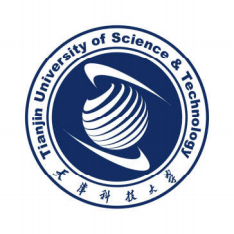


毕业设计

(2023届·本科)



毕设(论文)题 目： 基于Rust的跨平台实时操作系统的设计与优化

学 号： 19036212

学 生 姓 名： 王 炼

学 院 名 称： 人工智能学院

专 业 名 称： 物联网工程

指 导 教 师： 梁 琨

**2023年6月**

**基于Rust的跨平台实时操作系统的**

**设计与优化**

**Design and Optimization of Cross-Platform Real-Time Operating System Based on Rust**

|  |  |
| --- | --- |
| **专 业：** | **物联网专业** |
| **姓 名：** | **王 炼** |
| **指 导 教 师：** | **梁 琨** |
| **申请学位级别：** | **学 士** |
| **论文提交日期：** | **2023年6月3日** |

**天津科技大学**

**学位论文原创性声明**

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全知晓本声明的法律后果由本人承担。



学位论文作者签名：

日期：2023年6月3日

**天津科技大学**

**学位论文使用授权书**

本人同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。

本学位论文属于 **：**

☑**公开论文**

□**内部论文**，保密□1年/□2年/□3年，过保密期后适用本授权书。

□**秘密论文**，保密 年（不超过10年），过保密期后适用本授权书。

□**机密论文**，保密 年（不超过20年），过保密期后适用本授权书。

（请在以上方框内选择打“**√**”）

学位论文作者签名： 指导教师签名：

日期：2023年6月3日 日期：2023年6月3 日

天津科技大学本科毕业设计(论文)任务书

学 院：人工智能学院 专 业：物联网工程

学生学号：19036212 学生姓名：王炼 指导教师姓名:梁琨

完成期限：2023年1月5日至2023年6月10日

一、题目名称：基于Rust的跨平台实时操作系统的设计与优化

二、设计(论文)内容及要求：

该设计以Rust作为主要编程语言，编写一个可适用于物联网设备的跨平台实时操作系统，支持x86\_64、AArch64、RISC-V64三种不同架构，并对系统的完整性、实时性等方面进行设计与优化。具体要求如下：

（1）进程调度设计：可以支持RR时间片轮转调度，改进CFS完全公平调度，降低实时性测试的数值。利用Rust语言对无标准库支持的红黑树数据结构进行设计与实现。

（2）显示输出优化：支持x86\_64真实物理机的grub引导启动。显示输出包括串口输出和VGA显示两种方式，使适用场景更加丰富。

（3）增加网络通信模块：使系统支持UDP通讯，将网卡设备统一以MMIO方式映射到内存进行管理，可以对接内核模块，保证操作系统为网络通信提供所需的系统调用，实现三种架构的中断逻辑，封装为统一接口供上层调用。

（4）硬件开发板移植：增加RISC-V架构的硬件开发板K210支持。

参考文献：

[1] Redha Gouicem and Damien Carver, Jean-PierreLozi et al. Fewer Cores, More Hertz[J]. 2020 USENIX Annual Technical Conference. 2020， 6（15）:435-448

[2] Ricardo Bianchini and Ram Rajamony. Power and energy management for server systems[J]. Computer. 2004, 37(11):68–76.

[3] 易家颖,司宾强,朱纪洪. 嵌入式实时操作系统浅析[C].第十六届全国信号和智能信息处理与应用学术会议论文集.2022:375-381.

[4] 潘庆霖. zCore 操作系统内核的设计与实现[D].北京:清华大学.2020.

[5] 王润基. Rust 语言操作系统的设计与实现[D].北京：清华大学.2019.



指导教师签名：

填写日期：2023年2月20日

注：本任务书发给学生，毕业设计（论文）完成后装入毕业设计（论文）档案袋。

摘 要

现今，物联网设备的快速发展让人们的生活更加便捷，根据Statista的数据，到2025年，全球物联网设备的数量将达到386亿台。物联网操作系统作为物联网设备的控制中心，控制设备的硬件资源使用和与外界的交互。因此物联网操作系统实时性、安全性等指标具有极高的要求，这也是评价操作系统设计的关键指标。

本项目基于开源操作系统采用了Rust语言，设计和实现了支持多个平台的嵌入式设备使用的微型实时操作系统NerdOS。NerdOS以当今物联网主流工业界开源操作系统如TencentOS Tiny 和 AliOS Things以及知名的FreeRTOS为基准进行开发。由于国内大部分开源项目都瞄准于可裁剪和定制化的设计目标，国内外学者们也在研究和探索相关的模块化设计。因此本设计大量借鉴了国内外成熟的开源项目，探索性的引入模块化和定制化的设计理念。

目前本项目完善了多个平台的统一调用接口，主要工作和贡献有：（1）支持和外界的UDP通信；（2）支持不同格式的显示输出包括串口输出和VGA输出；（3）在实时性优化上探索中断优化和调度算法的设计。现在NerdOS可以在QEMU仿真中支持x86\_64、AArch64、riscv64架构的运行，也可以在勘智k210开发板和x86物理机引导启动。经过完整的性能测试后系统，系统实时性达到实时操作系统要求，和外界交互上UDP通信可以正常发送消息，其他系统调用测试程序也正常运行。NerdOS全程在GitHub开源实现，目前收获到了62个星标，项目地址是https://github.com/cargo-youth/nerdos。

关键词：实时操作系统；模块化设计；调度算法；跨平台

**ABSTRACT**

Nowadays, the rapid development of IoT devices makes people's lives more convenient. According to Statista, by 2025, the number of IoT devices worldwide will reach 38.6 billion. As the control center of IoT devices, the IoT operating system controls the use of hardware resources of the devices and the interaction with the outside world. Therefore, indicators such as real-time performance and security of the IoT operating system have extremely high requirements, which are also key indicators for evaluating the design of the operating system.

This paper uses the Rust language based on the open source operating system, and designs and implements NerdOS, a miniature real-time operating system used by embedded devices that support multiple platforms. FreeRTOS was used as the benchmark for development. Since most domestic open source projects are aimed at tailorable and customized design goals, scholars at home and abroad are also researching and exploring related modular designs. Therefore, this design draws heavily on mature open source projects at home and abroad, and explanatorily introduces modular and customized design concepts.

At present, NerdOS has improved the unified call interface of multiple platforms. The main work and contributions are: (1) Support UDP communication with the outside world; (2) Support display output in different formats, including serial port output and VGA output; (3) Explore real-time optimization Design of interrupt optimization and scheduling algorithms. Now NerdOS can support the operation of x86\_64, AArch64, RISCV64 architecture in QEMU emulation, and can also be booted on Kendryte k210 development board and x86 physical machine. After a complete performance test of the system, the real-time performance of the system meets the requirements of the real-time operating system, UDP communication can send messages normally when interacting with the outside world, and other system call test programs also run normally. NerdOS is open-sourced on GitHub. Currently, 62 stars have been obtained. The item address is https://github.com/cargo-youth/nerdos.

**Key words**：Real-time operating system; Modular design; Scheduling algorithm; Cross-platform

目 录

[第一章 引言 1](#_Toc137265915)

[第一节 课题背景及意义 1](#_Toc137265916)

[第二节 本文结构说明 2](#_Toc137265917)

[第二章 NerdOS内核系统分析 4](#_Toc137265918)

[第一节 可行性分析 4](#_Toc137265919)

[第二节 系统功能结构设计 5](#_Toc137265920)

[第三章 NerdOS内核系统设计 11](#_Toc137265921)

[第一节 使用相关技术介绍 11](#_Toc137265922)

[第二节 调度算法数据结构的设计 17](#_Toc137265923)

[第四章 NerdOS系统内核实现 20](#_Toc137265924)

[第一节 开发环境 20](#_Toc137265925)

[第二节 各模块系统实现 21](#_Toc137265926)

[第五章 NerdOS系统内核测试 25](#_Toc137265927)

[第一节 测试目的和意义 25](#_Toc137265928)

[第二节 测试方法 25](#_Toc137265929)

[第三节 测试内容 26](#_Toc137265930)

[第四节 测试结果 26](#_Toc137265931)

[第六章 总结与展望 30](#_Toc137265932)

[第一节 结论 30](#_Toc137265933)

[第二节 未来展望 30](#_Toc137265934)

[致 谢 34](#_Toc137265935)

1. 引言
2. 课题背景及意义
3. 选题目的和意义

近些年来随着物联网浪潮席卷世界，各种物联网设备层出不穷，根据Statista的数据，到2025年，全球物联网设备的数量将达到386亿台，物联网时代的到来带来的是万物互联的便捷，更是一片新兴技术的蓝海。其中作为物联网设备的控中心—物联网操作系统也进入爆发增长时期，各大企业也紧跟发展趋势推出了自己的操作系统。这其中为了能够及时处理相关请求，基本大多数都采用了实时操作系统来满足各种需求。其中比较著名的物联网实时操作系统都进行了开源，设计者们也能够从这些项目中窥见物联网实时操作系统的一些设计思路和现实意义。以国内两款开源物联网实时操作系统TencentOS Tiny 和 AliOS Things为例，内核设计者可以看到其可裁剪和定制化的设计的优秀思路，使得应用场景更加的广泛。本设计的选题来源于全国大学生系统能力培养赛阿里巴巴开源社区的题目，用Rust重写AliOS-Things内核，经过调研赛题相关参考项目和目前Rust书写的开源RTOS项目，可以发现目前很多的项目设计，一是缺乏文档和启动配置，二是很少及时更新和同步，三是一些大型项目因为巨大的工程量和耦合性很难被借鉴。在查阅大量的开源项目后，最后决定在组委会给出的赛题基础上结合清华大学贾越凯博士的一个开源项目进行设计和优化。用Rust进行内核的书写对比传统C语言有一定的优势，这主要得益于Rust语言的一些特性，避免了内核开发的内存错误和并发访问错误[1]，以Windows为例，在微软的一次演讲中提到过Windows的安全漏洞有70%是内存安全问题，那么可以想象Linux中同样有这样的问题。因此，Linus Torvalds 曾提到希望看到在 Linux Kernel 5.20 中融入 Rust。在此背景下，Rust书写内核成为了目前工业界和学界的主要趋势。

本文设计开发了一个基于Rust的跨平台实时操作系统，取名为NerdOS。由于跨平台的支持，可以支持多种架构运行，同时也可以进行广泛的开发板适配，目前支持QEMU仿真和真实x86物理机启动，暂时支持的开发板是勘智k210。现支持了UDP通信并进行了多次本地测试，在调度算法的优化上也探索了目前尚未有任何参考的内核态的Rust设计的完全公平调度[2]。在Rust开发实时操作系统方面提供了一些思路并且可以具备一些实际用途。

二、国内外研究现状

国外优秀的Rust编写的实时操作系统有Tock OS用于传感器网络和物联网安全的嵌入式实时操作系统。该项目在GitHub享有4.2k个star的评价，它的创新之处在于基于Cortex-M和RISC-V的嵌入式平台，运行多个并发的、相互不信任的应用程序。Tock的内核和驱动程序都是由Rust编写的，得益于Rust是一种提供编译时内存安全和类型安全的系统编程语言，因此Tock天然的有安全性上的优势，其次Tock使用内存保护单元将应用程序彼此隔离。这些设计上能够看到Tock对于驱动设计的相关优秀思路和对于安全性的追求。

国内方面对于Rust编写内核有着许多成功的尝试，清华大学的开源教学操作系统Rcore系列[3]，不仅仅是代码精美而且对于理解和学习操作系统相关知识有着非常好的效果，在和相关社区维护的学者和工程师的交流中，可以发现首先这是社区紧跟国外发展的趋势结合麻省理工大学、加州大学伯克利分校、卡内基梅隆、斯坦福这样的北美四大顶尖计算机强校最新的课改成果，在实践的基础上决定以新的RISC-V架构作为基础采用Rust书写内核，并且重新设计实验和指导书后推行展开，多年的发展下来证明Rust书写操作系统成功的可能性，在此之后继续推出了微内核Zcore[4]和模块化操作系统AcreOS等等。而作为阿里社区出题的参考给出了中科院相关工程师对于Rust重写FreeRTOS进行了许多的尝试。这些已有工作实现了两方面的改进，一是统一的FFI接口可以使得与C语言进行无缝通信；二是使用智能指针的双向链表作为优化的数据结构。这些工作为Rust重写C语言的相关实时操作系统提供了一些可能的思路。

