北京理工大學

本科生毕业设计（论文）

跨操作系统的异步串口和网络驱动模块设计与实现

Design and Implementation of Asynchronous Serial Port and Network Driver Module Across Operating Systems

学　　院：计算机学院

专　　业：计算机科学与技术

班　　级：

学生姓名：林 晨

学　　号：1120202738

指导教师：陆慧梅

2024 年 5 月　日

原创性声明

本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。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。

特此申明。

本人签名：　　日 期：　　年　月　日

关于使用授权的声明

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名：　　日 期：　　年　月　日

指导老师签名：　　日 期：　　年　月　日

跨操作系统的异步串口和网络驱动模块设计与实现

摘　要

关键词：

Design and Implementation of Asynchronous Serial Port and Network Driver Module Across Operating Systems

Abstract

Key Words:

目 录

第 1 章 绪论

1.1 研究背景

在操作系统发展中，外设起到了非常重要的作用。为了满足不同需求而设计的外设，帮助操作系统实现了丰富的功能。从操作系统的架构上看，驱动程序是与各种外设进行直接交互的软件组件。由于外设种类多样，不同厂家对于相同外设的设计也不尽相同，这就导致了操作系统开发人员需要花大量的时间在对硬件手册的阅读以及对外设驱动程序的实现和调试上。但几乎所有的硬件外设，独立于上层的操作系统实现，因此针对于一种特定的硬件，设计并实现一个对上层操作系统独立的硬件驱动模块，供操作系统开发人员直接调用，能够帮助在其开发的操作系统中快速实现对硬件的控制。

从操作系统用户进程的角度来讲，用户进程需要的是对外设进行读写，读写的方式是通过系统调用方式。在读写系统调用执行模型的设计上，主要包括阻塞和非阻塞、同步和异步等执行模型。如果一个系统调用采用同步执行模型时，那么调用该系统调用时，在完成该系统调用的全部任务前，该系统调用都不会返回；而当一个系统调用采用异步执行模型时，那么调用该系统调用时，会立即返回，尽管该系统调用所规定的读写任务还未达成。在计算机中有些 IO 处理比较耗时。在调用这种同步代码时，如果进程在此处长时间等待，会严重影响程序的性能，因此在 IO 密集型的应用场景中，采用异步编程，能够极大程度的提升程序的运行性能。另外，采用无栈协程实现的异步，相比于传统的多线程并发设计，更是省去了操作系统对于上下文，堆栈等进行开辟和切换的开销。

Rust 语言是一门适合系统编程的新兴编程语言，其拥有高于 C 语言的内存安全性、更现代的语法特性和等同 C 语言的性能。RISC-V 指令集则是一套开源、简洁、模块化的指令集。因此，基于 Rust 语言和 RISC-V 指令集的操作系统是操作系统开发的新兴方向之一。这一方向上已经出现了一些教学和科研目的的操作系统，例如 rCore 教学操作系统[1]。 Rust 所使用的包管理工具 Cargo 为使用 Rust 语言开发的项目提供了方便的依赖包管理，能够使操作系统开发人员很容易的使用本项目开发的异步驱动模块。另外，Rust 语言也对异步和协程的相关特性有所支持，能够使我们更加自然地实现一个异步的驱动模块。

1.2 国内外研究现状

异步 I/O 是指在进行事务处理时并发执行输入、输出以及计算操作的能力，可以将逻辑独立的 I/O 操作从指定的动作中分离出来单独执行。[2]简单来说，相较于阻塞的同步 I/O，异步 I/O 是非阻塞的，无需等到 I/O 操作返回相应的 I/O 结果，就可继续执行接下来的操作。

实现异步 I/O 的方式，包括使用操作系统提供的多线程接口，编程语言提供的函数回调、Future 等语言特性编写程序。方兴等人[3]提出了一种基于 WIN32 的多线程异步 I/O 模型，用于解决复杂的多路并发 I/O 问题，并说明了该模型的运行机制及其优越性，这是使用操作系统支持的多线程实现同步。段楠的研究[4]使用 Java 提供的“消息队列”方式进行了异步的网络通讯开发，在语言层面使用语言特性对异步进行了支持。Harris T 的研究[5]中更是将介绍了一组在 C/C++等原生语言中用于可组合异步 IO 的语言结构 AC，为开发人员提供了更方便的异步程序开发。Rust 在语言层面提供了对协程和异步的支持，使得开发人员能够方便的使用 Future 特性开发异步模块，实现异步外设驱动。

