觉悟 OS 小组选题和调研报告

姬子琢, 吴宇翀, 区家彬 and 舒佳豪

2024 年 3 月 31 日

1 前言

随着计算机科学与计算机工艺的不断进步，计算机内的可用资源逐渐增多。与此同时，用户对计算机的要求也逐渐多样化，从以前的重复计算，到足不出户访问世界，再到人机交互等等。但是在这过程中，不变的基本要求是安全性与便捷性。操作系统作为计算机系统的核心软件，它通过抽象底层硬件的控制信号为我们提供接口来使用，并且最大限度地保证系统资源的合理分配与限制意外情况的发生，其重要性日益凸显。近年来，Rust 语言凭借其内存安全、并发处理和性能优化等特性，逐渐在系统软件领域崭露头角。LiteOS 作为一个轻量级的实时操作系统，广泛应用于物联网、嵌入式系统等领域，其模块化设计和灵活性使其成为学习和实践操作系统原理的优秀平台。

本小组的大作业选题是“使用 Rust 语言改写 LiteOS”。通过这一选题，我们旨在深入探究 Rust 语言在操作系统开发中的应用潜力，同时提升我们对操作系统原理和实现的理解。在改写过程中，我们将重点关注 Rust 语言的内存安全特性如何与 LiteOS 的设计理念相结合，以及这种结合如何带来更高的系统安全性和稳定性。

为了全面了解改写 LiteOS 的可行性和挑战，我们进行了深入的调研。调研内容涵盖了 Rust 语言的特性、LiteOS 的架构和源代码、以及相关的 Rust 语言编写操作系统的工作。通过调研，我们期望能够找到一条既能充分发挥 Rust 语言优势，又能保持 LiteOS 原有特色的改写路径。

在接下来的报告中，我们将详细介绍改写 LiteOS 的项目背景、本次大作业的立项依据、该项改写工作的重要性、目前各界在该领域的相关工作和预期成果。我们相信，通过本次大作业的实践，我们将能够更深入地理解操作系统的原理和设计，同时为 Rust 语言在系统软件领域的应用积累宝贵的经验。

目录

2 项目背景

2.1 物联网领域迅速发展

2014-2022 年，我国物联网市场规模整体呈稳步上升的趋势, 根据历史数据初步测算 2023 年中国物联网市场规模超过 3 万亿元。嵌入式的操作系统，广泛应用于各种微型计算机上, 调度各种硬件设备, 在物联网领域作用不言而喻，因此变得越来越重要。

2.2 操作系统面临的安全危机

近年来，windows,iOS 等操作系统上，常常会出现“漏洞”，造成了严重的财产损失, 而因为内存管理导致的内存泄漏也会导致操作系统崩溃。

根据谷歌威胁分析小组（TAG）和 Mandiant 联合发布的最新报告，2023 年全球范围内被利用的零日漏洞数量急剧上升，达到 97 个，相比 2022 年的 62 个增长了超过 50%, 攻击者利用这些漏洞, 偷窃了企业、政府的大量数据。

Tenable Research 团队统计了 2023 年微软星期二补丁数据指出，2023 年，微软共修补了 909 个漏洞, 这些对这些漏洞补丁也造成了极大的开销。

Armis Labs 安全研究人员曾在嵌入式设备实时操作系统 VxWorks 中发现了 11 个 0 day 漏洞，预计影响超过 20 亿设备。这极大影响了物联网领域的可靠性。

2.3 改写操作系统已成趋势

2022 年，在 Linux 基金会开源峰会上，Linus Torvalds 提到他希望看到在 Linux Kernel 5.20 中融入 Rust。

2023 年 3 月份，Windows 操作系统安全总监 David “dwizzle” Weston 在以色列特拉维夫的 BlueHat IL 2023 上宣布 Rust 进入操作系统内核——用 Rust 重写核心 Windows 库，并表示在接下来的几周或几个月内，大家将会在内核中使用 Rust 启动 Windows，该项工作的基本目标是将其中一些内部的 C++ 数据类型替换成 Rust。

2023 年 12 月，美国网络安全和基础设施局（CISA）联合其他机构颁发了一份《内存安全路线图指南》，其中就指出了 C 和 C++ 是内存不安全的编程语言。

可见，许多公司正在用更安全的语言对操作系统进行改写。

2.4 Rust

Rust 程序语言的开发始于 2006 年，最初是由 Mozilla 资助的问题解决项目, 属于私人项目，其目的在于减少存在于火狐 (Firefox) 浏览器引擎中的内存安全问题。(当时，Gecko 面临的问题是: 用 C++ 写的并发模块，经常出现内存泄漏问题。)

其于 2010 年正式公开发行，其第一个稳定版本在 2015 年发布. 随后按照“6-week train model”进行发布，可能每天都会产生一个开发版本，固定每 6 周发布一个测试版本。因其天然的高并发高性能，又因其独特的所有权机制能够有效保障系统安全，近年来被广泛用于操作系统领域。

2021 年 2 月 Rust 基金会宣布成立,赞助商包括了华为、AWS、Google、Microsoft、Facebook 等，基金会将会完全专注于 Rust 语言的开发与生态发展。

