

**本科生毕业设计(论文)**

# 开题报告

**北京理工大学本科生毕业设计（论文）题目**

**The Subject of Undergraduate Graduation Project (Thesis) of Beijing Institute of Technology**

|  |  |
| --- | --- |
| 学 院： | 计算机学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 07112101 |
| 学生姓名： | 戴骏翔 |
| 学 号： | 1120210610 |
| 指导教师： | 陆慧梅 |
| 校外指导教师： | 向勇 |

|  |
| --- |
| 1. **选题依据**   随着嵌入式系统和安全关键系统对可靠性和安全性的需求不断提升，微内核架构逐渐成为操作系统设计中的重要方向。在这一领域，sel4 微内核因其卓越的安全性和高效性获得了广泛关注。选题中的rel4是使用rust改写的sel4，并针对ipc和系统调用实现了异步化，现在正处于开发阶段，而作为参考的开源网络协议栈，选择了smoltcp协议栈，接下来将从该三个方面介绍选题背景。  **1.seL4：**  seL4 是全球首个通过形式化验证的操作系统内核，其自 2004 年开始开发，并于 2014 年实现开源。它采用微内核架构，其许多特性保证了其作为嵌入式系统内核的广泛应用场景。  首先是安全性：sel4 通过比传统 linux 更为严格的能力机制实现安全高效的访问控制，从而提供了细粒度的资源管理能力。此机制使得任务能够严格遵循最小权限原则，显著降低了恶意操作和意外错误的风险。  其次是性能优化：尽管微内核设计的初衷并非追求处理速度，但sel4为了应对现实使用的需要，实现了成为处理速度最快的微内核。  最后是形式化验证：sel4 是首个通过数学形式化方法验证正确性的操作系统内核。这一验证保证了理论上内核没有bug，确保其具备严格的可靠性。其形式化验证是安全性的重要保障  以上这些特点，使其广泛应用于军事、航空航天、铁路信号和医疗设备等拥有高安全需求的领域。  **2.Rust 语言**  rust 语言因内存安全成为近年来操作系统编程的可靠工具，其无需依赖垃圾回收等机制和方便的编译器环境适合作为系统编程语言。而其安全性与sel4匹配，适合开发对安全性和性能要求苛刻的系统组件。  **3.smoltcp开源网络协议栈**  smoltcp是一个轻量级、Rust实现的嵌入式TCP/IP协议栈，专为资源受限的嵌入式系统设计。其通过利用Rust语言的内存安全特性和零成本抽象，提供了高性能、高安全性的网络协议栈解决方案，适用于低功耗设备、传感器节点等场景。  Smoltcp在低内存开销和高性能的基础上，提供模块化、可扩展的网络协议支持。其完整实现了TCP、UDP、ICMP、ARP等协议,并且现在依然在不断更新开发中。是一个理想的学习和可供调整使用的嵌入式协议栈。 |
| 1. **研究目标和内容**   研究的目标是基于 Rel4 微内核，结合rel4已经实现的异步系统调用和异步IPC，设计并实现一个高性能的异步网络协议栈，使其具备 Socket API 兼容性，适应嵌入式系统的需求，保证高性能。  在主要内容方面，结合学长之前的研究方向，确定使用smoltcp作为参考的开源网络协议栈进行设计， smoltcp作为一个轻量化的tcp模型，比较适合rel4嵌入式的特征，而研究的重点在于对其进行修改，使得其与rel4进行适配，这个过程中得以与smoltcp的广泛使用，有较多的benchmark可以直接使用，方便后续测试。smoltcp作为一个为嵌入式提供帮助的开源协议栈，其设计是面向没有标准库的环境，并且大部分协议有进行简化，并不是一个完全按照七层架构进行设计的网络协议栈，所以首要需要进行分析理解其运行机制。在此基础上其性能并没有进行很强的优化，很多方面都有设计上的不完美之处，现在为了适配异步机制的Rel4微内核，提供更好的网络性能，对smoltcp进行异步的修改，并在一些函数上进行设计的调整是必要的。  具体来说，如smoltcp在readme中的介绍，因为其开发团队并不是很大，很多有用的功能都还没实现，所以此时并没有集中对其异步性进行调整，如其发送接收的关键函数poll是依然通过同步去进行的调用，这样的设计局限了其发送接受效率。虽然smoltcp现在也是支持以异步的方式来执行的，也就是通过waker来实现通知，从而实现socket操作的非阻塞执行，但是这一部分异步的覆盖面并不大，实际上，在之前学长有进行测试，即使小部分的异步修改就能产生很大幅度的性能提升，所以进一步进行异步的修改依然存在必要性。在现在的代码中，异步的存在并不广泛，有很大的改进空间，如修改同步执行的poll，并分离其发送接收的功能。 |