总体来看，使用Rust语言进行操作系统的编程，目前已经有相当数量的成熟开源项目，对于适用于物联网领域的实时操作系统探索来说，Rust也拥有着不俗的表现。而大家的目光都聚焦在实时操作系统内核的安全性和及时性，这些作为物联网设备核心的特性也自然需要得到额外的重视[5]。本设计在借鉴现有的优秀开源项目上，进行了对于实时性性能提升的优化探索，尤其是进程调度算法方面做了许多创新的工作，而在驱动支持方面为了和外界通讯支持了UDP相关的网卡驱动，这里需要统一不同架构的中断接口在网络传输数据时能够保证独占访问，本设计还支持GURB 启动规范引导和进行示例演示。

1. 本文结构说明

本文包括六个章节，以下是对每章的标题和大致内容进行介绍：

第一章：导论，这一章重点介绍了本毕业设计的选题背景、系统设计依据和国内外研究Rust实现操作系统现状。

第二章：内核系统分析，这一章从系统的可行性分析、系统的功能设计两方面进行了具体、合理地剖析。在科技层面、经济层面、功能层面、运行层面以及社会层面五大层面上展开了可行性分析。在系统的功能设计中，提出了系统的软件结构和功能框架，为进一步完善系统的体系结构提供了依据。最后对该系统使用到的技术进行了具体的介绍。

第三章：系统详情，这一章节详细介绍了本系统用到的模块化设计，调度算法和一些优化方向。

第四章：系统详细设计。这一章对本软件系统的开发环境和各模块的开发设计做出了比较详细的介绍，从总体设计思路、详细代码、设计流程图等方面进行了详细的阐述。

第五章：测试系统，这一章首先说明了测试系统的目的和意义。重点是可用性和系统实时性的测试，根据测试结果进行优化和改进。

第六章：总结，这里重点是总结本系统设计的优缺点，将和主要的开源实时操作系统比较，探索本设计可能的优化和研究方向，之后对于整个Rust编写实时操作系统领域贡献一些可能的参考。

1. NerdOS内核系统分析
2. 可行性分析
3. 技术可行性

这个模块是对开发本设计所用到的技术和相关工具链进行分析。在开发语言上采用了Rust语言，这是由Mozilla开发一款系统编程语言，它在自身独有的内存安全[6]和并发性[7]、性能优化上得到的很多基础架构公司的应用，因此在内核编程和数据库存储KV等都占据了很高的比例。内核方面Google开发的Fuchsia其雄心勃勃的计划和现今取得的成绩给予了Rust在内核编程领域的很多启示和一定的引导，数据库领域国内知名的数据库公司PingCAP编写的tikv存储引擎使用了Rust，在教学操作系统上Blog\_os在2015年开始在GitHub开源并且发布教程和新的开发计划，目前该项目取得了11.9K的star量同时有多种语言的博客和说明文档参考。可以说是Rust较早的一批内核书写探索。在谈及为何选择Rust上这些设计取得的成绩也是回答技术可行性的答案。Rust具有以下优点：

（1）静态语言，最大程度的保证性能；

（2）无垃圾回收机制（GC），无需手动释放内存；

（3）Memory safe，尽量避免 dangling pointer，memory leak 等问题；

（4）Therad safe，不会遇到data race等问题；

（5）优秀的包管理；

（6）高效的C语言绑定[8]，这使得和C交互不会有额外的开销。

这些优良的机制和开发特性使得采用Rust进行编程时拥有着高效安全的保证。在仿真方面，本设计采用了QEMU7.0进行模拟，因为其支持多种CPU架构，保障开发跨平台操作系统时得到有利的仿真支持，并且可以在QEMU上面进行不同架构之间的仿真运行，从而得到相应的反馈来完善和优化。在编译工具方面，本设计采用了Linux-musl-cross交叉编译工具，它的优点在于即保证足够安全性的同时，也增加和其他POSIX兼容系统的兼容性，这样就可以用它来编译创建产生高效且安全的二进制文件。

二、 经济可行性

本设计可以在虚拟化环境上运行，目前是部署在腾讯云服务器上即可运行，在编译iso文件后也可用真实的x86物理机引导启动，在更换引导程序后可以适配一些开发板（如K210开发板），其目前售价相对比较便宜，官网给出的售价参考区间在几百人民币的范围，因此本设计的开发和调试都是比较便宜和方便的。

三、 操作可行性

本设计统一了不同架构之间的兼容，旨在将不同的抽象实现细节统一为上层留下了调用接口，同时调试和编译等有规范的shell脚本和Makefile文件，可以便捷的进行开发调试。在项目环境配置上贴心的放置了各种环境支持的终端指令。目前在GitHub上README文件上书写了环境配置和启动命令以及代码树导读。

四、 功能可行性

本设计增加UDP通信支持，多种平台运行和GRUB启动。在反复的调试和测试上均无异常。启动系统实时性测试软件运行得到的值在实时操作系统允许的范围内。

五、 运行可行性

本设计已经在开源社区开源，目前收获了62个star的评价，在开发和调试的过程中接受了大量开源社区开发者们的意见和建议，经过一番改进，在其本地的环境运行测试均无异常。同时在环境配置上给出了相关的终端命令和常见问题说明，以及如何启动测试和相关的代码树导读，本设计一直开放issue接受开源社区的意见。以上表明本设计已经打磨为一个比较成熟可靠的开源项目，可以支持实际的运行和调试，具备一定的研究和学习价值。

六、 社会可行性

本设计全程开源于GitHub遵循GPL-3.0 license协议，设计得到了一些社区爱好者的改进，尤其是一些相关领域的学者和工程师提出了许多改进建议和指导，在他们的帮助下项目成功做出了一些成果，在开发过程对参考的仓库和个人都在文中有所提及，并获得了他们的授权许可。总结来看本设计遵照主流开源协议并无学术不端等不道德和违法行为，因此具有社会可行性。

1. 系统功能结构设计

本设计主要基于清华大学贾越凯博士的开源项目，一个实验性的实时操作系统。同时结合其他的开源项目，如清华大学的rcore项目、Solana开发的no\_std红黑树、阿里社区AliOS Things、腾讯的开源物联网内核TencentOS Tiny ，以及很多开源的Rust书写的嵌入式内核，结合它们的优点探索模块化实时操作系统的设计和优化，主要是为了降低实时响应的时延，支持多种架构的设计。网络通信协议参考了开源的smoltcp项目和启元实验室杨金博工程师的一些实现。其中需要注意的是no\_std红黑树设计给出了和其他B树以及平衡二叉树的性能对比，可以看出它有着不俗的性能表现，这在内核编程这样穷尽性能的地方做到了极致。图2-1展示了本文所设计的跨平台实时操作系统NerdOS的总体框架，它可以分为内核态（SuperVisor mode）和用户态（User mode），以及一些独立于操作系统的软件库和真实的物理硬件。下面将对几个重要的部分进行说明。

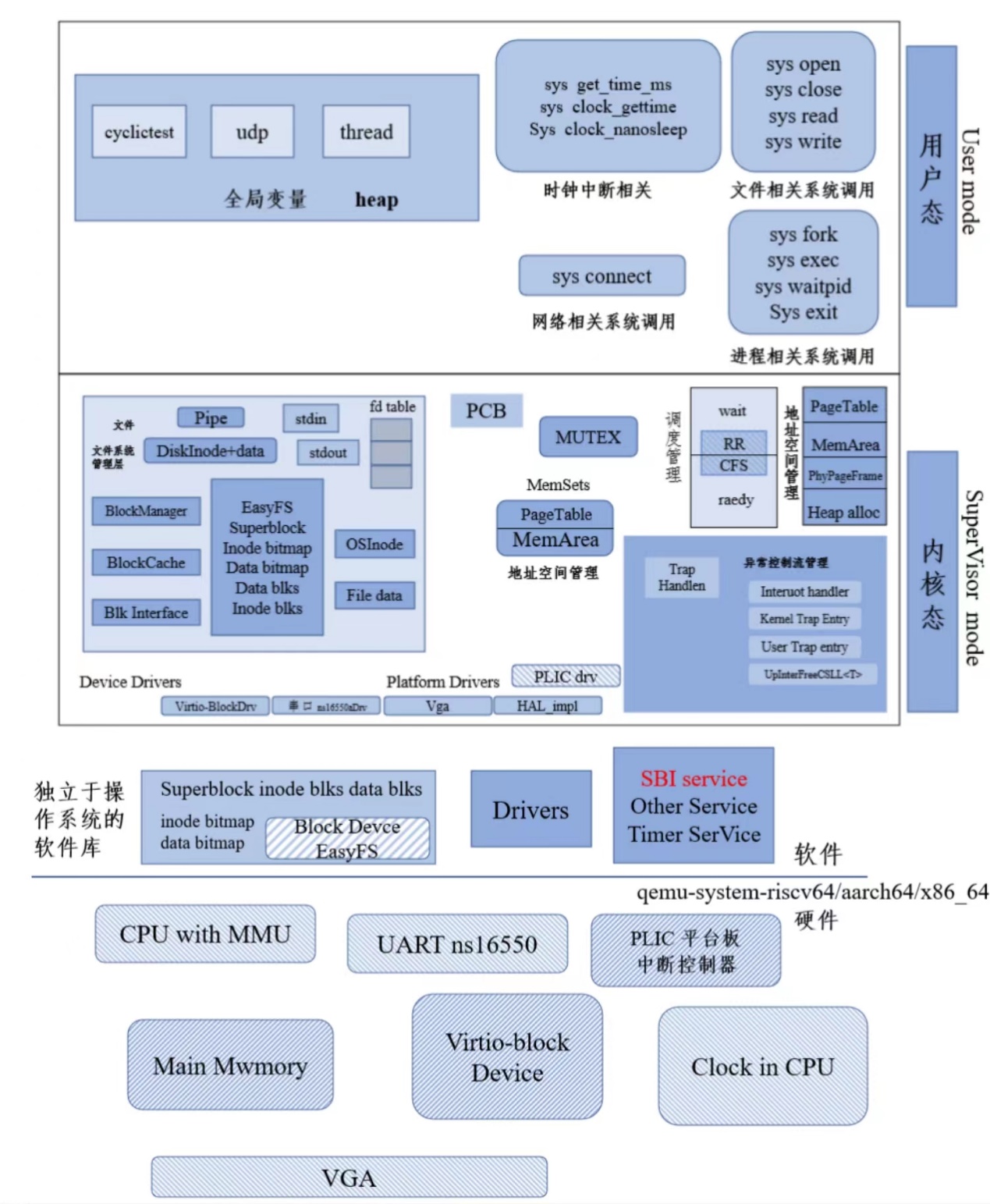


图2-1 NerdOS总体框架

一、 用户态

用户态的程序主要是用于展示操作系统支持的系统调用，以自己编写的shell脚本输出并且运行能够加载程序运行，其中最主要是系统实时性测试软件cyclictest。这是C语言编写的程序将其编译后形成的二进制文件加载到内核运行,在真实物理机上的测试得到的结果是满足实时性要求，在和外界通讯上本设计支持了用户态的UDP测试，本设计在本地主机编写Python文件进行端口监听，在信息成功发送后会在不同的终端中显示，现在将主要的过程说明为如下几点：

（一） 通讯链接建立

网络通讯的基础需要建立在成功链接的基础上，如图2-2所示，在Python编写的程序上开启6200端口的监听，在QEMU模拟的虚拟机上创建一个基于virtio的虚拟网络设备之后连接到net0的网络设备上，配置主机的UDP端口转发到虚拟机的UDP端口2000使得可以正常通讯。经过反复的测试和验证，在本地链接的建立中能够正常运行。