根据 SlashData 最近的一项调查，到 2023 年，全球大约有 280 万 Rust 开发者，这一数字在过去两年中几乎增加了两倍。

2.5 Huawei liteOS 与 OpenHarmony

Huawei LiteOS 是华为面向物联网领域开发的一个基于实时内核的轻量级操作系统, 发布于 2015 年 5 月的华为网络大会上。自发布起, 广泛应用于智能家居、个人穿戴、车联网、城市公共服务、制造业等领域, 并建立了良好的生态环境。

2015 年华为发布“1+2+1”的物联网战略，Huawei LiteOS 作为战略的重要组成部分，是支持物联网终端产业快速发展、使能终端设备智能化的软件开发平台。Huawei LiteOS 发布以来，支持了很多优秀产品的上市，包括华为高端智能手机、可穿戴设备、物联网芯片等，设备使用量已经超过 5000 万。

OpenHarmony 是华为捐献给开放原子开源基金会（OpenAtom Foundation）孵化及运营的开源项目, 目标是面向全场景、全连接、全智能时代，基于开源的方式，搭建一个智能终端设备操作系统的框架和平台，促进万物互联产业的繁荣发展。

OpenHarmony 使用先前的 Huawei liteOS 作为地基, 推出了 liteOS-m 内核，liteOS-m 面向 MCU 类处理器 (类似单片机) 例如 Arm Cortex-M、RISC-V 32 位的设备，硬件资源极其有限，支持的设备最小内存为 128KiB，可以提供多种轻量级网络协议，轻量级的图形框架，以及丰富的 IOT 总线读写部件等。可支撑的产品如智能家居领域的连接类模组、传感器设备、穿戴类设备等。

基于以上背景，我们小组准备使用安全性高的 Rust 语言对在嵌入式领域具有较好前景的实时操作系统 liteOS 进行部分改写，以提高嵌入式领域操作系统的安全性。

3 立项依据

关于使用 Rust 语言改写 LiteOS，主要从 Rust 语言本身以及 LiteOS 的相关特点进行分析：

3.1 为什么要挑选 LiteOS 作为改写对象?

工作量适合

LiteOS 是华为开发的一个开源轻量级实时操作系统，面向物联网 (IoT) 领域。它的主要特点是体积小、资源占用少，适用于微控制器单元 (MCU) 和其他资源受限的嵌入式设备。其开源项目代码量对初步接触操作系统的人来说适中，适合作为改写项目的对象。

功能齐全

目前 LiteOS 开源项目支持 ARM64、ARM Cortex-A、ARM Cortex-M0，Cortex-M3，Cortex-M4，Cortex-M7 等多种芯片架构。其内核架构包括不可裁剪的极小内核和可裁剪的其他模块。极小内核包含任务管理、内存管理、中断管理、异常管理和系统时钟。可裁剪的模块包括信号量、互斥锁、队列管理、事件管理、软件定时器等。

生态丰富

LiteOS 还集成了一些网络协议栈，比如 CoAP 和 MQTT，这使得它能够更容易地用于构建物联网应用。此外，LiteOS 支持与 Huawei LiteOS IoT SDK 和 HiLink 平台的集成，这为开发者提供了丰富的开发资源和广泛的设备生态。

目前存在的局限性

LiteOS 作为一个轻量级的实时操作系统（RTOS），在设计上主要针对资源受限的嵌入式设备。尽管它在这些领域表现出色，但仍存在一些局限性，尤其在安全性、性能、可维护性和

生态兼容性方面，我们小组主要考虑改进的是安全性。

在安全性方面，LiteOS 也在持续进行改进，增加了一些安全特性来保护设备免受常见威胁。然而，由于其设计初衷是轻量化，并未专门为高安全需求的应用设计，因此它在安全机制上可能不如一些为安全性设计更为深入的操作系统，如细粒度的访问控制、安全启动、运行时防护等。并且由于其资源受限，LiteOS 可能没有足够的资源来实施复杂的安全机制，如硬件加速的加密和解密功能。

3.2 为什么挑选 Rust 作为改写的语言？

图 1: Rust 语言 logo

3.2.1 Rust 语言的优势

Rust 语言主要在安全性、性能、生态支持方面有如下优势：

安全性

Rust 在安全性方面具有以下具体优势：

1. 所有权和借用检查:
   Rust 通过所有权（ownership）和借用（borrowing）规则强制执行内存安全，防止悬垂指针和数据竞争等问题。

2. 类型系统:
   Rust 的类型系统十分强大，它可以在编译时期捕获许多错误，包括无效的索引、类型不匹配和无效的引用等。

3. 无垃圾回收:
   Rust 不使用垃圾回收机制，因此避免了垃圾回收可能引起的延迟和复杂性，同时也减少了内存泄漏的风险。

4. 并发原语:
   Rust 提供了安全的并发原语，如互斥锁（Mutexes）、读写锁（RwLocks）、原子操作等，帮助开发者编写并发代码而不用担心数据竞争。