使用异步 I/O 接口，在大量 I/O 的应用场景下，能够极大的增加整个系统的吞吐量，使用无栈协程所实现的异步 I/O 相较于用多线程实现的异步 I/O，更是能够避免分配大量的堆栈等内存空间，节省了系统资源。沙泉的研究[6]在嵌入式 Linux 的串行通信中实现了异步事件驱动模型，在一定的硬件和实例测试的环境下说明了异步串口通信程序的设计思路和实现。Zhu L[7]、Kwon G[8]等人的研究分别在自己所设计的监测系统和实时数据归档系统中使用了异步驱动模块，在 I/O 量比较大的应用场景下，提升了整个系统的吞吐率和安全性。

目前，使用 Rust 语言编写操作系统组件逐渐成为趋势。Linux 已经对其众多的硬件驱动模块进行 Rust 语言改造，也进一步印证了使用 Rust 语言可以编写更安全的操作系统正在逐渐成为操作系统领域的共识。在 Rust 语言社区中，基于 Rust 语言对各种硬件设备的驱动开发也正在如火如荼地进行着。

1.3 本文研究内容与贡献

1.3.1 研究内容和关键问题

本研究的主要目标是使用 Rust 语言针对 QEMU 虚拟机模拟的串口设备以及

Vision Five2 实体开发板上的串口设备，开发跨操作系统的异步驱动模块，并对该异步驱动模块的性能与传统的同步模块的性能在一定条件下进行比较，最终得出异步驱动模块相较于同步模块的优势。

本文的研究内容主要包括以下几项：设计并实现基于 Rust 语言的异步串口驱动；在 AlienOS 和 ArceOS 上适配该异步串口驱动；设计对比实验，对该异步串口驱动的性能与同步串口驱动的性能进行比较等。

本研究的关键问题包括：学习 Rust 语言对于异步的支持、了解 Rust 语言在模块化方面的支持、学习和总结已有串口驱动的实现、学习 AlienOS 和 ArceOS 的相关结构并分别在 QEMU 环境下和开发板环境下运行、设计对比试验得出异步串口驱动的优势。

1.3.2 本文贡献

本文基于 Rust 语言，设计了能够在 QEMU 模拟器所模拟的 qemu-system-riscv64 虚拟环境下 AlienOS 和 ArceOS 都能够使用的异步串口驱动。截至目前，本研究已经完成了异步串口驱动的设计与实现，在 QEMU 中添加了多串口的支持使得在支持多个串口的 qemu-system-riscv64 环境下的 Alien 中能够同时使用原有的同步串口驱动和我们所开发的异步串口驱动模块。我们将新创建的使用异步驱动的串口绑定到一个指定的终端，在对应的终端能够看到上层操作系统通过异步串口驱动输出的相关信息，也能通过终端输入完成 OS 创建的读字符操作。

第 2 章 相关技术介绍

2.1 Rust Cargo 和 Crates.io

本部分我们将介绍 Rust 语言在包管理部分的相关技术，包括软件包管理器 Rust Cargo 以及官方的包管理平台 Crates.io，说明使用 Rust 语言开发驱动模块能够更方便地使用他人的已有成果，并使得我们的异步驱动模块能够更方便地被其他有需要的人所使用。

2.1.1 Rust Cargo

Rust 是为了解决 C/C++等底层程序设计语言存在的安全问题，设计的新一代安全系统级编程语言，通过其精心设计的所有权、生命周期等机制，保障了其不会出现 C、C++语言中悬垂指针等危险的内存操作。但除了其本身是一门安全、出色的编程语言之外，Rust 还具有一个名为 Cargo 的构建系统和软件包管理器。Rust 与 Cargo 捆绑在一起，在安装 Rust 时会自动安装 Cargo。