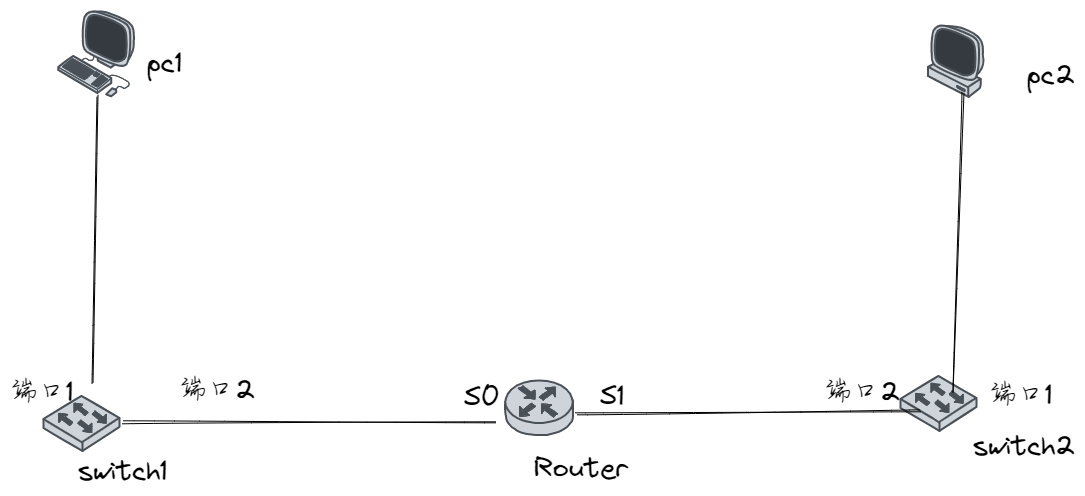


图2-2 网络通讯链接建立

（二） 网络协议支持

在网络数据通信过程中网络协议发挥着重要的作用，因为它的存在，数据传输才能安全可靠并且正确的在两端进行通信。具体来说网络协议的作用包括：（1）确定数据的传输格式：这样约定俗成的格式确保了数据的安全传输和校验，采用统一约定好的编码方式和校验位方便解码和校验。（2）管理网络通讯流程：在建立连接传输数据的过程中规范过程包括数据传输和扰动排除。（3）差错控制：采用一定的纠错协议例如CRC校验和海明码校验等等保证数据的准确性。（4）确保数据安全。具体的分层协议如图2-3所示，在本设计中综合考查了很多开源网络协议栈和相关的标准。其中重点关注了Rust编写的网络协议，其中就有知名的smoltcp协议，该 协议专注于实时性要求很高的嵌入式操作系统，本身代码量比较小巧，在可移植性和可扩展性上都有很高的研究学习价值，这很符合项目一开始的设计思路，在实践中基于smoltcp定制的网络协议进行迭代开发，后面考虑到时间和编码量后采用了杨金博工程师的简易网络协议版本，在此基础上进行了代码适配系统的改进。

（三） I/O设备支持

为了高效访问外设[10]（网卡）来实现数据通信，操作系统必须要具备强大管理和访问效率。这决定了在直观上对于操作系统快慢的判断。为此首先需要在QEMU-VIRT文件中设置好MMIO内存起始地址，并且为了能够完成外设相关的中断处理流程，设计需要扩展之前的异常处理，得益于之前统一封装好的中断接口，因此借鉴和使用UpInterFreeCell<T>接口，这样的设计是为了在对可写数据访问时能够有效的屏蔽中断，在写数据结束后再使能中断，并且将三种架构的中断控制统一为两个调用接口enable\_irqs()使能中断接口和irqs\_diabled()屏蔽中断接口。保证了数据的独占访问，避免了可能发生的数据竞态报错。同时增加了内核层的平台级中断管理程序PLIC以及相关的驱动库，添加了virtio\_net驱动，在文件系统调用方面也做出了改进，总之这些努力使得网络通信是安全可靠的且在操作系统的层面是可控的。

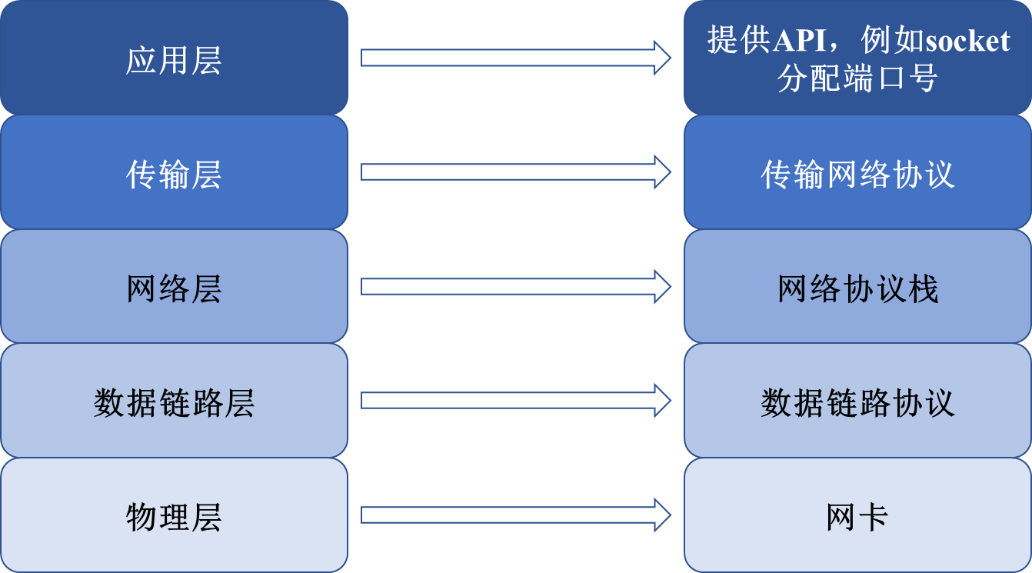


图2-3 TCP/IP网络协议参考模型

二、 内核态

（一） 进程调度

进程调度的本质是对于CPU资源的占用，因此本设计研究调度算法的本质就是要能够高效的利用到CPU资源。现代操作系统基本上都是多任务的运行环境，但是很多时候由于设计编程时提供的抽象机制会使任务认为自己是独占CPU运行的，而一个好的调度算法其本质上是会完全改变一个操作系统的效率。自操作系统诞生以来，进程调度算法自然受到了学界和工业界的高度重视，目前常用的调度算法有FIFO、时间片轮转、多级反馈调度和完全公平调度等。他们都有各自的用途和优势。在本设计构想之初，大量参考选题给出的文档和相应C语言实现代码，对于调度算法[9]的选取最初基于的项目是时间片轮转调度RR（Round-Robin），其效率和表现是够用的，但是更希望能探索一些新的优化方向，于是参考了大量开源项目和博客，最终选择采用Rust编写CFS(Completely Fair Scheduler)。这是目前Linux内置的一种调度算法，在Linux2.6.23版本中作为默认的调度器。它的主要思想是公平地分配CPU资源使得理想情况下的任务能够高效的分配CPU资源。采用Rust写这一个任务调度的探索是艰难的，一方面是语言的安全性决定了用Rust编写一般的数据结构就比较困难，更不用说CFS调度需要用到红黑树这样的高级平衡二叉树；另一方面算法本身需要深入硬件底层去掌控CPU的带宽依次保证每个任务公平的使用CPU的资源。目前探索取得了一点成绩但是仍有可以改进的空间，这其实也是本系统设计的初衷，既做一点有开创性的工作，为开源社区贡献一下智慧，也为操作系统的一些可能改进的方向进行探索。

（二） 内存管理

实时操作系统和分时操作系统（如Linux）之间的区别关键在于它们对于内存管理的设计，由于对实时响应的快速处理，要求实时操作系统（RTOS）必须设计一套快速高效的内存分配释放机制。这意味着本设计要保证任务内存不被改变，以提高系统的响应速度。本设计在内存分配模块中既沿用了Rcore内存分配机制的基础，也实现了相应的改进，保证了实时性的要求。总体上来说，为保证实时操作系统在特定应用场景中的高性能表现，必须要在有限的内存上进行合理的分配和使用。同时为了支持外设I/O设备还需要在MMIO上进行地址的映射操作，保证相关设备的正常运行。

三、 独立于操作系统的软件库

（一） 文件系统

在文件系统的选用上本设计需要考虑实时操作系统的设计需求和应用场景，为此本设计的操作系统需要具备：1）数据一致性：也就是不易失性，保证断电后数据的安全。2）快速响应[11]：应和实时操作系统的设计目标一致保证快速响应。3）内存占用：实时操作系统大多用于嵌入式设备，那么它的内存大小就不可能很大本设计目前的内核设计也就128M大小因此文件系统的内存占用不能太大。4）可靠性：这是文件系统通用的要求必须保证数据是可靠的。目前常见的实时操作系统文件系统有FAT、YAFFS、EFSL、EFS等。综合考虑后本设计选取了Easy FS作为本设计的文件系统支持，它满足上述的要求并且也有实践的先例，经过修改和测试，目前文件系统测试满足预期目标。

（二） 驱动和硬件抽象层

这一部分的实现主要是为了支持网卡的实现，为此本设计必须实现一个可以检测硬件和选用合适的硬件驱动程序的设计，以保证通信的正常工作。为了实现这一目标，本文借鉴了一些成功的设计，实现了硬件抽象层（HardWare Abstraction Layer，即HAL）为不同的设备实现统一的接口，方便内核直接管理和调用。当在QEMU上虚拟化一个网卡设备后硬件抽象层会选择合适的驱动程序进行交互（当然这也要自己实现网路驱动相关程序），为此需要在了解相关的设备规范上进行学习和设计，其中包括正式的串口设备NS16550和自己虚拟的Virtio设备，它们和CPU交互的方式是需要关注的重点。在成功的学习和了解这些支持后，本设计不仅实现了网络驱动设备，还增加了Virtio-blk块设备的驱动。

1. NerdOS内核系统设计
2. 使用相关技术介绍

一、 Rust介绍

作为本文多次提及和介绍的编程语言，本节将它的优点结合代码展示出来。结合斯坦福CS110L课程上的一些介绍和许多学者论文以及工程师的评价，总结得出Rust语言的一些优点。

Rust 是一门注重性能、可靠性和生产力的系统级编程语言，这也是Rust语言核心的特性之一[12]，它借鉴了C++语言的RAII（Resource Acquisition Is Initialization，也即资源获取初始化），这样的好处在于局部对象的生命周期有操作系统管理，无需人工干预。Rust在此基础上又引入了核心机制——所有权系统，如图3-1所示。

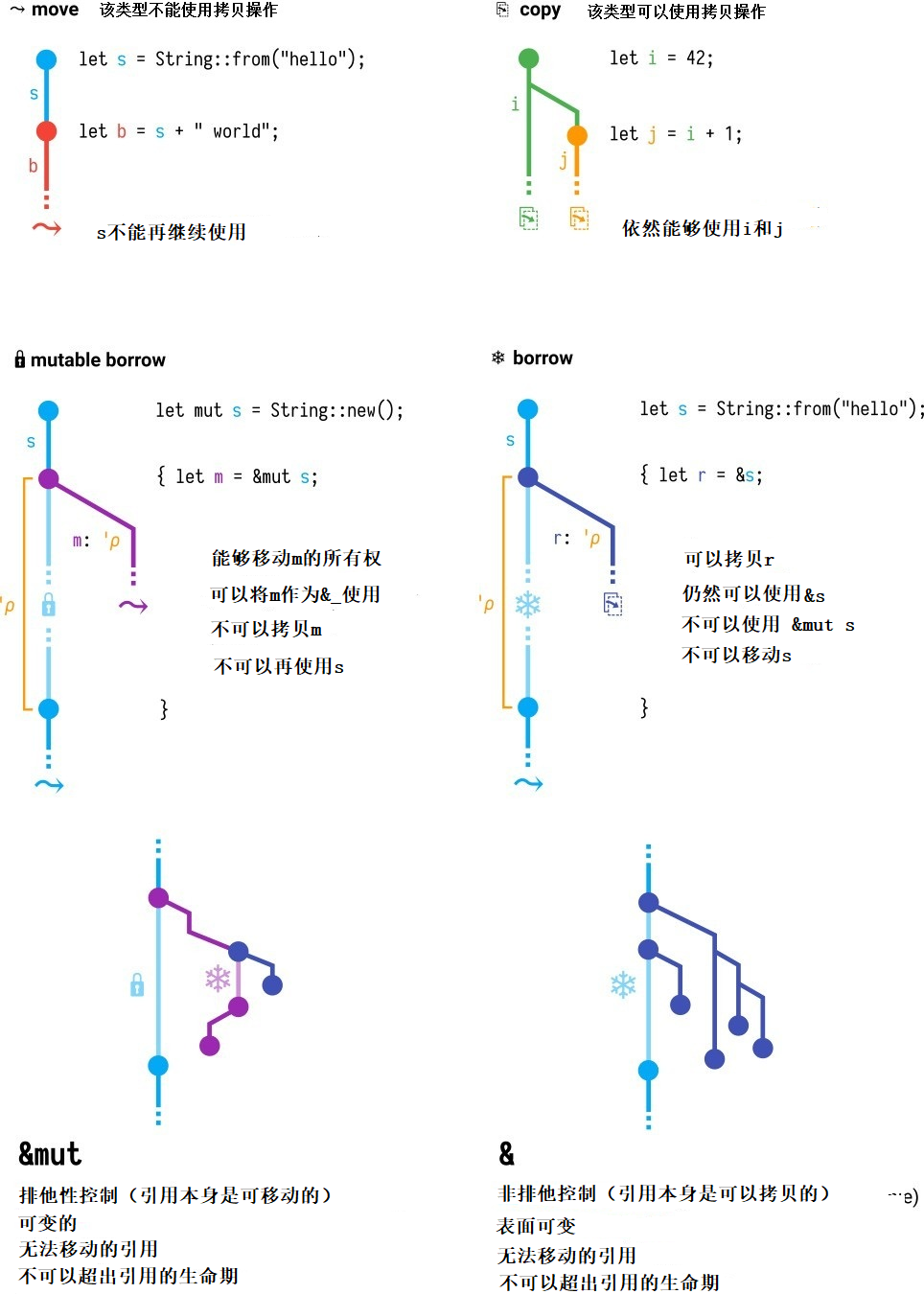


图3-1 Rust所有权系统示意图

上述机制保证了编译器在编译时按照约定的规定进行检查，这样的检查只会存在于编译期，在运行时不会产生性能损失，在此开发者要牢记三条核心原则[3]：

（1）Rust中每一个值被一个变量持有，即该变量是值的所有者；

（2）一个值只能被一个变量持有；

（3）变量离开作用域范围时会被丢弃（drop）。

根据上面的原则，以图3-1的第一个程序为例，声明的s变量绑定了字符串“hello”，但是后续因为b是新变量，所以没有字符串“hello”的所有权。当采用字符串连接的方法编程时，必然会因为没有所有权而报错。其余的五个样例分别阐释了在执行拷贝操作时，可变借用和不可变借用，以及可变引用和不可变引用，这些机制解决了单线程情况下的内存安全问题。但是在多线程情况下，由于多线程的原子性、可见性和有序性等特点，导致情况会更加的复杂，具体表现为：（1）原子性：操作在CPU执行的过程中因为中断乱序；（2）可见性：一个变量在一个线程中的更改对其他线程来说是不可见的；（3）有序性：程序乱序执行。为了解决这些问题Rust引入了Sync和Send特性。这样的实现保证了线程间的安全传递[13]。Rust语言在安全性上所做的努力如表3-1所示[10]。