5. 生命周期:
   Rust 的生命周期（lifetimes）概念确保了引用的有效性，防止了悬垂引用的产生。

6. 抽象安全:
   Rust 允许开发者编写抽象代码而不必牺牲安全性，因为它的类型系统和所有权模型可以在编译时确保抽象的正确实现。

7. 错误处理:
   Rust 提供了一种表达性强的错误处理机制，要求显式地处理潜在的错误，避免了隐藏的错误和意外行为。

8. 标准库:
   Rust 的标准库经过精心设计，提供了一系列安全的数据结构和算法，减少了开发者自行实现时可能引入的安全漏洞。

9. 社区和生态:
   Rust 社区注重安全，经常进行代码审计，并在生态系统中提供了大量的安全工具和库。

10. 编译时检查:
    Rust 的编译器能够在代码编译时执行严格的检查，以确保遵守了语言的安全规则。

这些特性共同使得 Rust 成为一个在系统编程和嵌入式开发领域中备受推崇的安全语言选择。

高性能

Rust 语言在改写 LiteOS 时，在性能方面的具体优势体现在以下几个方面：

1. 零开销抽象:
   Rust 的抽象不会引入运行时开销。它允许开发者直接操作硬件，同时提供了高级抽象，这意味着可以在保持性能的同时，编写出更安全、更易于维护的代码。

2. 优化的编译器:
   Rust 的编译器（例如 LLVM）经过优化，能够生成高效的机器码。编译器会进行各种优化，比如内联函数、循环展开、向量化等，以提升执行效率。

3. 数据并行和并行计算:
   Rust 支持数据并行和并行计算，这可以充分利用多核处理器的能力，加快计算密集型任务的执行速度。

4. 无垃圾收集器:
   Rust 没有垃圾收集器（GC），这避免了 GC 可能带来的暂停和不确定的延迟。在实时操作系统中，这种确定性是非常重要的。

5. 控制流分析:
   Rust 的编译器能够进行复杂的控制流分析，优化代码路径，减少不必要的检查和跳转，从而提高执行速度。

6. 缓存友好的数据布局:
   Rust 允许开发者精确控制数据的内存布局，这有助于优化数据在缓存中的访问模式，减少缓存未命中。

7. 硬件接近性:
   Rust 提供了与硬件接近的编程接口，允许开发者编写高效的硬件操作代码，而不必通过高级抽象层。

8. 异步编程:
   Rust 的异步编程模型（通过 ‘async‘/‘await‘关键字）允许非阻塞 IO 操作，这可以在不牺牲多任务处理能力的情况下，提高 IO 密集型应用的性能。

3.3 社会与生态支持

Rust 语言项目开发有着丰富的社会与生态支持，具体表现如下：

1. Rust 语言本身的生态:
   Rust 语言拥有一个活跃的社区和丰富的生态系统，提供了大量的库（crates），这些库可以用来处理网络、加密、串口通信、硬件抽象等在嵌入式系统中常见的功能。

2. 工具链:
   Rust 提供了稳定的工具链，包括编译器、包管理器（Cargo）和各种调试工具，这些工具支持跨平台编译和部署，非常适合嵌入式开发

3. cortex-m 系列 crate:
   对于基于 ARM Cortex-M 微控制器的系统，存在一系列专门的 crate，比如 cortex-m、cortex-m-rt 和 cortex-m-semihosting，它们提供了对 Cortex-M 核心的支持，包括上下文切换、中断处理等功能。对应的 github 仓库网站：https://github.com/rust-embedded/cortex-m

4. 官方资源:
   Rust 语言的官方网站提供了大量的教程、文档和指南，帮助开发者学习如何使用 Rust 进行嵌入式开发。对应官方网站：https://www.rust-lang.org/

5. 论坛和社区:
   Rust 社区活跃在多个论坛和社交媒体平台，如 Reddit、Stack Overflow、Discord 和 IRC，开发者可以在这些平台上提问、分享经验和获取帮助。

6. 教育资源:
   存在很多在线课程、书籍和教程专注于 Rust 的嵌入式开发，例如“The Embedded Rust Book”和“Bare Metal Programming in Rust”。

3.4 市场需求

在 LiteOS 的安全性方面，当前市场可能存在以下需求：

1. 增强的内存保护:
   市场需要能够防止恶意软件和程序错误导致的内存破坏的机制。

2. 更精细的访问控制:
   为了保护敏感数据和功能，需要实现基于角色的访问控制（RBAC）或类似机制。

3. 执行保护机制:
   需要实施代码和数据执行保护措施来防止恶意代码执行。

4. 漏洞管理和响应:
   市场需要有效的漏洞发现、评估、通知和修复流程。

5. 加密和认证支持:
   需要集成更强大的加密库，支持数据加密、安全认证和数字签名等功能。

6. 网络安全增强:
   市场需求包括防火墙、入侵检测系统和支持 SSL/TLS 的网络安全解决方案。

7. 安全启动:
   为了保证系统在启动过程中不被篡改，需要实现安全启动机制。

8. 限制调试和日志记录:
   需要对调试信息和日志记录进行限制，以减少潜在的安全风险。

9. 第三方组件的安全管理:
   市场需要确保所有依赖的第三方库和组件都是最新和最安全的版本。

10. 用户教育和文档:
    对于特定行业或应用，可能需要定制化的安全解决方案来满足特殊的安全需求。

11. 合规性和标准遵从性:
    市场需求包括符合各种国际安全标准和法规，如 ISO/IEC 27001、GDPR 等。

随着物联网设备在各个行业的广泛应用，对 LiteOS 等轻量级操作系统的安全性要求越来越高，因此上述市场需求对于提升 LiteOS 的竞争力和市场份额至关重要。

3.5 项目的可行性分析

3.5.1 使用 Rust 语言带来的优势

一、内存安全

Rust 的所有权和借用模型能够在编译时期就避免很多内存错误，这对于操作系统这种底层软件来说至关重要。

二、ABI 兼容性

Rust 可以生成与 C 语言兼容的 ABI，使得 Rust 编写的模块可以与 C 语言编写的代码无缝交互。这意味着，只改写 LiteOS 其中一部分内容，在接口处理好的情况下也能够使内核运行起来，而无需全部推倒重来。