Rust Cargo 为 Rust 开发者提供了便捷、高效和可靠的开发工具和基础设施：在项目初始化时，开发者可以使用 `cargo new` 命令立刻创建一个新的 Rust 项目的基本结构，包括 Cargo.toml 文件和 src 目录，以及 src 目录下一个默认的 main.rs/lib.rs；在项目引用某些依赖项时，开发者可以通过 Cargo.toml 文件指定项目所依赖的外部 crate，并附带上版本信息和其他约束条件，Cargo 会负责下载和管理这些依赖，保证项目的构建和运行环境符合开发者的要求；在构建系统时，开发者可以使用 `cargo build` 命令快速地编译、构建和打包 Rust 项目，同时支持调整构建参数和选项，以及与编译器和链接器的集成；在测试支持方面，Cargo 还内置了对 Rust 项目的测试支持，开发者可以在项目中编写单元测试和集成测试，并使用 `cargo test` 命令来运行这些测试。

同时，在项目的发布方面，开发者可以通过 Cargo 轻松地发布他们编写的 Rust 项目，把他们精心设计的 crate 发布到 crates.io（Rust 包的官方仓库），使其能够提供给其他开发者引用，促进 Rust 生态的健康发展。

2.1.2 Crates.io

Crates.io 是 Rust 编程语言的官方包管理平台，也是 Rust 生态系统中最重要的组成部分之一。在 Crates.io 上，开发者可以发布、发现和共享 Rust crate，从而

促进 Rust 社区的发展与合作。该平台提供了一个中心化的位置，使得开发者们能够更加轻松地查找和使用其他人编写的 crate，同时也方便了开发者分享自己的代码作品。

通过 Rust Cargo 和 Crates.io 的集成，使得开发者能够通过 Cargo 直接安装和管理他们在 Crates.io 上发现的 crate，同时这种紧密的集成也减轻了开发者本身需要在依赖管理上投入的精力，提高了整体的项目开发效率。在本研究所开发的异步串口驱动因此能够更方便地使用其他人已经实现的 Rust crate，而其他操作系统开发者也会更方便地使用我们的驱动 crate 进行操作系统开发。

2.2 Rust Future

本部分将介绍一些常见的并发编程方式，在对比中分析 Rust Future 所具备的优势，之后介绍有关 Rust 运行时的相关信息。

2.2.1 常见的并发编程方式

常见的并发编程的方式主要包括：线程、绿色线程、函数回调、Promise 等。

1）使用线程实现并发编程

首先需要肯定的是，操作系统线程肯定有一些相当大的优势，使得我们在讨论“异步”和并发性的时候总是把线程摆在首位。相较于其他方法，使用操作系统提供的线程是非常简单易用的，并且由于操作系统提供的任务切换机制，在不同的线程之间的切换非常快，不需要程序员使用过多的付出便能得到很好的并发性，因此对于大量的并发问题而言，使用操作系统提供的线程通常都是最正确的解决方案。

但也存在一些场景，使用线程机制实现的并发可能会带来一些麻烦：由于操作系统级别线程的堆栈都比较大，如果在类似于负载很重的 web 服务器的情况下，服务器的内存将会被很快的用尽；操作系统有的时候会有很多事情需要处理，可能并不能像程序员所希望的那样快速切换回需要执行的线程。另外，有的操作系统本身可能并不支持线程机制，也就导致了其上的应用程序无法使用线程实现并发编程。

2）使用绿色线程（Green Thread）实现并发编程

绿色线程是一个完全在用户空间实现的线程系统，因此也被称为用户态线程。在具体的实现上，用户空间中的线程库会把管理线程所需要的数据结构在进程地址空间内部实例化，然后就可以调用线程库来管理用户空间内的多线程。由于绿色线程完全运行在用户空间内，因此不需要内核支持线程机制，内核也并不知道这些现成的存

在。但也正因为这些线程对于内核是不可见的，所以一旦这些线程中有一个被阻塞，则其余的线程均被阻塞。

但由于该特性并不是一个零成本抽象（即在语言中加入该特性后，会对不使用该特性的其他开发者所编写的程序的性能产生影响），于是 Rust 语言的开发者在 Rust 1.0 版本删除了该特性。同时如果我们想要对绿色线程支持许多不同的平台，也很难正确实现。