表3-1 Rust语言在内存安全和线程安全的机制

|  |  |
| --- | --- |
| 安全问题 | Rust相应机制 |
| 空指针 | 不存在未初始化的引用 |
| 悬垂指针 | 生命周期保证 |
| Double free | 仅在生命周期结束后释放内存 |
| 越界访问 | 边界检查 |
| 数据竞争 | Send和Sync特征 |

二、GRUB启动技术

一般操作系统的启动流程可分为BIOS阶段、MBR阶段、GRUB阶段、Kernel阶段以及Init五个阶段，如图3-2 所示。

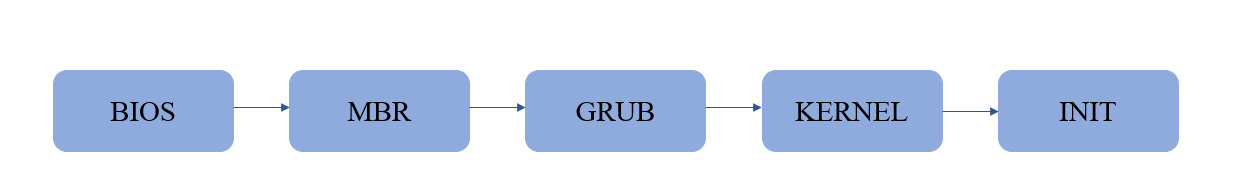


图3-2开机启动流程

第一阶段BIOS（Basic Input/Output System)阶段，这个阶段主要是进行开机自检，计算机上电的时候，会去读取事先约定好的区域检查CPU和内存，以及计算机的基本组成单元和其他硬件，之后引导和进入第二阶段。

第二个阶段MBR（Master Boot Record）阶段，这个阶段的引导记录存在于磁盘第一个分区512bytes的区域之后引导进行。

第三阶段GRUB（Ground Unified BootLoader）阶段，用于识别和挂载文件系统，将Kernel加载到内存，然后将控制权交到第四阶段。

第四阶段Kernel内核阶段，该阶段将解压缩的内核文件交给用户态Init阶段。

本设计主要是在GRUB阶段对原有内核的引导启动环节书写规范化文件。GRUB的启动需要在文件路径中找到MultiBOOT的位置， MultiBOOT的主要功能是允许在一个计算机上启动多个操作系统，这样的规范也是GRUB启动所需要的。由于之前是在拥有串口的真实物理机启动的测试，但是这样的机器不是随手可得的，所以在尝试制作的过程中遇到了无法显示输出等问题，这也是本设计的一个难点。在贾越凯博士的指导下，将串口输出换成VGA输出后，系统成功将GRUB启动完成。这里简要说明下GRUB启动的步骤：（1）BIOS的自动启动运行。（2）引导扇区查找GRUB程序之后加载到内存。（3）加载GRUB.cfg文件，需要将之前的Multiboot和镜像文件放置在菜单项中一并启动。（4）显示之前制作的菜单。（5）加载内核之后就会将整个的内核镜像文件加载到内存。（6）启动操作系统内核这部分开始将控制权转交到内核，之后操作系统就会开始运行QEMU仿真，并且加载预先设计和编译完成的shell脚本交互程序。

三、 可适配的一些开发板

实时操作系统的设计本身就是为了支持一些嵌入式设备而设计的。为此，本文验证了一些经典的开发板的可适配性。事实上，仅需要更换一些目标平台的引导固件就能适配，对于常见的x86和ARM开发板不再赘述，本节主要介绍近年来的业界新贵——RISC-V开发板。

（一）RISC-V指令集

目前x86和ARM指令集收费问题导致深入的指令集研究和改进需要另辟蹊径，而指令集决定了CPU的架构和设计，这样的核心技术自然需要牢牢地掌握在自己手里。目前国内自主设计的LoongArch取得了比较重大进展，但是RISC-V一直得到了大家无限的关注，其原因就在于它的开源特性。RISC-V指令集自2010年诞生于伯克利分校后，一直可以被任何人出于任何目的而设计、制造和销售，其开源属性以及它自带的精简指令集设计的优良特性，使得它受到了广泛的欢迎和工业落地。2020年为了规避中美贸易制裁，RISC-V基金会搬迁到瑞士并且更名为RISC-V国际，继续开源非盈利性质工作。正是由于这些优势使得RISC-V的发展取得了可喜的成绩。国内开源项目有阿里巴巴在2021年发布的玄铁910，其搭载了LLVM编译器。此外，中科院香山、西部数据SweRV等也陆续发布开源版本。在2023年3月26日OS2ATC十周年大会上， 发布了RISC-V笔记本ROMA，可见目前它在生态上的成熟已经从嵌入式设备向高性能的桌面服务端迈进。在发布会结束后，尽管存在一些问题但是总体来讲依旧是回应了社会的质疑。“两岸猿声啼不住，轻舟已过万重山”RISC-V桌面服务端有了一定的突破。

（二）RISC-V硬件开发板

HiFive Unmatched是全国大学生系统能力培养赛支持的开发板。第一款开发板HiFive Unmatched如图3-3所示。这款开发板性能强悍，搭载四个SiFive U74的主核和一个SiFive s7小核内存方面采用的是16GB的DDR4，以及32MB的闪存。

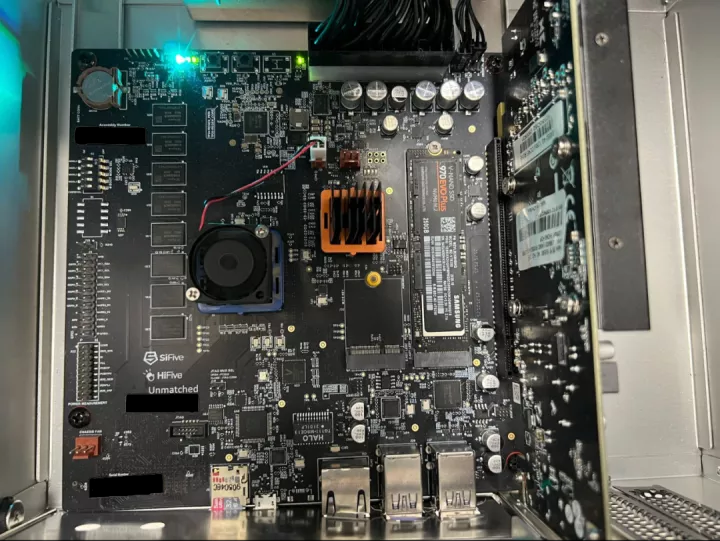


图3-3 HiFive Unmatched 开发板

HiFive Unmatched作为目前RISC-V架构的性能最为强悍的开发板，在2022年被大赛组委会作为远程调试的支持开发板，参赛队员在其中学到了很多的相关知识。目前由华中科技大学蒋周奇同学在GitHub进行了引导环境的支持，因此只需要更换引导和一些相关参数设置就可在上面运行。但是目前的设计对于这样的硬件需求无疑是大材小用，其强大的生态环境已经可以支持很多天马行空的想法。HiFive Unmatched目前可以完整支持Linux这样庞大的系统运行，在许多方面都是业界开发和探索的重点。唯一遗憾的是价格不是很亲民，目前售价为665美元。

Kendryte K210开发板也是系统赛成功运行内核测试样例后免费发放给参赛队的一款开发板。作为2022年的参赛队员，对该开发板有一定的了解和开发经验，故将其作为本设计的首选开发板。Kendryte K210搭载的是双核64位处理器，具备优良的工作性能和算力，且价格亲民，目前开发套件价格在170-700元人民币左右的区间，其生态建设比较完善，其系统架构如图3-4所示。

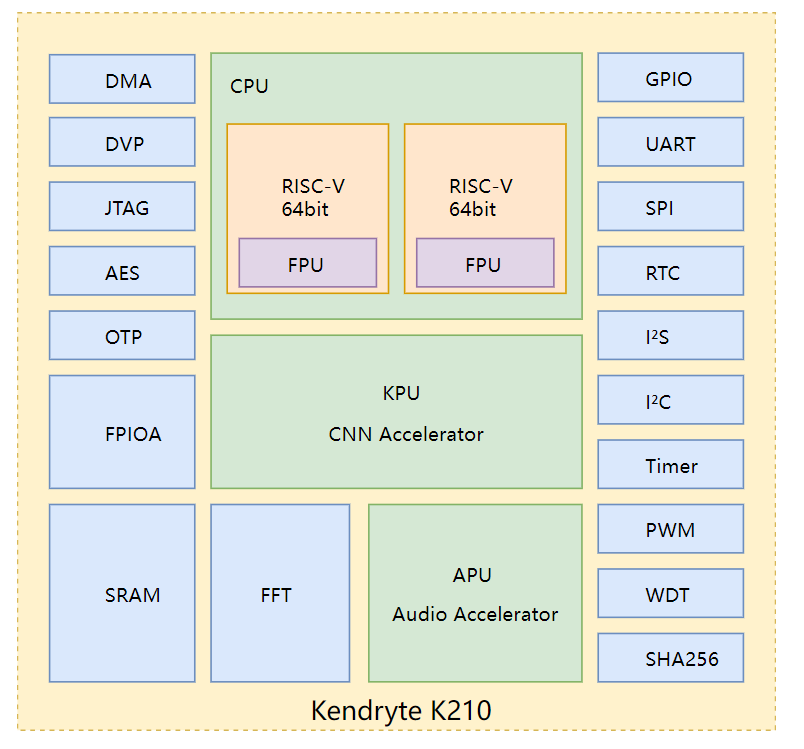


图3-3 Kendryte K210系统架构

本系统开发主要是将文件系统镜像拷贝到内存卡，插入K210后将开发板连接到PC，之后依赖于K210的串口通信（在此之前需要安装Python的串口通行库以及简单的串口终端）实现在K210上运行制作好的操作系统。

四、 内核裁剪以及组件化技术

上文所提到的开源社区的几款主流实时操作系统大多具备内核裁剪功能，被广泛的用于物联网设备。其核心在于根据具体的场景修改或者删除不必要的模块和功能，进而可以根据需要的场景定制化设置，以适配最合适的解决方案。

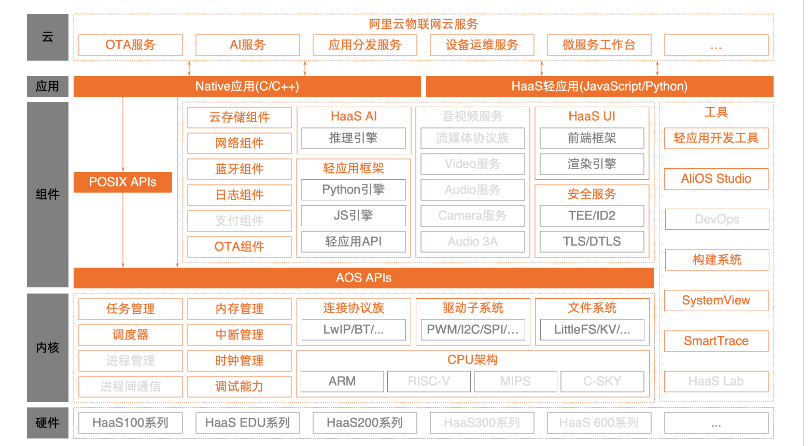


图3-5 AliOS Things架构总览

以AliOS Things为例，内核十分简单仅仅包含了一些基本的框架（如图3-5），核心在于裁剪搭配所需要的组件，例如搭配蓝牙组件、网络组件以及相关的开发套件。在很多嵌入式领域对于内核的裁剪都是很常见的，这类工作需要设计师了解需求之后，根据不同的模组配置内核，通过一些列的技术安装和启动内核，这类技术不仅可以定制化内核，还可以大大的提高内核的安全性。