ABI 兼容性（Application Binary Interface 兼容性）是指不同程序或程序的不同部分在二进制层面上能够互相操作和交互的能力。它定义了函数调用约定、数据类型的布局、寄存器使用、栈布局、异常处理机制等底层细节。　　简而言之，ABI 是程序员在编写程序时不需要关心的底层硬件和操作系统细节，这些细节由编译器、链接器和操作系统来处理。在实际开发中，ABI 兼容性允许一个程序能够调用另一个程序（无论是库还是系统服务）提供的函数，即使这两个程序是用不同的编程语言编写的，或者是在不同的编译器版本下编译的。例如，一个用 C 语言编写的库可以被用 Rust 语言编写的程序调用，前提是它们遵循相同的 ABI。

三、性能保障

Rust 的性能接近 C/C++，对于需要高性能的操作系统模块来说，这是一个重要的考虑因素。Rust 在性能上的保障主要得益于其语言特性和编译器优化，以下是几个关键点：

1. 零开销抽象:
   Rust 力求在提供高级抽象的同时，不牺牲性能。这意味着 Rust 的抽象不会引入额外的运行时开销。

2. 所有权和借用模型:
   Rust 的所有权和借用模型确保了内存安全，同时避免了垃圾收集器的引入，这对于需要精确控制内存和性能的系统编程来说是一个巨大的优势。

3. 高效的内存分配:
   Rust 提供了多种内存分配策略，包括栈分配、堆分配和静态分配，开发人员可以根据具体情况选择最合适的方式。

4. 并发原语:
   Rust 内置了强大的并发原语，如互斥锁（Mutexes）、读写锁（RwLocks）和原子操作，这些都是以高性能为目标设计的。

5. LLVM 优化:
   Rust 编译器背后使用的 LLVM 优化框架能够生成高度优化的机器码。编译器能够进行许多复杂的优化，比如内联函数、循环展开、向量化和死代码消除等。

6. 类型系统:
   Rust 的类型系统在编译时进行大量检查，这有助于捕捉潜在的性能问题，并确保类型信息在编译时可用，以便进行优化。

7. 无运行时:
   Rust 没有标准的运行时（除了必需的少量代码，如线程管理），这意味着没有额外的运行时开销，所有的性能都直接转化为程序的执行速度。

8. 社区和生态系统:
   Rust 有一个活跃的社区，不断有新的库和工具被开发出来以优化性能。此外，许多库都经过了精心的优化，以确保在 Rust 生态系统中提供最佳性能。

9. 性能剖析工具:
   Rust 提供了各种性能剖析工具，如 cargo prof 和 valgrind 的 callgrind 工具等，这些工具可以帮助开发人员识别和优化性能瓶颈。

通过以上特性，Rust 能够在改写代码时提供与手写 C/C++ 代码相媲美甚至更优的性能。

3.5.2 LiteOS 模块化设计带来的便利

LiteOS 作为一个轻量级操作系统，其模块化设计有利于逐个模块地进行迁移，对于使用 Rust 改写模块非常有利，原因如下：

1. 独立性:
   模块化设计允许每个模块独立开发和测试，这意味着可以单独将一个或几个模块转换为 Rust，而不会影响其他模块的运行。

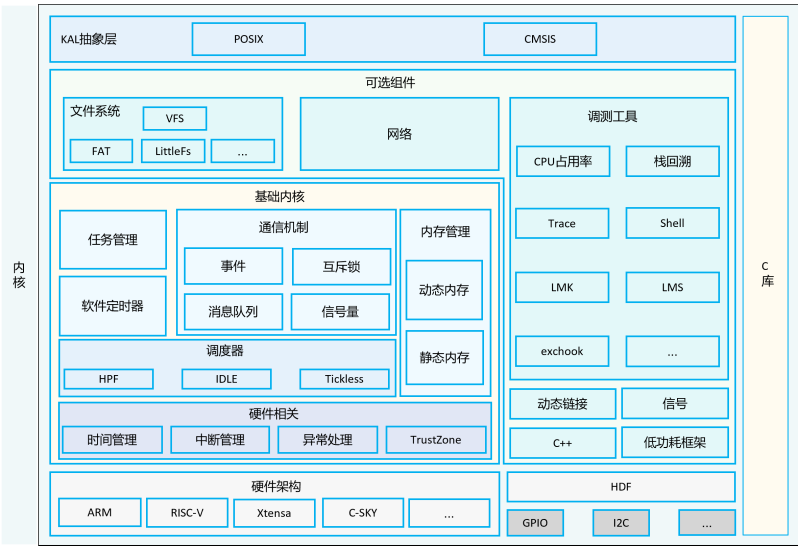

图 2: LiteOS 内核架构图

2. 清晰的接口:
   模块化设计通常伴随着明确的接口定义。LiteOS 中的每个模块都有清晰定义的功能和接口，这使得将这些模块用 Rust 重写时更容易保证与 C 语言模块的兼容性。