| 1. **研究方案**   **1.已完成内容：**  1.学习了Rust异步编程，在前面rust知识的基础上，主要学习 Rust 异步框架的设计和实现细节，如Rust 提供的 async/await 等语法，并在此过程中复习rust在操作系统开发中的代码组织，确保代码的简洁可读性，同时保持代码的安全和稳定性。  2.对smoltcp进行简单的代码阅读以及分析，了解其运行环境以及函数作用，并结合对rel4的基本了解，判断不能直接使用async等rust提供的异步编程工具，确定后续需要使用异步ipc实现协议栈的异步。  **2.后续研究方案：**  1.配置实验环境，首先配置Linux 双系统以保证开发与测试的效率，在此基础上使用docker配置rel4环境，使在开发过程中与设计rel4异步化的学长保持一致，避免环境差异带来的问题。之后跑通未修改前的smoltcp，实现对于已有工作的全部复现，为后续工作提供环境  2.学习sel4的具体知识，并阅读rel4代码，因为最终的目标是与之后需要结合rel4 的机制来进行编程，需要分析学习异步化后 Rel4 微内核中的ipc机制，明白如何利用 Rel4 的异步 IPC 来实现协议栈的异步操作，并尝试是否可以通过使用相应的异步系统调用实现操作上的简化。  3.学习与分析 smoltcp 协议栈，在单纯的代码学习基础上，针对可改进的部分精细化理解其实现，如理解调用关系，明确需要改写的函数，结合论文以及网上资料学习异步编程的知识，将知识点与smoltcp中的实际代码结合。  4.完成开源仓库了解和环境部署基础后，进行代码编写。其中分为smoltcp本身的异步化增强和socket通信等机制的补充，因为需要把 Rel4 的异步 IPC 机制引入到协议栈的通知机制中，所以在此过程中需要保证与rel4的适配，没有调用无法使用的函数。  5.研究首先在qemu上进行，先跑通项目，再进一步的提升性能，因为进行异步的修改主要目的是提升性能，所以需要通过实际的发包测试与之前的结果比较得到具体的数据，便于进行后续的结果展示以及确认异步的具体效果 |
| **四、研究计划及进度安排**  1．学习ReL4和seL4的相关论⽂、背景知识和熟悉代码。（1-2周）。 .  2．学习并掌握Rust异步编程基础（3-4周）。 .  3．学习smoltcp的代码，熟悉其实现，进行环境配置，设计对其进行修改的方案（5-8周）。  4．编码和调试（9-11周） .  5．进⾏性能测试并分析数据。（12周）  6．完成本科⽣毕业设计（论⽂）外⽂翻译。（第 1 周-第 7 周） .  7．完成毕业论⽂，提交软件及相关⽂档。（第 13 周-第 14 周） .  8．完成本科⽣毕业设计（论⽂）答辩。（第 15 周） |
| **五、创新点及预期研究成果**  **1.改进smoltcp的创新点**  首先有效的提高性能和资源利用率，使用async异步编程，在网络事件发生时才触发处理逻辑，可以减少 CPU 占用率并提高响应速度，有效的提升网络发包速度。在资源受限的环境中，异步模型可以更高效地利用硬件资源，特别是在低功耗或实时性要求较高的场景中。在多连接、高并发的场景下，异步模型也可以显著提升性能和可扩展性。  其次简化了代码逻辑，通过异步编程，可以将底层的同步机制更改为异步任务，用户代码可以更简洁地使用await来等待网络事件，提供更直观的 API，降低代码的阅读成本，同时减少潜在的错误和资源管理问题。  **2.Rel4 的开发愿景**  Rel4 项目结合了 seL4 的微内核特性和 Rust 语言的安全性，可以构建一个更可靠的微内核系统。使用Rust 语言的丰富类型系统和内建的错误处理机制有助于减少常见的编程错误，保证**内核代码的可维护性**：通过 Rust 的内存和并发安全特性，进一步降低低级内存管理中的安全隐患，保证内核的安全性，**而**Rel4 的成功开发将为未来更多的微内核项目提供参考，实现更安全和高效的操作系统内核。 |
| **六、参考文献**  [1] Klein G, Elphinstone K, Heiser G, et al. seL4: Formal verification of an OS kerne[C]//Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. 2009: 207-220.  [2] Heiser G, Elphinstone K. L4 microkernels: The lessons from 20 years of research and deployment[J]. ACM Transactions on Computer Systems (TOCS), 2016, 34(1): 1-29.  [3] Klimiankou Y. Micro-CLK: returning to the asynchronicity with communication-less microkernel[C]//Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems. 2021: 106-114.  [4] Heiser G. The seL4 Microkernel–An Introduction[J]. The seL4 Foundation, 2020, 1 |
| **七、指导教师意见**  **签字：**  **年 月 日** |
| **成绩： ，占比： %** |
| **八、开题审核负责人意见**  签字：  年 月 日 |