关于内核组件化技术以及相关的研究，最具代表性的是清华大学陈渝老师[14]等一系列学者们的思考和实践。重要的时间节点是在2022年12月12日，Linux6.1版本发布，将之前的R4L(Rust For Linux)子系统加入，这是Rust在Linux上的重要突破。基于此，目前研究的主要设想是用Rust开发不同的内核模块，这些技术将主要用于Rust实时优化的Linux内核中，也是Rust for RTOS的重要新技术方向。陈瑜老师组织的研究学习小组每周分享相关技术进展，关于如何快速启用Linux Rust Feature并运行测试，分析Rust在Linux 6.x内核里的用法，内核中Rust与C的ABI兼容性与互操作系统性的研究，为linux其他子模块封装rust api，用rust重写子模块，完善Rust-for-Linux并提交补丁，实时操作系统（Linux RT-Preempt、RTOS等）等关键技术研究。

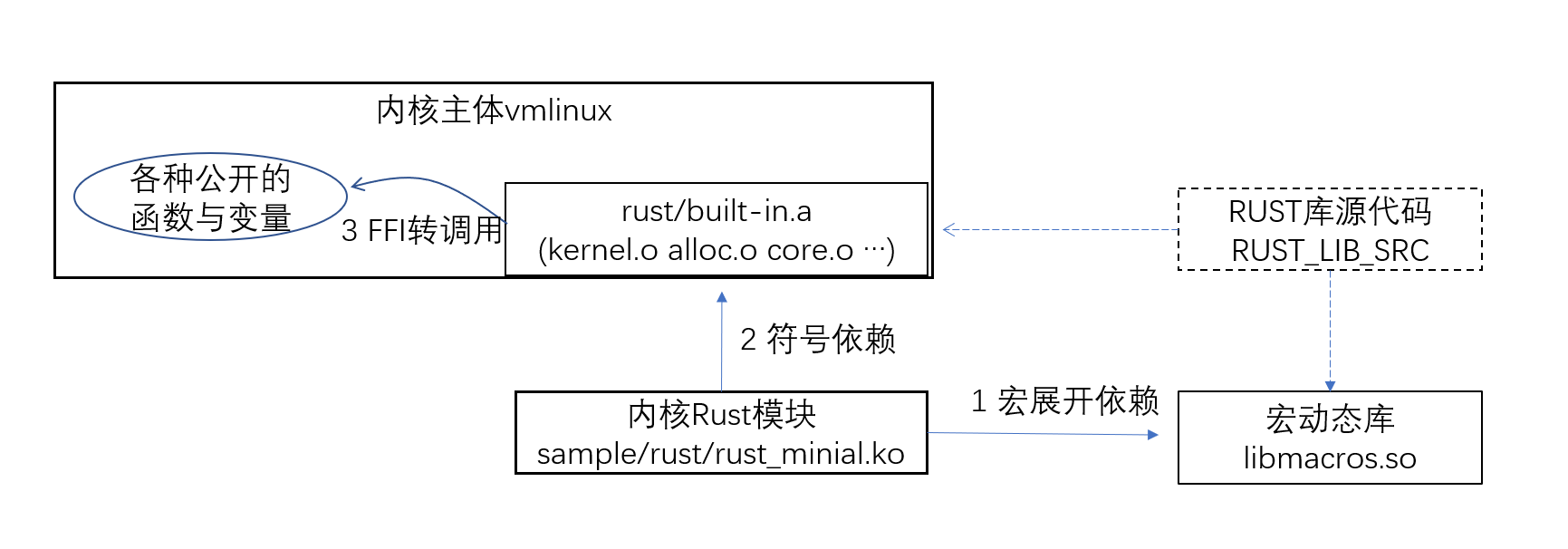


图3-4 内核与模块的关系

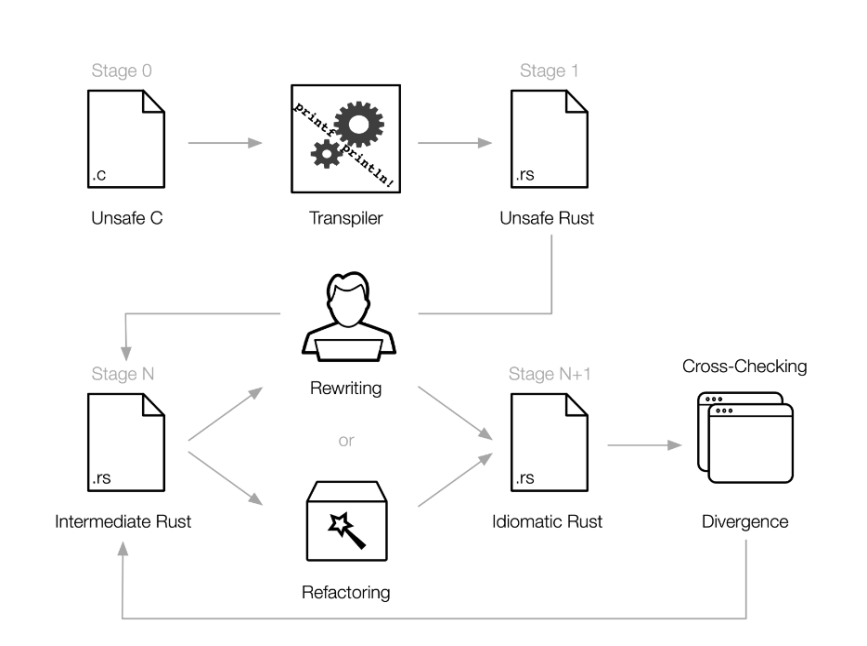


图3-5 C2Rust流程

本文以乾云启创工程师石磊的报告为主要内容，结合其他同学的探索和报告介绍一些相关工作的进度。首先是R4L(Rust For Linux) 的机制——内核于模块的关系如图3-6所示。由于C语言和Rust高效的绑定特性因此可以很容易的实现FFI转调用机制，具体的流程如图3-7所示。

1. 调度算法数据结构的设计

一、 数据结构选用

为了支持操作系统中的相应功能，NerdOS需要多种数据结构。值得一提的是，内核编程与直接调用相应库实现的数据结构完全不同。由于Rust的语言特性，使得实现数据结构也成为了一个枷锁，用Rust实现一个常见的数据结构和其他语言相比难度完全是不在一个量级。而很多其他的语言或者应用场景中却可以大量使用已经封装好的数据结构的库，如知名的C++语言提供的STL库。而在内核编程中，开发者不可以使用一些已经完善封装的标准库。同时还要对性能和内存对齐等的难点进行考量和改进，才可以顺利的调用。本设计需要使用到队列和红黑树等数据结构，在调度算法中这些数据结构的应用十分广泛，这些在其他语言可能有更简单的实现，但是这里却要付出更多的努力。

二、 二叉搜索树介绍

首先在调度算法中，红黑树是CFS(Completely Fair Scheduler)中必须要用到的数据结构，它自身的设计保证了CFS算法能够及时且高效的找出最小值，同时在数据的更新中，本身的拓扑结构变化也比较小。结合本系统的实现，下面将详细介绍几种二叉搜索树的特点和应用场景。

首先是需要使用到的红黑树这一经典的数据结构，由于节点的颜色而得名，它的一系列规则是为了实现其平衡二叉树插入或删除操作后，拓扑结构的性能在*O*(1)时间之内就能完成的特点而做出的规定。尽管有整体拓扑结构随着数据操作变化很小的优点，但是由于数据量大的时候红黑树本质上仅仅只是一棵2-4B树，所以树高也会很高，因此不适合做IO操作。而B+树是B树的一个变种，由于它的设计采用的是叶子节点用来存放数据，因此在数据库领域有着很高的适配。同时在检索数据和IO交换上有很强的优势。而LSM树是一种比较异类的树，是因为在一些场景对IO处理进行了优化，因此在数据库方面的性能表现更胜一筹。

本设计所用的红黑树在性能上也做了一些优化，这个由Rust编写的项目来源于Solana项目的开源no\_std红黑树实现，由于其在性能上做了一点的补偿，使得它和BtreeMap在性能上相比也有了一定的优势。由表3-1中数据可知红黑树在访问和插入上慢两到三倍，但是在访问速度上比较快。

表3-1 红黑树和BtreeMap性能测试

|  |  |  |
| --- | --- | --- |
|  | BtreeMap | RBTree |
| 反序列化10个元素 | 472ns | 13ns |
| 反序列化1280个元素 | 109000ns | 13ns |
| 访问10个元素树中的元素 | 10ns | 23ns |
| 访问1280个元素树中的元素 | 19ns | 33ns |
| 插入10个元素树中的元素 | 78ns | 147ns |
| 插入1280个元素树中的元素 | 106ns | 239ns |

三、 数据结构操作

首先，数据结构操作提供了两种类型的选择，一种是需要使用内存大小已知的红黑树，则直接采用RBTree。初始化代码如下：

|  |
| --- |
| use slice\_rbtree::tree::{tree\_size, RBTree, TreeParams};  // RBTree 要求输入切片具有适当的大小  // `RBTree` 中的每个节点都有一个在编译时已知的固定大小，  // 所以要估计这个大小 `KSIZE` 和 `VSIZE` 参数应该传递给 tree\_size  let size = tree\_size(  TreeParams {  k\_size: 50,  v\_size: 50,  },  10,  );  let mut buffer = vec![0; size];  let mut movie\_reviews: RBTree<String, String, 50, 50> =  RBTree::init\_slice(&mut buffer).unwrap(); |

从注释中可以看出，在编译时知道内存大小的情况下采用RBTree数据结构。程序员通过手动传入合适的参数完成红黑树的申请，在使用的时候需要先申请一块切片的内存之后再使用。而如果需要使用一个编译时不知道内存大小的红黑树，那么就需要使用到RBforest数据结构。它和上面结构体声明不同的地方是多了一个树的数目声明参数，可以预先开辟一块足够大的空间。

|  |
| --- |
| use slice\_rbtree::forest::{forest\_size, ForestParams, RBForest};  // RBForest 要求输入切片具有适当的大小  let size = forest\_size(  ForestParams {  k\_size: 50,  v\_size: 50,  max\_roots: 2,  },  10, // 所需的节点数  );  let mut buffer = vec![0; size];  // `String` 类型具有可变长度，但我们必须选择一些固定的最大长度（键和值均为 50 字节）  let mut reviews: RBForest<String, String, 50, 50> =  RBForest::init\_slice(&mut buffer, 2).unwrap();  reviews.insert(0, "Office Space".to\_string(), "Deals with real issues in the workplace.".to\_string());  reviews.insert(0, "Pulp Fiction".to\_string(), "Masterpiece.".to\_string());  reviews.insert(0, "The Godfather".to\_string(), "Very enjoyable.".to\_string());  reviews.insert(0, "The Blues Brothers".to\_string(), "Eye lyked it a lot.".to\_string()); |

1. NerdOS系统内核实现
2. 开发环境

一、 运行环境

NerdOS的开发在Ubuntu22.04的虚拟机云服务器上进行，主体采用Rust语言完成主要逻辑的书写，版本采用nightly-2022-11-03并且锁定，并且在汇编逻辑处理方面有少量的不同架构的代码，总体上是x86的汇编居多，也有RISC-V和AArch64的部分汇编代码，构建编译方面主要是用Makefile书写，QEMU版本为7.0发行版本。

二、 测试软件

内核编程方面的测试需要一个能够全程跟踪各种细节的软件，正因如此，它才能够保证开发者得到的信息是准确的。GDB这款工具在内核调试过程中发挥了极大的用途。由于内核调试能用到的工具并不多，GDB作为一款优秀的debug工具在开发设计中起到了很大的作用。搭配上开源的GDB DasBoard插件，可以使得开发者调试面板可视化效果更强，更方便与开发者进行交互，效果如图4-1所示。



图4-1 GDB DashBoard插件效果图

1. 各模块系统实现

在第二章中介绍了系统架构的总体设计，本节将对每一个具体的模块进行详细介绍。

一、 用户程序测试模块

用户程序测试模块主要是由Rust和C语言编写的程序，测试内核的系统调用和功能是否完善。操作系统实时性测试的用途主要是为了测试内核的时延，这有助于判断内核的实时性是否满足要求，它是一个开源测试项目。其原理为：测试时延本身就是中断延时和调度延时，所以需要模拟出这样的场景，因此这个测试的源码会在运行一个普通进程之后，启动一个指定优先级的实时进程，这个指定的进程会有一个计时器，当计时器溢出后会触发中断和调度其他进程，这时获取到的实时进程从切换到被唤醒的时间就是实时操作系统测试需要的延时时间。参照官网给出的测试参数说明，开发者可以得到实时操作系统允许的时延，如图4-2所示。