3. 易于集成:
   由于 LiteOS 的模块化设计，Rust 编写的模块可以作为独立的库被编译和链接到系统中。Rust 提供了与 C 语言交互的外部函数接口 (FFI)，这使得 Rust 模块可以调用 C 模块，反之亦然。

4. 逐步迁移:
   模块化的结构允许开发团队逐步迁移系统。首先可以选择性能要求最高或最容易用 Rust 重写的模块开始，然后再逐步迁移其他模块。

5. 风险隔离:
   在模块化设计中，如果 Rust 重写的某个模块出现问题，不会立即影响到整个系统的稳定性。这允许开发团队在不影响系统整体运行的情况下修复问题。维护和扩展性：模块化设计使得系统更容易维护和扩展。随着技术的发展，可以使用 Rust 的最新特性来改进或扩展系统功能，而无需重构整个系统。

6. 性能优化:
   Rust 在性能优化方面具有优势，模块化设计允许针对特定模块进行性能调优，而不会影响到其他模块的性能。利用 Rust 特性：Rust 的特性如内存安全、并发处理等，在模块化设计中可以更容易地应用于特定的功能区域，从而提高系统的整体质量。

总之，LiteOS 的模块化设计为使用 Rust 改写模块提供了灵活性和便利性，有助于实现平滑的过渡和长期的系统维护。

3.6 改写过程中可能遇到的挑战

Rust 语言为改写 LiteOS 提供了良好的语言环境及工具，而 LiteOS 的模块化设计也为改写过程中的隔离与调试提供了便利。尽管如此，在改写过程中也不可避免遇到以下的挑战：

1. 学习曲线:
   Rust 的语法和概念与 C 语言有显著差异，团队成员可能需要时间来掌握。编译器和工具链：Rust 的编译器和工具链需要能够支持裸机或接近裸机的环境，这可能需要对现有的工具链进行大量的定制和优化。

2. 标准库:
   Rust 的标准库（stdlib）并不适合直接用于操作系统开发，需要编写或集成一个适合操作系统级别的标准库，如 libcore 或 libstd。

3. 硬件抽象层:
   需要为不同的硬件平台编写硬件抽象层（HAL），这是实现跨平台支持的基础。

4. 兼容性和迁移:
   现有基于 C 语言的 LiteOS 代码库需要被迁移到 Rust，这涉及到大量的代码重写和测试工作。

5. 现有代码库:
   LiteOS 的现有代码库主要是用 C 语言编写的，将其迁移到 Rust 可能需要大量的工作，包括数据结构的设计、模块功能的具体实现、库函数的改写等等。

6. 错误处理机制:
   Rust 的所有权和借用规则强制执行了内存安全，但同时也带来了独特的错误处理机制。在改写模块时，需要合理地处理错误情况，确保系统的鲁棒性。

7. 测试和验证:
   每个改写的模块都需要经过严格的测试，包括单元测试、集成测试和系统测试，确保其功能正确且与其他模块协同工作良好。这要求测试人员需要对所有可能出现的测试输入做全盘考虑，以应对一切可能发生的意外。

4 该项改写工作的重要性

4.1 C 语言操作系统的安全问题

随着物联网和嵌入式设备的快速发展，安全性和性能成为越来越重要的考虑因素。c 语言以其简洁高效的特性使其成为了许多嵌入式系统和操作系统的首选。然而，C 语言的低级特性也为安全问题埋下了隐患，需要我们认真对待和解决。

4.1.1 安全问题分析

在 C 语言操作系统中，常见的安全问题主要包括：

缓冲区溢出（Buffer Overflow）： 当程序向缓冲区写入超出其分配空间的数据时，可能会导致数据覆盖、程序崩溃甚至远程代码执行等严重后果。

空指针解引用（Null Pointer Dereference）： 当程序试图解引用空指针时，可能会导致程序崩溃或发生不可预测的行为，存在一定的安全风险。

内存泄漏（Memory Leaks）： 在 C 语言中，动态内存的分配和释放需要由程序员手动管理，若管理不当就会导致内存泄漏问题，使得系统资源得不到释放，进而影响系统性能和稳定性。

4.1.2 0day 漏洞与黑客攻击

0day 漏洞（Zero-day vulnerability）：0day 漏洞是指在软件厂商尚未发现并修复的漏洞。黑客利用这些未知漏洞进行攻击，而软件开发商还没有来得及发布补丁。这类漏洞具有很高的危险性，因为黑客可以利用这些漏洞进行攻击，而用户和厂商很难发现并防范。一旦 0day 漏洞被公之于众，就会引起广泛关注，厂商需要尽快发布补丁来修复这些漏洞。仅在 2023 年，就发生了多起针对操作系统的 0day 漏洞攻击。

微软 Windows 和 Office

2023 年，微软产品曝出的最严重漏洞之一就是 CVE-2023-36884（CNNVD 编号:CNNVD-202307-797），这是 Windows 搜索工具中的远程代码执行（RCE）漏洞。该漏洞是在微软 7 月发布的周二补丁日中首次披露的，主要影响了 Windows 和 Office 软件。

