北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

北京理工大学毕业设计开题报告

报告人：穆新宇

导　师：陆慧梅

时　间：2024/2/26

Contents

结构大纲

01

课题背景

Background of the project

在过去，许多操作系统和设备驱动程序使用的是同步编程模型，这意味着每个操作都需要等待设备完成后才能继续执行下一个操作。这种同步模型在处理慢速设备或需要长时间等待的操作时效率较低，因为它会阻塞其他任务的执行。而异步编程模型则可以在等待设备或操作完成的同时，继续执行其他任务，从而提高系统的性能和吞吐量。

使用Rust开发跨操作系统的异步驱动模块是一种解决方案，因为Rust具有良好的异步编程支持和底层硬件访问能力。Rust的异步编程模型基于Futures和async/await语法，使得编写异步代码更加直观和高效。同时，Rust的内存安全性和并发性能使其成为开发高性能和可靠驱动程序的理想选择。

02

课题目标

Project Target

使用Rust语言开发基于RISC-V开发板的跨操作系统异步驱动模块

利用Rust开发多个跨操作系统crate，每个crate被视为一种设备驱动(例如某种网卡驱动).

由于Rust天然对异步编程具有良好的支持，所以可以将异步机制与底层硬件特性结合，开发更高效的驱动，供上层操作系统使用.

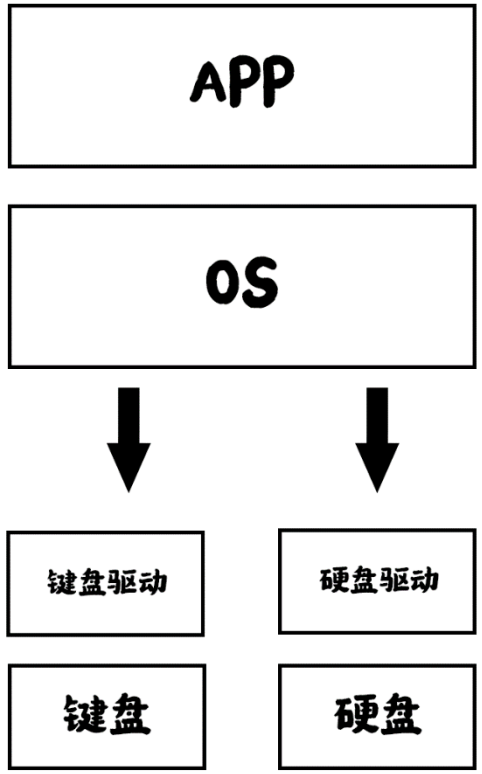
课题的最终目的是为RISC-V开发板上面的硬件开发对应的驱动程序，并测试其性能.

03

实现方案

Implementation Plan

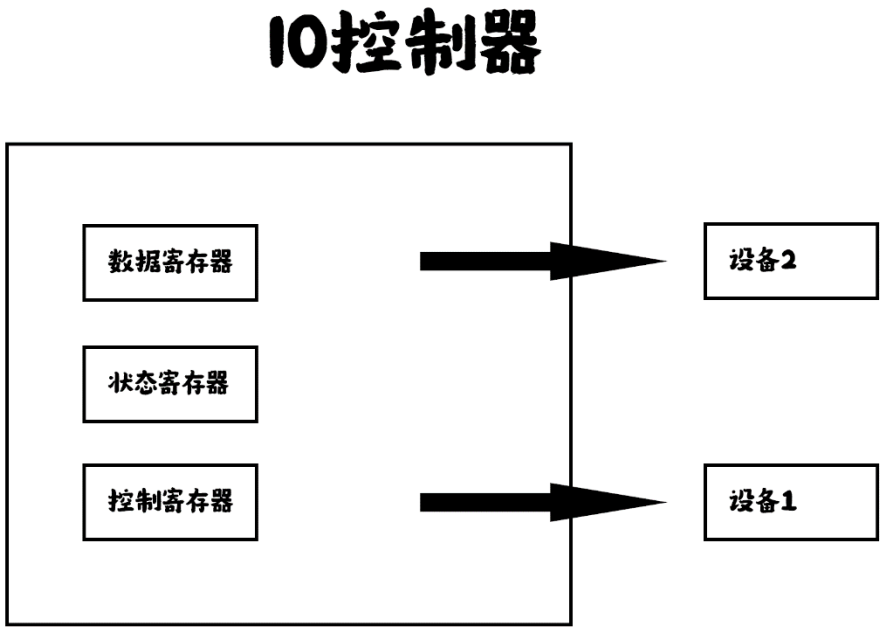
实现方案-IO软件层级



App使用操作系统提供的系统调用来访问硬件(例如对块设备的统一访问系统调用是read(哪个设备,从那开始，几个字节)).

操作系统规定了对于不同类型设备的驱动程序接口的形式，硬件设备厂商实现的驱动程序必须遵循这些接口和规范才能被操作系统调用.

实现方案-IO控制器



设备驱动程序可以作为操作系统内核的一部分，运行在内核态.

设备驱动程序读取、更改IO控制器中的三种设备寄存器，IO控制器将更改后的寄存器状态通过总线发送给外部硬件设备，从而使硬件设备能根据不同的寄存器状态来正确工作.

实现方案-跨操作系统


```rust
#[cfg(target_os = "linux")]
fn platform_specific_code() {
    println!("Running on Linux");
    // Linux-specific code here
}

#[cfg(target_os = "windows")]
fn platform_specific_code() {
    println!("Running on Windows");
    // Windows-specific code here
}

#[cfg(not(any(target_os = "linux", target_os = "windows")))]
fn platform_specific_code() {
    println!("Running on an unsupported operating system");
    // Code for unsupported operating systems here
}

fn main() {
    platform_specific_code();
}
```

Rust提供了条件编译的功能，可以根据目标操作系统进行选择性编译.

通过使用cfg属性，可以编写适用于特定操作系统的代码块,并在编译时选择性地包含或排除这些代码块. 这样可以根据目标操作系统的不同，编写和调整相关的驱动代码.

实现方案-异步


```rust
use tokio::fs::File;
use tokio::io::AsyncReadExt;

async fn read_file_content() {
    let mut file = File::open("example.txt").await.unwrap();
    let mut buffer = vec![0; 1024];
    println!("Start reading file...");
    let bytes_read = file.read(&mut buffer).await.unwrap();
    println!("Bytes read: {}", bytes_read);
    println!("Content: {:?}", &buffer[..bytes_read]);
}

#[tokio::main]
async fn main() {
    tokio::spawn(async {
        println!("Background task started");
        tokio::time::sleep(tokio::time::Duration::from_secs(2)).await;

        println!("Background task completed");
    });
    read_file_content().await;
}
```

Future：是一个trait，关键方法是poll，根据这个方法的返回值确定接下来要执行的动作.

async/await：async声明一个函数是异步函数，await等待异步操作的完成.

异步运行时：提供了异步任务的执行环境、任务调度器、定时器和其他必要的工具(tokio，async-std).

04

规划愿景

Plans of the Project

规划愿景

复现

参考已有的网卡驱动实现, 尝试自己复现一个简单的驱动程序, 弄清楚开发的大致过程.

异步编程

Rust对异步编程有很好的支持, 需要寻找参考资料弄清楚如何使用Rust进行异步编程.

跨操作系统

条件编译

测试

对已实现的驱动程序进行大规模系统测试, 并在保证正确性的基础上分析性能优劣.

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

感谢各位老师
请您批评指正

报告人：穆新宇

导　师：陆慧梅

时　间：2024/2/26