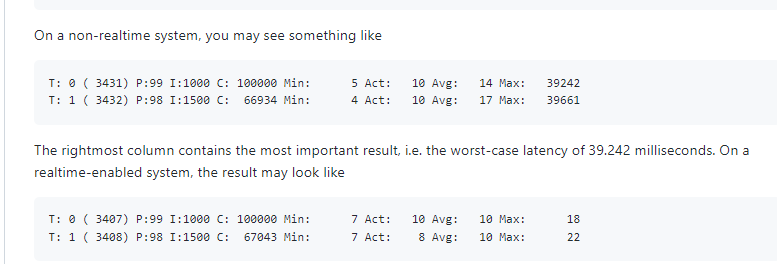


图4-2 系统实时性测试参数说明

max参数是实时性能指标[15]，在QEMU模拟的机器上不同的架构有不同的结果，最优秀的性能测试结果是x86上的输出大概会在500到600之间，经过和贾越凯博士的沟通在真实的物理机并且带串口输出的值会在10-20之间满足实时性场景的要求。并且为了规范化启动，NerdOS进行GRUB启动设计，这也是为了能够适用于各类测试场景，为了更好地显示，兼容了VGA输出的支持。目前的测试结果有优化的空间，但是可以肯定的是本系统的设计是满足实时测试的标准的。

其他的用户程序包括了进程相关的fork和exit调用，以及输出字符和时钟中断计时测试，这是为了测试一个操作系统最基本的系统调用而设计的。需要说明的是，UDP的测试支持主要是为了使用UDP协议进行的connect系统调用以及之前上文提到的驱动和HAL等是否符合设计规范。经过本地ping命令多次反复的测试均可以成功实现互联和消息的发送。

二、 内核模块

在内核模块中将重点介绍不同架构的细节以及系统实现时的规范统一。首先是中断模块，中断是一种内核中异常的控制流，一般是指外设I/O设备请求内核服务打断当前执行的控制流。其次是串口模块的统一设计和封装，UART（Universal Asynchronous Receiver/Transmitter，异步收发传输器）用于实现异步数据的传输，本设计以一些常用的串口协议如PL011和16550为基础，实现更好的通用性和兼容性。在网络通信模块中，考察了目前的一些Rust书写的网络协议，并且在相应的内核适配上进行了一些修改和测试，对开源社区及爱好者具有一定的参考价值。

（一） 中断模块

x86的中断管理系统设计：参考现代x86系统的管理设计，即同时兼有APIC（Advanced Programmable Interrupt Controller，高级可编程中断控制器）和i8259（Intel 8259A，可编程中断控制器）中断管理设计。在操作系统的发展中，后者i8259渐渐被淘汰，但是为了拥有向前兼容的能力[16]，目前i8259主要是负责传统的8个外部设备设计的中断，而APIC主要用来处理更加高级的中断信号，比如处理器内部设备和多核的中断请求。本文设计的NerdOS将二者统一封装为一个接口[17]。

ARM的中断管理系统设计：ARM平台的GICv2（Generic Interrupt Controller virtualization 2，通用中断控制器虚拟化版本2）在实现的过程中考虑到系统的性能和可靠性以及参考设计标准，在中断控制器分发寄存器上依次列举和分发[18]。在代码细节上，为了保证规范需要在代码中使用#[allow(non\_snake\_case)]宏保证命名的规范，之后会正确的分配中断信号完成处理。中断控制器和CPU接口寄存器的实现也同样遵照规范，统一封装接口提供给上层调用[19]。

RISC-V的中断管理系统设计：RISC-V的中断管理直接使用的riscv crate模块提供的SIE（Supervisor Interrupt Enable），因此逻辑处理相对比较简单，同样采用统一封装接口提供给上层调用。

这三种架构最后为上层提供中断管理服务，统一封装为enable\_irqs()使能中断和disable\_irqs()关中断两个函数调用接口，在后面的虚拟IO设备设计中，中断请求发挥了很大的作用，保证了数据独占内核的传输。

（二） 串口模块

PL011：PL011是ARM公司早期开发的一种串口芯片，该设计主要用在ARM Cortex-A处理器上。它具备这些特点：1）可适配的传输速率和数据位数，应用场景十分广泛；2）16byte的发送和接收缓冲数据区，传输效率高；3）数据缓存和FIFO机制极大的降低了CPU的负载。这些优良的特点保证了整个实时操作系统的设计为硬件的通信和控制等留下了很好的协议接口。

UART 16550：UART 16550 是由National Semiconductor公司推出的，主要用于早期的PC机和一些嵌入式设备上，它的技术特点和PL011十分相似。但是在最大传输效率上略有不足，PL011支持4Mbps，而UART 16550只能够支持1.5Mbps的最大速率，但是在标准传输效率上均为115.2kbps。目前，UART 16550衍生了许多后续的版本，但是在市场占有率上已经被许多后续新的串口协议所替代。出于学习和Rust实现的便利，本设计采用了这样一个经典的串口协议。

（三） 网络通信模块

网络协议选择：SmolTCP是目前用Rust实现的轻量级TCP/IP协议栈，早期的Rust内核实践项目（如清华大学Rcore）就采用了这一协议，不过在阅读其项目实现过程中，发现他们所考虑支持的状态过多，代码比较庞大。因此，本设计采用了杨金博工程师实现的简易协议栈，使其更符合本设计适用于嵌入式领域代码量小、功能精简的特点。

功能测试程序：实现通信协议和相关的系统调用之后，需要搭配一些程序测试相应的功能，在用户测试程序上设计了一个向本机6200端口发送消息的Rust测试程序，发送的内容为“Hello NerdOS user program”而监听程序使用Python程序进行监听并且成功之后回复。

涉及到的网络设备操作：当今使用者已经很少关注智能设备内存容量和CPU性能，最直观的感受是看一个智能设备的响应是否迅速，显示是否流畅，触摸屏是否迅捷等。其实这些都是操作系统为用户提供的对于外设的管理和访问。同样，网络设备作为I/O设备的一种，为了支持网络通讯，本系统设计需要有两方面的支持。一方面是上文提到的网络协议方面的支持；另一方面是硬件层面上需要实现驱动网卡设备的驱动程序，在这之前要处理好CPU和外设的连接，需要把相关的控制寄存器映射到相应的内存区域（MMIO 即Memory Mapped I/O）。这样CPU就能像访问内存地址那样访问外设的寄存器。其中需要注意的是外设是不是和CPU直接相连，因此需要平台级中断管理控制（Platform-Level Interrupt Controller，PLIC）来收集和处理外设的中断信息，再传递给CPU。但是NerdOS作为一款简洁的实时操作系统，因为适配的外设和中断场景不是很多，所以PLIC就不需要实现。

网卡设备驱动程序的书写，主要是完成以下的工作：(1)定义设备相关的数据结构，包括设备信息、设备状态、设备操作标识符等。(2)设备初始化，包括分配好所需I/O操作的内存，设置好中断处理例程。(3)处理可能产生的中断例程。(4)和上层模块（文件系统这里的实现是easy-fs）交互。(5)完成上层模块的处理需求。此后是硬件设备树的管理，NerdOS沿用的是rcore的virtio实现。至此完成了硬件层面的支持，基本上就完成了虚拟环境上的工作。而其他数据收发工作所需的硬件抽象层（HAL）也一并在virtio中实现。

（四） 调度算法模块

这部分的内容本来应该是属于内核模块的，但是因为其工作量和实现难度比较大所以单独作为一个模块。本设计的重心是完全公平调度（CFS）策略，这部分也是整个系统设计的重要创新之处。利用这一算法可以用于在高频核心上改善程序的性能，CFS 有一个功能似乎与这个解决方案相关：sched\_child\_runs\_first。在创建线程时，此功能会为子线程分配一个较低的 vruntime，使其具有比其父线程更高的优先级。这一调度算法已经是Linux发行版在早期就采用的版本。本身来看已经有很多成熟的版本，就算法来看，相应的调度策略也开源了很多可以参考的项目，但是实现Rust语言编写的版本是十分困难的。目前已经有的参考大多都是用户态模拟的程序，也就是可以使用很多已经实现的标准库的功能。而要在内核中做移植，所面临的挑战主要有：(1) 红黑树书写需要自己分配内存，这或许在其他语言来说不是很难的问题，但是Rust语言的设计决定了在书写这类数据结构时面对的是编译器严苛的检查。(2) CFS调度算法的核心是实现CPU带宽的控制，因此必须要对硬件的细节有足够的了解才能在任务优先级计算中得出较为准确的值。为此，本文进行了很多尝试，在保障系统整体运行的基础上，首先参考用户态开源项目的实现，完善了一个用户态的调度。其次是将所用的标准红黑树换成内核编程状态下的红黑树，这部分工作量庞大，但是最终实现了用户态模拟和内核态红黑树的编写。后续将整合前期工作植入NerdOS的整个调度模块。

1. NerdOS系统内核测试
2. 测试目的和意义

系统测试是指在设计和开发基于Rust的跨平台实时操作系统NerdOS之后，对该系统各个功能和性能进行的一系列测试，以满足用户在实际应用中的各种需要。所谓的系统测试，就是寻找出系统的问题，然后再进行改进，提高用户的信心与好感。如果不做系统测试，那么很有可能会因为系统中的一些漏洞而引发一系列无法预估的问题，让用户与本系统的关系变得疏远。良好的系统测试，有助于完善用户对系统的使用体验，减少系统的问题。系统测试不仅要发现系统中的错误和缺陷，还要对其进行全面地评价，从而改善系统的性能。同时，在测试中也不能一味地追求测试的结果，而忽视了测试的投入。

在内核的开发过程中，Debug是非常重要也是非常困难的一件事，可以依赖的无非就是GDB这样最底层的调试，而且有些BUG是GDB无法发现和定位的。由于特定的执行顺序，一些并发问题可能会突然出现也可能会突然消失，这样对于系统安全性来说是万万不可接受的。然而在现实世界中这样的BUG确实比比皆是。对此，学术界和工业界都给出相应的解决办法，学术界的一种倾向是形式化验证，对所有可能的情况进行穷举，以此找出问题进行对应的修正。而工业界的解决办法主要是采用混沌工程，即随机产生BUG尽可能的覆盖可能性。但是由于这样的不确定性，导致BUG复现和调试修复都是很困难的事，本设计开发过程中尽管采取了在成熟的开源项目基础上进行开发，但是由于并没有文档介绍和环境配置的信息，因此一开始的工作并不是很顺利。即使是后面成功的进行了相应的改动，也出现了借用无法归还等奇怪的BUG。这也是用Rust进行内核开发的常态，想要更少的BUG和更多的可靠性需要和编译器进行更多搏斗。

总而言之，开发者要在实际环境中对程序进行测试，检验各功能要求是否正确，如果出现了与最初设计不符的问题，应立即进行修正，避免出现问题。系统测试既是对系统的功能完成程度的检验，也是一个与用户满意程度密切相关的具体、合理的检测流程。所以，在系统的开发和设计中，系统的测试是一个很重要的环节，如果没有通过测试就直接投入到系统中，那就是对系统的不负责。

1. 测试方法

一、 黑盒测试

黑盒测试又称为功能测试或者数据驱动测试[23]。黑盒测试，是对所有的功能进行测试。在测试时，将程序视为一个无法开启的黑匣子，只是在程序接口上进行测试，而不会考虑程序的内部构造与内在特征。该测试方法只会检验程序的功能是否符合要求规范的要求，程序能否正确地接受输入参数，并产生正确的输出结果[24]。黑盒侧重于程序的外在构造，而非内在的逻辑架构，其重点是软件的接口与功能。

二、 白盒测试

白盒测试也被称为结构测试、透明盒子测试、逻辑驱动测试或者代码测试[25]。“白盒”是一种测试案例设计的方式，“盒”是指被测试的软件，“白盒子”是指能看到盒子里有什么，也就是它的工作原理。“白盒”方法充分理解程序的内部逻辑结构，并对所有的逻辑通路进行测试。“白盒”方法是对路径的穷尽。采用这种方法，首先要对程序的内部构造进行检验，然后根据程序的逻辑，得到相应的数据。在整个程序中，单独的通道数目是一个天文数字。