与其他的微软漏洞相比，CVE-2023-36884 漏洞主要有两大特点：首先，RCE 漏洞在披露时没有补丁，微软仅提供了缓解措施以防止被利用，该漏洞一直到 8 月的周二补丁日才得到修复；其次，某东欧地区的网络犯罪组织将 CVE-2023-36884 用于侧重间谍的网络钓鱼活动以及出于牟利的勒索软件攻击。据微软报告，该组织的攻击目标是北美和欧洲的国防组织和政府实体。攻击者利用 CVE-2023-36884 绕过微软的 MotW 安全功能，该功能通常阻止恶意链接和附件。

苹果 iOS 和 iPadOS

苹果在 2023 年也曝出了 0day 漏洞，特别是 9 月 21 日披露的 iOS 和 iPadOS 中的三个漏洞尤为突出。这些漏洞包括：CVE-2023-41992（操作系统内核中的特权提升漏洞，CNNVD 编号为 CNNVD-202309-2064）、CVE- 2023-41991（让攻击者可以绕过签名验证的安全漏洞,CNNVD 编号为 CNNVD-202309-2065）以及 CVE-2023-41993（苹果的 WebKit 浏览器引擎中导致代码任意执行的漏洞,CNNVD 编号为 CNNVD-202309-2063）。这些漏洞被用在一条漏洞链中，用于投放商监视供应商 Cytrox 的间谍软件产品 Predator。埃及议会前议员 Ahmed Eltantawy 在 2023 年 5 月至 9 月期间成为了 Predator 间谍软件的目标。研究人员调查了其手机上的活动，发现手机感染了 Predator 间谍软件。

Linux

linux 被发现了 CVE-2023-0266 漏洞。这是 Linux 内核 ALSA 驱动中的竞争条件漏洞，可从系统用户访问，并为攻击者提供内核读写的访问权限。该漏洞是因为 ALSA 驱动程序于 2017 年被重构时更新了 64 位的函数调用而忽略了对 32 位函数调用的更新，从而将竞争条件引入了 32 位兼容层。Google TAG 在 3 月份发布报告称，针对最新版的三星 Android 手机的间谍活动中，该漏洞被作为包含多个 0-day 和 n-day 漏洞的利用链的一部分。

4.1.3 对安全语言的渴望

数据泄露、服务中断、财务损失..... 上述安全问题已经对全世界各个互联网公司造成了

巨大的经济损失，因此世界迫切需要转向 Rust 编程语言，以提升系统的安全性和稳定性！Windows 正在如日中天的 rust 改写 windows 内核，其重要性可见一斑。

4.2 RIIR（Rewrite It In Rust）

Rust 作为一种安全的系统语言，将语言层面的语义约束与编译器自动化推导深度结合，实现了更加严谨的编程风格和更加安全的编程方式。基于我们的分析，Rust 会成为时代的选择。

4.2.1 编程语言回顾

回顾过去，每一个十年，都有自己时代选择的编程语言，世界被一次又一次地改写。
20 世纪 60 年代：Fortran（因为 IBM！）
20 世纪 70 年代：BASIC（因为 Byte Magazine！）
20 世纪 80 年代：Pascal（因为结构化编程！）
20 世纪 90 年代：C++（因为面向对象！）
21 世纪初：Java（因为万维网！）
2010 年：JavaScript（因为前后端开发！）
2020 年：Python（因为机器学习！）
...
2030 年：Rust？

4.2.2 Rust 对 C 的颠覆

几年之前，微软就开始对 Rust 表现出兴趣，认为它是一种能在产品正式发布前捕捉并消除内存安全漏洞的好办法。自 2006 年以来，Windows 开发团队修复了大量由 CVE 列出的安全漏洞，其中约 70% 跟内存安全有关。

Rust 工具链一直努力防止开发者构建和发布存在安全缺陷的代码，从而降低恶意黑客攻击软件弱点的可能性。简而言之，Rust 关注内存安全和相关保护，有效减少了代码中包含的严重 bug 数量。

谷歌等行业巨头也已经公开对 Rust 语言示好。

随着业界对于内存安全编程的愈发重视，微软也在 Rust 身上显露出积极的探索热情。去年 9 月，微软发布一项非正式授权，Microsoft Azure 首席技术官 Mark Russinovich 表示新的软件项目应该使用 Rust、而非 C/C++。

现在，Rust 已经进入了 Windows 内核，Weston 表示微软 Windows 将继续推进这项工作，那么 Rust 很快就会得到广泛的应用。

4.2.3 Rust 改写 LiteOS 的重要性

使用 Rust 重写 LiteOS 的安全优势：

LiteOS 作为一款轻量级、高效的嵌入式操作系统，在物联网和嵌入式领域具有广泛应用。然而，随着嵌入式设备的复杂性增加以及对安全性和性能的要求提高，传统的 C 语言编写的 LiteOS 可能面临一些挑战。

因此，使用 Rust 改写 LiteOS 是一个非常 promising 的项目。Rust 作为一种安全、并发、

高性能的编程语言，具有巨大的潜力在嵌入式领域发挥作用。我们可以通过逐步迁移和测试的方式，慢慢地将 Rust 引入到 LiteOS 的开发中，以保证项目的稳健性和可维护性。

