失
 - 采取共享内存的策略，牺牲进程内存的安全性换取效率，RT-Thread Smart

中断处理

CPU 异常发生在各种错误情况下，例如访问无效内存地址或被零除时。为了对它们做出反应，我们必须建立一个中断描述符表来提供处理函数。这有助于进一步增强我们所写的操作系统内核的安全性和稳定性

内存管理

分页是一种非常常见的内存管理方案，我们也可以考虑在操作系统中使用它。它处理了内存隔离，内存碎片问题，引入了虚拟内存的机制

并发相关

考虑向内核添加对 `async/await` 的基本支持，例如实现一个异步键盘任务，基本执行器

其他项目参考

- [Rust for Linux](#) 一个 GitHub 组织, 得到了 Google 的大力支持, 致力于提供 Rust 对 Linux 内核的各种支持.

2021 年 7 月 4 日, Linux 内核团队发布添加 Rust 支持的“v1”补丁;
2021 年 12 月 6 日, Linux 内核团队发布支持 Rust 的“v2”补丁。

在 2021 年底, Rust 已经成为 Linux 开发的官方语言. 据报道, 到 2022 年, 开发者有望看到 Linux 内核正式支持 Rust. 截止目前 (2022 年 4 月 3 日) 已经发布了 “v5” 补丁.

- [BlogOS](#), [GitHub Repo](#)

特点: 是一个博客系列, 每篇博客有教程和完整的代码, 对应 GitHub Repo 里的一个个 branch.

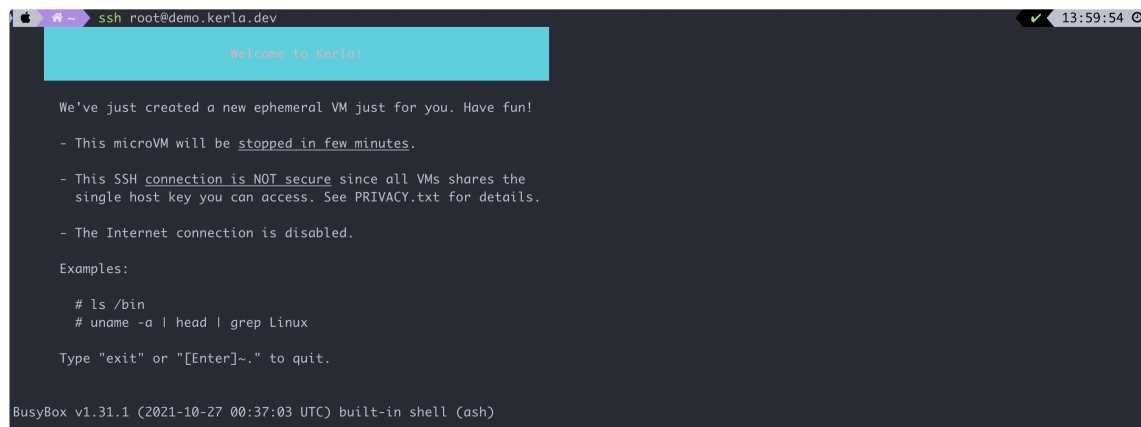
- [Kerla](#)

仓库介绍:

Kerla is a monolithic operating system kernel written from scratch in Rust which aims to be compatible with the Linux ABI, that is, it runs Linux binaries without any modifications.

也是一个 Written from scratch 的 Rust 内核项目, 目标是在 ABI 层面上兼容 Linux, 目前已经支持上下文切换, 部分系统调用, `fork` 等功能.

我们可以直接通过 ssh 进入 Kerla, 看看别人做出来的内核是什么样的:



- [redox](#), [GitLab Repo](#)

Redox is a Unix-like Operating System written in **Rust**, aiming to bring the innovations of Rust to a modern microkernel and full set of applications.

是 Pure Rust 的, 同时文档非常详细, 有一本书 [Redox book](#) 详细介绍了设计的思路, 可以作为参考.

- [rCore](#)

THU 的操作系统项目, 基于上面说的 BlogOS, 相对更贴近大作业项目的水平. 同时支持多种架构, 并可以运行在 QEMU 上.

- [intermezzOS](#)

intermezzOS is a teaching operating system, specifically focused on introducing systems programming concepts to experienced developers from other areas of programming.

一个教学用的操作系统, 旨在向其他领域的程序员介绍系统方向的编程.

- [tock](#)

是 Rust 内核, 但属于嵌入式操作系统, 关系不大, 仅供参考.

参考资料

- 《操作系统概念》亚伯拉罕·西尔伯沙茨
- [Rust 官方](#)
- [Rust VS C++](#)
- [Rust Wikipedia](#)
- [Rust 学习参考](#)
- [Rust Book](#)
- [Rust Blog](#)
- [CS 110L](#)
- [rCore-tutorial-book](#)
- [计算机自制操作系统：完成总结](#)
- [新增 3.2 万行代码，Linux 内核有望在 2022 年正式支持 Rust](#)
- [如何用 Rust 编写一个 Linux 内核模块](#)
- [开源项目：使用 Rust 写一个兼容 Linux 的内核](#)
- [Rust is now an official language for Linux development](#)
- [A comparison of operating systems written in Rust](#)