1. 测试内容

一、 编译启动

（1）能否正常启动；

（2）工具链版本是否正常适配；

（3）多种架构的启动是否正常；

（4）GRUB脚本镜像制作是否正常；

二、 内核模块

（1）基准测试用例是否成功；

（2）系统调用是否正常；

（3）系统实时性测试是否满足要求；

（4）网络通信验证小程序是否正常交互；

三、 算法模块

（1）能否正常运行；

（2）是否会出现死锁；

（3）各个任务的运行是否公平合理；

四、 硬件端模块

（1）文件系统镜像拷贝；

（2）开机正常；

（3）串口通信是否正常；

1. 测试结果

一、 编译启动

经测试，在指定的Rust版本和QEMU版本上，本设计采用的OpenSBI引导固件和所用的交叉工具链在编译运行时均正常启动，符合预期。在内核目录下执行指令“make run ARCH=riscv64 LOG=warn”后成功运行如图5-1所示。

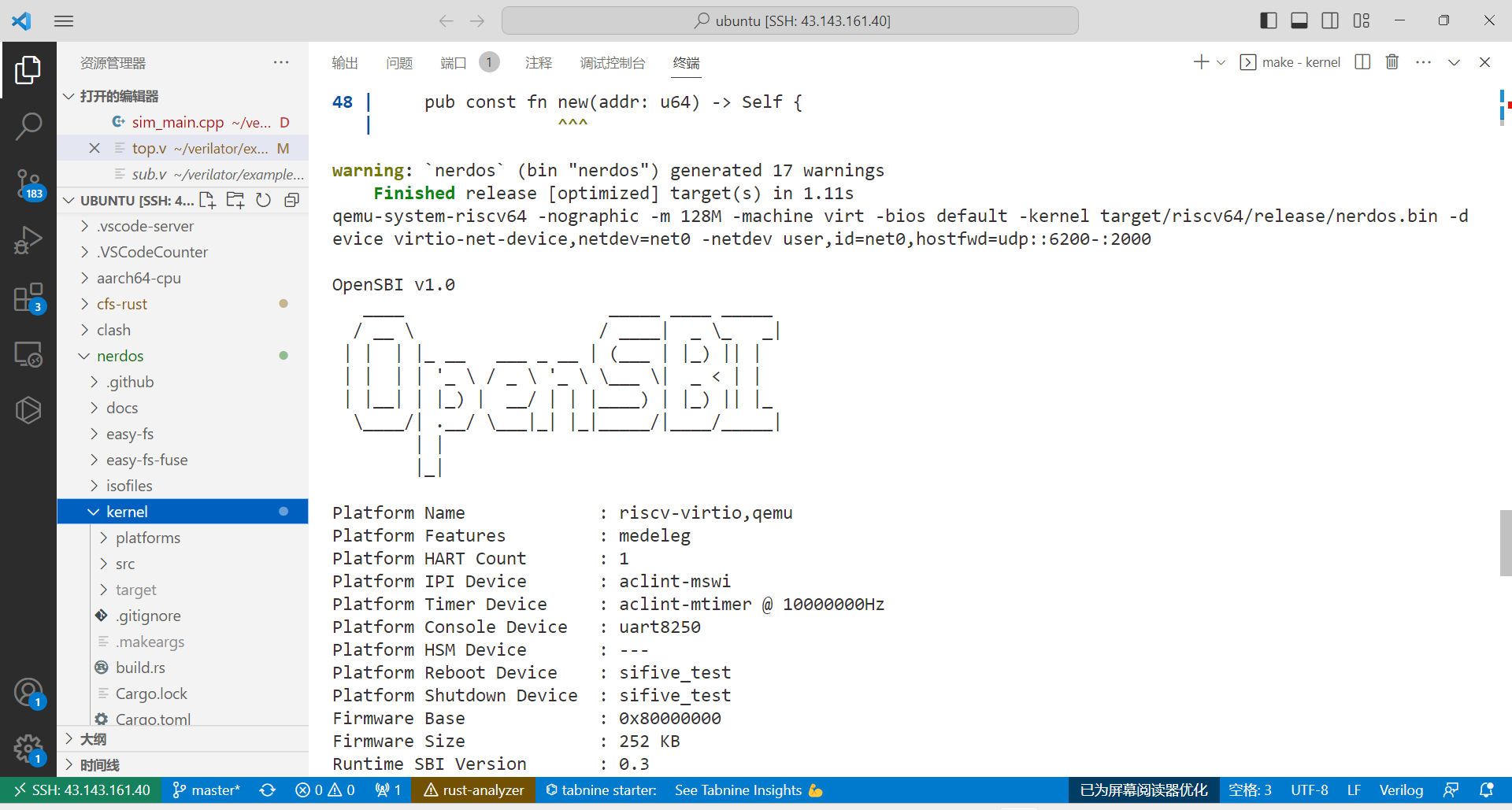


图5-1 开机启动示意图

二、 内核测试模块

经测试，内核的各个基准测试样例均能够正常加载执行，验证程序和实时性测试结果均在正常范围。如图5-2所示

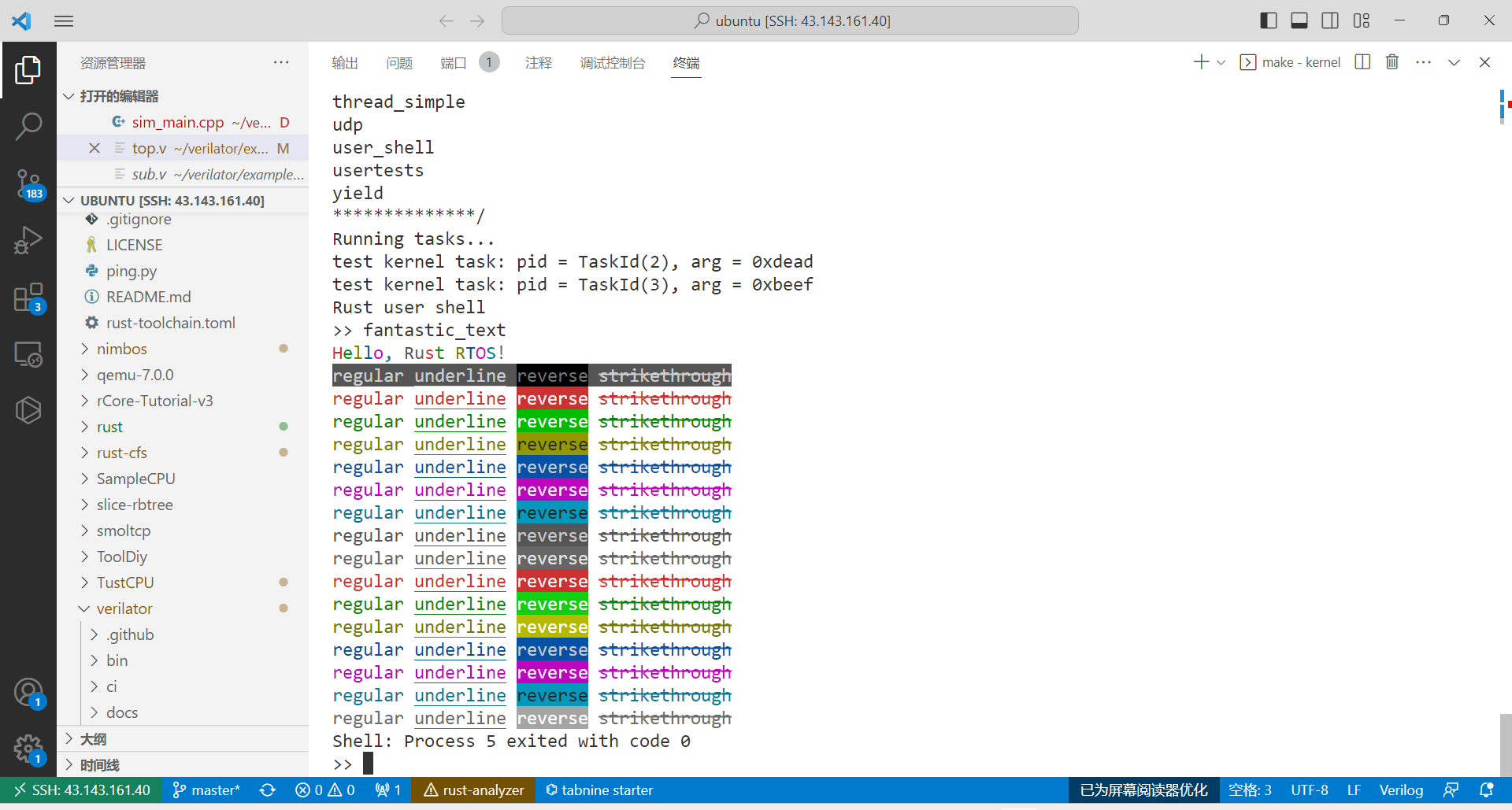


图5-2 基准测试样例示意

三、 算法模块

根据原来的设计，调度算法需要在用户态进行演示，这里的话参考一些开源实现，在Python脚本中尝试测试的任务并且分配CPU运行时，之后存入到文本中，程序会自动打开文本后调度这些任务，经过测试之后用户态的算法正常运行，之后是内核编程no\_std的红黑树也正常移植，目前主要是整合到内核模块的过程比较复杂还有待改进。

四、 硬件模块

经过测试，k210板子可以正常上电，文件系统设计也是没有问题的。在添加引导程序和更改Makefile文件后可以正常编译二进制文件，用开发板官网给出的烧录工具，在和开发板成功连接后可以正常烧写运行操作系统如图5-3所示。



图5-3 硬件平台运行截图

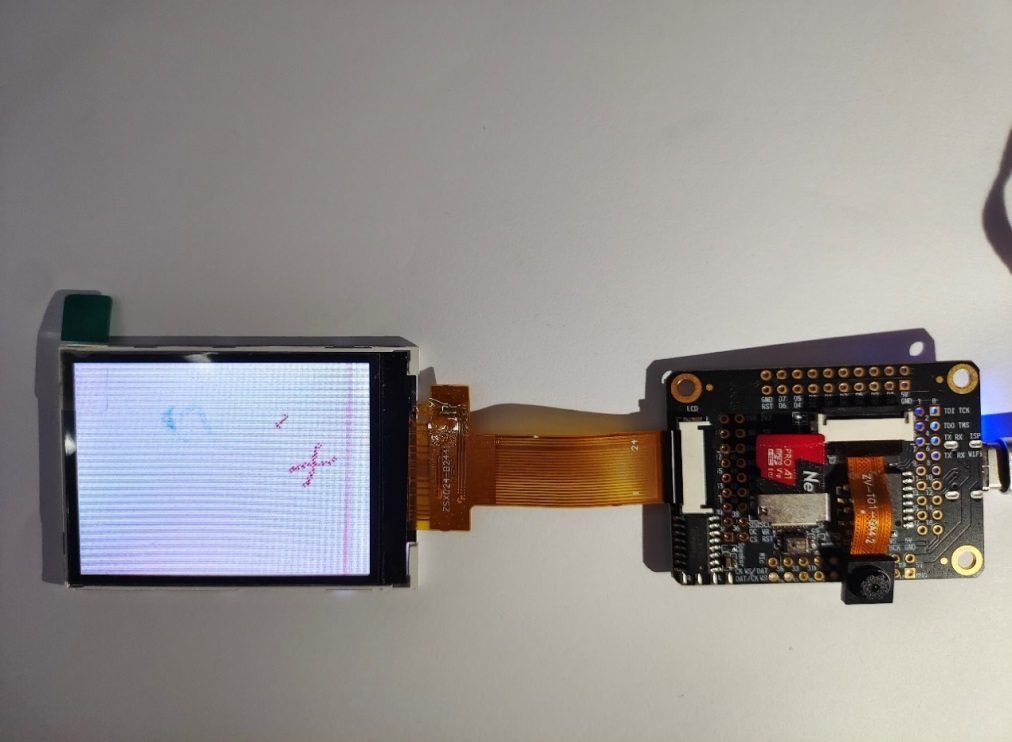


图5-4 K210板正常工作

1. 总结与展望
2. 结论

物联网设备的高速发展正在不断的改变人们的认识，新技术的出现是为了人们的生活更加便捷。探索操作系统的演进需要长久的兴趣和坚持，以伟大的Linux操作系统来说在1991年不过是芬兰学生的一个兴趣爱好，他写下了这样的一段话：“你好，所有使用 minix 的人 -我正在为 386 ( 486 ) AT 做一个免费的操作系统（只是为了爱好）”，这个爱好诞生了目前极为成功的Linux。正是由于这样的理想主义者的存在，人们的生活才具有了更多的可能性。本设计内核的开发和学习也是源于兴趣，当大二接触到系统赛内核设计比赛的时候，本人就和千千万万致力于学习和研究内核的朋友们一起学习和改进内核，目前这个项目可能还存在一些问题，但是它具备了很好的基础，代码风格上也大量参考了许多成熟的项目采用统一的规范。自项目立项以来本设计翻阅了大量的开源实现参加，参加了很多次由陈瑜老师组织的内核模块化和实时性探索的经验分享，同时在选择立项的时候就和向勇老师沟通，成功的加入操作系统开源毕设项目中，全程在GitHub开源实现目前收获了62个star同时也接受了不少同学们的建议和帮助，因此可以看出这个系统的设计总体上是有它的成功之处的.在一些维度上的创新也是有探索的意义的。从裸机上打印一句“hello world”开始，构建多个系统调用实现中断处理，网络通信支持。一步一步看着QEMU上构建起来的庞大世界，累计到目前为止仅仅是Rust代码大约就有11000行左右还有其他辅助构建的文件并未统计。不得不说这是一件非常工作量巨大而且有意义的事情。