使用 Rust 重写 LiteOS 不仅具有重要的意义和前景，还将带来一系列潜在的长期好处：

安全性增强： Rust 的内存安全性和类型安全性将大大提高 LiteOS 的安全性水平。通过静态检查和所有权系统，可以有效防止诸如缓冲区溢出、空指针解引用等常见安全漏洞的发生，从而保护设备和用户的数据安全。

可维护性提升： Rust 的代码清晰、模块化和易于理解，有助于 LiteOS 团队更好地理解和维护代码。同时，Rust 的生命周期管理和错误处理机制能够减少代码中的潜在 bug，并简化调试和测试过程，降低开发和维护成本。

未来扩展性： 随着物联网和嵌入式设备领域的不断发展，LiteOS 需要不断适应新的硬件和技术。Rust 作为一种现代化的编程语言，具有丰富的生态系统和活跃的社区，为 LiteOS 未来的扩展和创新提供了更广阔的空间。

5 该领域内的相关工作

5.1 本课程之前的相关作业

5.1.1 https://github.com/OSH-2019/x-rust-freertos

主要内容：用 Rust 改写 FreeRTOS（一个实时、嵌入式操作系统）, 完成了对 FreeRTOS 中所有的内核模块——移植（port）模块、链表（list）模块、任务调度（task）模块和队列与信号量模块的改写。

已经实现：

1. 统一的 FFI，允许操作系统与可移植层中的 C 代码无缝通信。
2. 使用智能指针的双向链表。（在 Rust 中真的很难）
3. 基于任务优先级的全功能任务调度程序。
4. 一个固定长度的队列，在其上构建信号量和互斥锁。
5. 一组可选编译的功能，可帮助您 DIY 内核。

总体设计概览：

1. 细致的模块化设计。在原模块下又分出小模块。
2. 基于 Cargo feature 实现了内核裁剪功能，实现了 FreeRTOS 中所有的条件编译。
3. 全局变量的处理。Rust 不支持结构体作为全局变量，所以使用了 lazy_static 包来封装任务链表，并使用全局 mutable 变量来存储系统状态，并创建 getter 和 setter，用 unsafe 统一对其进行访问。
4. 完善的日志。让代码执行过程可视化，方便调试。

5. 测试：用自己编写的功能测试算例与 FreeRTOS 官方 demo 中的算例对 C-FreeRTOS 和 Rust-FreeRTOS 进行性能比较（比较用时）。

不足：

1. 没有消灭所有的全局变量。这不是 Rust 鼓励的实现方式。
2. 我们使用了 std 库，但是没有自己实现它，但是一个真正的操作系统是要自己实现 std 库。
3. 一个可能的不足：在代码中使用了过多的智能指针（Arc Weak RwLock），它们基于操作系统的信号量机制，容易造成死锁。

5.1.2 https://github.com/OSH-2023/Phoenix-Flames

主要内容：用 Rust 改写 sel4 微内核，提供内存安全性和并发安全性。
细节：

1. seL4 特有的特点：最小化原则：裁剪掉了大量系统服务。seL4 官网上给出了使用 qemu 进行模拟的内核功能测试方法和测试包。通过替换 kernel.elf 文件可以对各种内核进行功能测试。(最终这个官方性能测试无法运行) seL4 调度算法：seL4 使 带优先级的轮转调度，相同优先级下采用轮转调度，具有 256 个优先级、抢占的、tickless（无节拍）调度器。所有线程都有 个最 可控优先级 (MCP) 和 个线程的有效优先级。seL4 性能的显著指标是 IPC，sel4 的 IPC（进程间通信方式）在 L4 同步 IPC 机制基础上引入了 notification 机制，seL4 的 IPC 分为 fastpath 和 slowpath, 在尝试 fastpath 失败后才会进入 slowpath。握手机制与虚拟寄存器的引入加速了 IPC 的实现，体现了性能的优化，是微内核的优势。Notification 机制是 seL4 里用于传递信号量的机制（非阻塞），其本质为由信号量组成的一个队列。其函数涉及队列操作。
2. 编译过程：sel4test 原本的构建过程将会得到两个二进制文件, 一个是内核镜像，一个是测试程序。其中，编译内核时会生成一个叫做 kernel_all.c 的文件，该文件包含了内核所有的 C 文件，通过对此文件进行修改，可以方便地编译出我们的内核。理解大型项目编译过程是小组进展最为艰难和耗时最长的地方。

技术路线：

1. 配置 Rust 来编写可在裸机上运行的程序。使用 gcc 编译可在裸机上运行的 C 程序。
2. 明确操作系统需要为用户态程序提供虚拟内存与系统调用，这个过程中要用到 ABI 即应用程序二进制接口（Application Binary Interface），是应用程序与操作系统交互的接口。
3. 根据 seL4 的架构改写实现：虚拟内存、Capability Space、notification 机制、消息传递、线程调度。

不足：

1. 没有完全改写 seL4。
2. 未能编写有效的性能测试对比程序，同时 sel4 官方的性能测试因未知的原因无法运行；因而我们的改写只能保证运行正确，对性能的比较还有待考量。