3）使用函数回调（callback）实现并发编程

基于回调所实现的方法背后的主要思想为保存指向一组指令的指针，这些指令会在我们希望执行的时候执行。而针对 Rust 语言，这将是一个闭包。

对于函数回调而言，它在大多语言中都易于实现。并且由于不需要记录线程中类似于线程上下文的数据结构，函数回调没有上下文切换的性能开销，大多数情况下在内存上的开销也相对较低。但函数回调的缺点也很明显：首先，每个任务必须保存它以后需要的状态，内存使用量也将随着一系列计算中回调的数量线性增长；并且，有的时候随着回调函数嵌套层数的增加，代码可读性会非常的差，后期并不好维护；此外，由于 Rust 语言自身的所有权机制，在 Rust 语言中使用函数回调时任务之间的状态共享也会是一个难题。

4）使用 Promise 实现并发编程

通常，Promise 和 Future 两个名词在并发模型中是混用的，无论从 Promise 还是 Future 都表示异步计算结果的抽象。对于 Promise 而言，编译器在编译时会将其重写为一个状态机。整个 Promise 中的每个异步事件执行成功/失败后都会更新状态机的状态，随着所有异步事件的依次执行完毕，Promise 会返回整个异步事件的结果。由于 Promise 与 Future 的实现比较相似，具体的这种状态机是如何设置，又是如何工作的，我们会在 2.2.2 Rust Future 中会详细介绍

2.2.2 Rust Future

Rust 中的 Future 类似于 Javascript 中的 promise，是 Rust 语言中的一种表示异步计算结果的抽象。它允许开发者编写异步代码，处理异步任务的完成和错误，并在任务完成时收获一定的结果。Future 是 Rust 中异步编程的基础，通常与 async 和 await 关键字一起使用。

+ Leaf-future

+ 代表访问一个资源（如 socket），运行时创建的实现了 `Future` trait 的对象

+ Non-leaf-future

  + 使用 async 标记的代码块

2.2.3 Rust 运行时

Rust 运行时与 Rust Future 的关系

+ 三事件

  + 轮询 （在 executor 上执行）

  + 等待

  + 唤醒 （由 reactor 等待事件并唤醒 Future）

2.3 Embassy

Embassy 简介

它里面的异步运行时分析

2.4 Alien 和 ArceOS

Alien 简介

ArceOS 简介

对于模块化的探索

第 3 章 异步串口驱动模块的设计与实现

3.1 串口基本机制

3.2 整体架构设计

3.3 异步运行时设计

3.4 接口设计

第 4 章 异步串口驱动测试

4.1 修改 QEMU 源码使其支持多串口

4.2 在 Alien 上使用异步串口驱动

4.3 测试结果分析

（4.4 在 ArceOS 上使用异步串口驱动 ）

第 5 章 总结和展望

5.1 论文总结

5.2 不足和展望

结　论

参考文献

[1] 孙卫真,刘雪松,朱威浦,等. 基于 RISC-V 的计算机系统综合实验设计[J]. 计算机工程与设计,2021,42(4):1159-1165. DOI:10.16208/j.issn1000-7024.2021.04.037.

[2]卫一芃,杨晓宁.嵌入式实时操作系统异步 I/O 技术的研究[J].信息通信,2017(01):141-142.

[3]方兴,秦琦,刘维国.多线程异步 I/O 模型[J].舰船电子对抗,2005(04):61-64.DOI:10.16426/j.cnki.jcdzdk.2005.04.014.

[4]段楠.异步非阻塞网络通讯技术研究[J].现代计算机,2019(17):79-82.

[5] Harris T .Special Topic: AC – Composable Asynchronous IO For Native Languages[C]//Conference on Object-Oriented Programming Systems, Languages, and Applications.ACM, 2011.DOI:10.1145/2048066.2048134.

[6]沙泉.异步事件驱动模型在嵌入式系统中的应用[J].微计算机信息,2007(29):33-34+73.

[7] Zhu L , Huang L , Fu P ,et al.The upgrade to the EAST poloidal field power supply monitoring system[J].Fusion Engineering and Design, 2021, 172(10):112757.DOI:10.1016/j.fusengdes.2021.112757.

[8] Kwon G , Lee W , Lee T ,et al.Development of a real-time data archive system for a KSTAR real-time network[J].Fusion Engineering and Design, 2018, 127(feb.):202-206.DOI:10.1016/j.fusengdes.2018.01.019.

附　录A

致　谢

值此论文完成之际，首先向我的导师……

致谢正文样式与文章正文相同：宋体、小四；行距：22磅；间距段前段后均为0行。【阅后删除此段】