回过头来，拥有一个属于自己的内核仍然是值得兴奋的一件事，这或许是计算机专业学生的一点浪漫，同时也是在开源的过程中，发现了很多保有热情的开发者们，大家一起在各自喜欢的领域探索交流是一件极其充实和幸福的事。设想之后把内核的学习和开发作为兴趣，关注最新的进展尝试为开源社区贡献一些代码或是在开源布道过程中进行可能的宣传和建设。这些工作无疑都是值得兴奋的。

1. 未来展望

本科的学习已经步入到最后的阶段，项目的开发和设计也基本上具备了一些功能，目前来看本设计具备了实时性内核基本特性，同时在调度算法上也有相应的探索，在网络通信，GRUB规范启动和VGA输出等都进行了多次的测试，但是总体来看本设计依旧有很多优化的地方。因此希望在以后有相关的感兴趣的开发者和工程师在原来的项目下面提出issue一起完善。目前仓库截图如6-1所示。

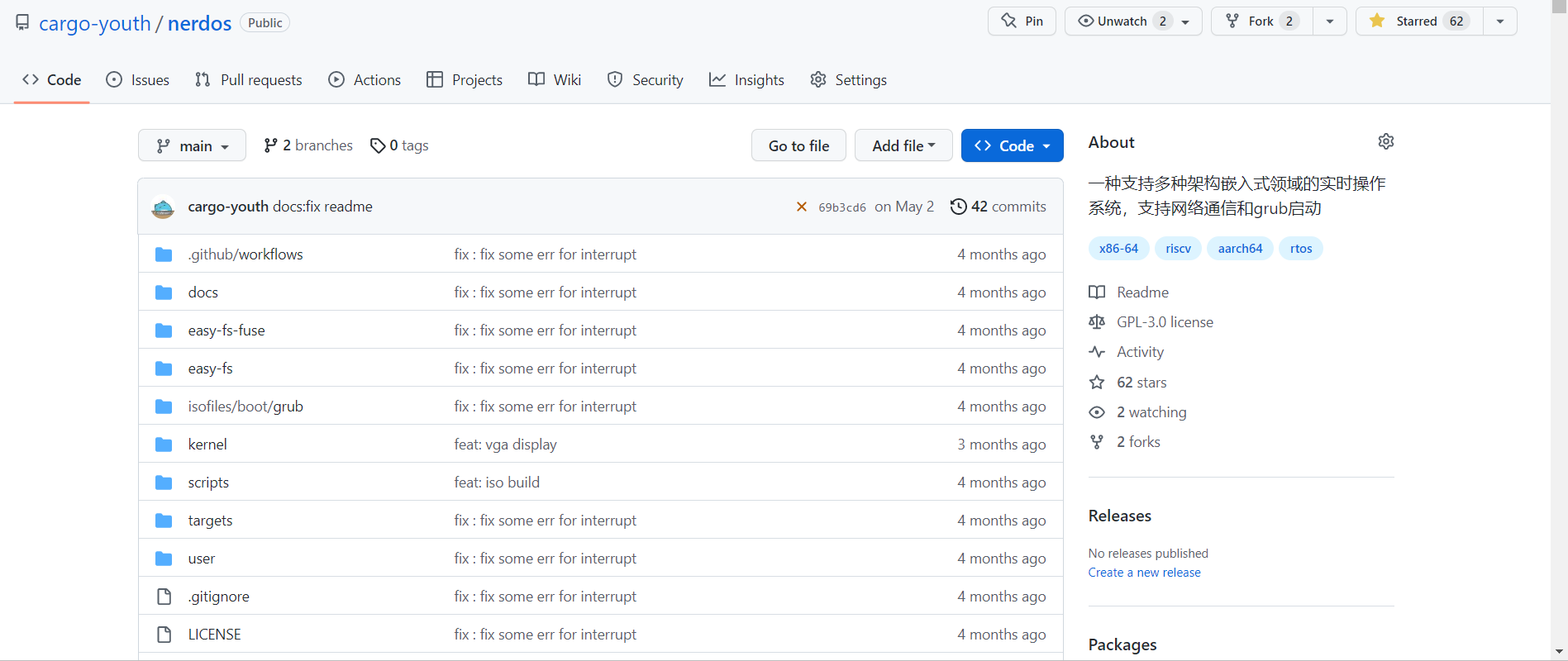


图6-1 仓库主页示意图

一、 软件端

调度算法的设计还不够完美，目前和整体项目的耦合度比较低，预期的目标是在之前RR时间片轮转调度的基础上同时兼有CFS调度的实现，需要共享一个trait这样的设计自然也需要付出更多的努力，当然之前的内核态调度也还有一定的挑战。而且在最近ChatGPT技术发展下也有很多同学和老师在研究用大模型辅助调度算法的研究，工业界开源之夏社区基于CFS调度的监控插件等的研究也是一个值得研究的领域。

其他方面，本设计开源的仓库部署了自动构建的CI环境，目前来看还不够完善，不能支持完成开发后的自动测试，这仍需要一定的调整，同时代码的规范性也需要加强。在其他项目的扩展上MQTT协议的移植也是很想做的一部分，目前移植了一些阿里社区的源码但是还没加载到内核中运行。

二、 硬件端

本设计因为是跨平台的设计，因此适配的硬件平台是比较丰富的，这既是优点但是在为硬件提供的配置支持上自然就会涉及很多不同开发板的细节，这也是较为繁琐的一点，目前在真实x86物理机上的启动测试已经是比较困难的任务了而RISC-V的开发板k210的目前调试仅限于就是烧录后启动这样相关的支持，在拓展相关模块细节上还有很多可以改进的地方，因此有很大的提升空间。

参考文献

[1] Rust programming language[EB/OL]. 2019[2019-05-23]. https://www.rust-lang.org

[2] Redha Gouicem and Damien Carver, Jean-PierreLozi et al. Fewer Cores, More Hertz[J]. 2020 USENIX Annual Technical Conference. 2020,6(15):435-448

[3] 王润基. Rust语言操作系统的设计与实现[D].北京:清华大学.2019.

[4] 潘庆霖. zCore 操作系统内核的设计与实现[D].北京:清华大学.2020.

[5] Ricardo Bianchini and Ram Rajamony. Power and energy management for server systems[J]. Computer. 2004, 37(11):68–76.

[6] Janis Voigtlaender and Sebastian Erdweg. RustBelt: Securing the Foundations of the Rust Programming Language[J]. ACM Transactions on Programming Languages and Systems. 2021, 43(1): 1-82.

[7] The rust programming language book[EB/OL]. 2019[2019-05-23]. https://doc.rust-lang.org/ book/.

[8] Trevor Jim, Greg Morrisett, Dan Grossman et al. Cyclone: A safe dialect of C[J]. 2002 USENIX Annual Technical Conference. 2002, 6（15）.1-14.

[9] Chan, W.-T., Lam, T.-W., Liu, K.-S., Wong, P.W.H.: New resource augmentation analysis of the total stretch of SRPT and SJF in multiprocessor scheduling. Theoretical Computer Science 359, 430–439 (2006)

[10] Kai Shen, Ming Zhong, and Chuanpeng Li. I/O system performance debugging using model-driven anomaly characterization. In FAST, pages 309–322, San Francisco, CA, USA, 2005.

[11] 陈磊,蔡铭,史昆.任务及中断负载下实时操作系统性能评估研究[J].计算机工程与应用.2014(17):80-85.

[12] Ralf Jung, Jacques-Henri Jourdan, and Robbert Krebbers. RustBelt: Logical Foundations for the Future of Safe Systems Programming[J]. ACM SIGPLAN Conference on Programming Language Design and Implementation.2022.

[13] Boqin Qin, Yilun Chen, Zeming Yu et al. Understanding Memory and Thread Safety Practices and Issues in Real-World Rust Programs[J]. ACM Digital Library.2020,6(15). 763–779.

[14] Levy A, Campbell B, Ghena B, et al. Multiprogramming a 64kb computer safely and efficiently [C]//Proceedings of the 26th Symposium on Operating Systems Principles. [S.l.]: ACM, 2017: 234-251.

[15] 李孝成.SylixOS大型实时操作系统实时性分析[J].单片机与嵌入式系统应用杂志.2019(11):10-12.

[16] 张东,卫一芃.嵌入式实时操作系统中虚中断的设计与实现[J].信息通信.2018(04):134-135.

[17] 吴雨俊.实时操作系统Nucleus的中断处理机制研究[J].福建电脑.2012(03):95-96.

[18] 王运盛,王坚.VxWorks实时操作系统中的中断处理机制分析[J].电讯技术.2007(04):178-181.

[19] 齐爱玲.嵌入式实时操作系统时钟中断服务的分析与改进[C].第五届全国信息获取与处理学术会议.2007:780-782.

[20] 史昆.基于中断响应模型的实时操作系统性能评价技术研究[A].2011.

[21] 陈云南.一种基于ARM芯片的实时操作系统中断机制[A].2011

[22] 任慰.以实时操作系统为中心的嵌入式系统平台化设计研究[D].湖北：华中科技大学.2013.

[23] 胡勤霞，刘庆峰，王勇利.软件系统测试的组织与管理方法分析[J].科技资讯，2007（26）：248-249.

[24] 陈卫卫.软件测试[M].西安:西安电子科技大学出版社,2011:12.

[25] 聂长海.关于软件测试的几点思考[J].计算机科学,2011,38(2):1-3.

致 谢

时间过得真快啊，转眼就走到了分别的路口。大家都在向自己想要的方向努力，这是一次四年本科学习生涯的回顾和致谢。首先，感谢我的导师梁琨老师，多亏你的照顾，在大学的学习生活中得到了很多帮助。同时，也要感谢物联网教研室的老师们，你们的严谨治学态度和尽心尽力的教学是我求学路上的宝贵财富。

在我的CS学习中，也遇到了很多朋友。大二时，是陈志扬同学和原毅哲同学介绍我参加了操作系统比赛，从此开始了一系列奇遇。顺利找到实习工作后，我遇到了很多真诚的同事，其中包括负责指导我工作的严欢（欢哥）、项目组长李韶雄（韶雄哥）和帮我调试代码的李宇姐、文宇哥、唐航哥以及沈梦菊同学、郑万杰同学、李锐同学和魏素雅同学。这些经历是我在后续内核比赛中获奖的重要支撑。在为内核比赛学习和开发的过程中，吴一凡同学和王润基同学、贾越凯博士给了我很多帮助，甚至在深夜帮我调试代码。他们对系统设计的严谨和热情，我时刻铭记着。同时感谢那些公开课群友们，他们是系统赛大赛群，一生一芯，mit824 、828，cmu15445、cs182、 cs61abc、健身交流群……这些群里我学习到了很多专业知识

此外，我要特别感谢向勇老师和陈瑜老师。他们的操作系统慕课教学深入浅出，注重社区建设，同学们有问题都可以在pizza和社区群讨论。向老师还十分关心同学们的发展和想法，他的指导帮助我把一个小想法变成了一个实际落地的开源网站，并被系统赛官网采纳。在这个过程中，我也要感谢社区同学不断地提交pr。

最后，感谢与我一起并肩作战的研友们（志扬、昊林、勇哥、花宝）和考研路上的引路人们（嘻宝、炸药哥、刀哥、关公哥、神明哥、匡佬、兰秋兄、dragon哥、威神等），我想向上攀爬的路总是令人感到振奋的，我们也终将顶峰相见。还要感谢我的家人和朋友们（潇洒哥和实验室五人组以及求学路上认识的同学们）多年来的支持，是你们见证了我生命中的每一个重要时刻也和我一起承担失败之后的痛苦。特别是考研失利后你们依旧笃定地支持我，勉励我学习，尤其是吴浩宇同学和张梓靖同学、温富康同学他们丰富的知识储备总能在关键时刻帮助我而且幽默风趣的性格总是让我感到轻松。我庆幸能够遇见这么多的良师益友，我庆幸在我最具才华的年纪，遇到我的热爱，与你们一起奋斗的时间是我此生永远的珍宝，我将铭记于心。