5.2 Rust 开发的较为成功的项目

1. gcsf 基于 Google 云端硬盘的虚拟文件系统
2. tfs 模块化，快速且功能丰富的下一代文件系统
3. tock 基于 Cortex-M 的微控制器的安全嵌入式操作系统
4. Redox 类 Unix 微内核操作系统
5. rCoreOS uCoreOS 的 Rust 版本

5.3 其他关于 Rust 的工作

1. 用封装函数强化 FFI(Foreign Function Interface)，在 Rust 系统与其他语言库交互时保护内存安全。封装函数是一个在单线程内存安全系统上下文中执行不受信任的外部代码的框架。封装函数由两种互操作机制组成——轻量级上下文切换、参与基于硬件的内存保护的函数调用的实现，以及一组用于与不受信任的外部内存安全交互的类型级抽象。
2. Unishyper：一个 Rust 编写的为嵌入式场景设计的 unikernel。
   (a) 它提出了 unilib 接口，这是一组基于卸载思想的接口，它跨越了 unikernel 和正在运行的 hypervisor 平台。基于 Unilib, unikernel 可以方便地使用运行在其他虚拟化分区上的通用操作系统提供的丰富的通用库接口，并选择卸载部分协议栈，进一步简化了 unikernel 的结构。
   (b) 故障记录和恢复。为了满足嵌入式场景对系统可靠性和易于追溯性的要求，Unishyper 在 Rust unwind 的基础上实现了故障记录和恢复机制，包括回溯错误调用链、故障处理和资源释放等。
3. ASAP（Adaptive Systems Architecture Project）是一个由亚马逊网络服务（AWS）赞助的研究项目，旨在开发一个高安全性、高性能的操作系统。这个项目特别关注于安全性和可适应性，希望能够在不断变化的硬件和软件环境中保持系统的稳定和安全。在考虑使用 Rust 语言重新编写操作系统的关键组件时，项目团队主要是基于 Rust 语言的几个显著优点做出这一决策的。
   (a) 首先，Rust 提供了强大的内存安全保障，通过其独特的所有权和借用检查机制，能够在编译时期就避免许多常见的内存错误，比如悬空指针、双重释放等。这对于提高操作系统的稳定性和安全性至关重要。
   (b) 其次，Rust 具有出色的并发编程能力。它提供了一套丰富的并发原语，如互斥锁、信号量、通道等，以及一种独特的所有权模型来帮助开发者编写安全的多线程代码。这使得 Rust 成为开发需要高度并发和实时性能的操作系统的理想选择。
4. Google 的 Fuchsia 操作系统采用了 Rust 语言来开发其部分组件。Fuchsia 是一个全新的操作系统平台，旨在为多种设备提供支持，从智能手机到个人电脑再到物联网设备。Fuchsia 使用了名为 Zircon 的微内核，而在 Zircon 之上，Google 鼓励开发者使用 Rust 来编写设备驱动和应用程序。
   使用 Rust 的原因在于其内存安全的特性，这有助于提高系统的稳定性和安全性。Rust 的所有权和借用检查模型可以防止常见的内存错误，从而减少安全漏洞和系统崩溃的风险。这对于一个面向未来、注重安全性的操作系统来说是非常重要的。
   Google 通过提供 Rust 工具链和 SDK 支持，以及在 Fuchsia 中集成 Rust 运行时，促进

了 Rust 在 Fuchsia 开发中的使用。这种做法不仅有助于提高 Fuchsia 自身的质量，也推动了 Rust 语言在系统级软件开发领域的应用和发展

5. ARM Pelion OS 是 ARM 公司提供的一个多功能物联网操作系统，它支持各种类型的连接设备，包括传感器、网关和可穿戴设备。Pelion OS 旨在提供安全可靠的运行环境，以满足物联网设备在实际应用中对数据处理和设备管理的需求。
   在操作系统的开发过程中，ARM 认识到了 Rust 语言的优势，特别是其内存安全的特性，这对于提高系统的整体安全性至关重要。因此，ARM 鼓励并支持在 Pelion OS 中使用 Rust 来编写设备驱动程序和其他关键组件。Rust 的内存安全保证有助于减少安全漏洞，这对于物联网设备来说尤其重要，因为这些设备往往暴露在外部网络中，容易受到攻击。
   通过使用 Rust，ARM Pelion OS 能够利用该语言的现代特性，如并发原语、模式匹配和包管理系统 Cargo，这些都有助于开发高效、安全且易于维护的代码。Rust 的生态系统也为 Pelion OS 带来了丰富的库和工具，进一步加速了开发进程。
   总的来说，ARM 通过在 Pelion OS 中采用 Rust 语言，展现了对安全性和可靠性的承诺，同时也推动了 Rust 在系统级编程领域的应用和发展。

6 目标与预期成果

我们小组计划对 LiteOS 的任务调度、队列管理与内存管理模块进行改写，大致计划为：

1. 1-2 周学习 Rust 相关语法知识，并且对 LiteOS 待改写模块代码进行分析 (如函数的功能，相互的引用关系等)；
2. 5-6 周使用 Rust 改写模块，并对模块进行功能性测试，准备中期汇报；
3. 3-4 周对模块进行底层链接，在虚拟平台或硬件上验证内核的功能；
4. 1-2 周对小组成果进行评估，撰写总结报告。