跨操作系统异步块设备驱动模块设计与实现

Design and Implementation of a Cross-Operating System Asynchronous Block Device Driver Module

董若扬

北京理工大学
计算机科学与技术专业

指导教师：陆慧梅

2024 年 5 月 31 日

任务书

北京理工大学

本科生毕业设计（论文）任务书

学生姓名	董若扬	学号	1120202944
学院	计算机学院	班级	07112005
专业	计算机科学与技术	题目类型	毕业设计
指导教师	陆慧梅	指导教师所在学院	计算机学院
题目来源	结合科研	题目性质	软件开发
题目	跨操作系统的异步块设备驱动模块设计与实现		

一、题目内容

利用 Rust 语言模块化、异步支持的特点，将异步机制与文件系统，底层块设备结合起来，实现更高效的异步驱动，并且能够在多个操作系统中应用。

二、任务要求

理解 Rust 语言异步编程及其原理；

理解 Rust 语言模块化思想；

理解块设备与文件系统的设计细节，思考其是否具有异步特性；

在 VisionFive2 晨光 2 实体开发板环境下完成性能对比实验。

三、进度安排

1. 在指导教师指导下阅读国内外　献和　学相关知识。（1-4 周）

2. **基于前序的知识学习，理解块设备与文件系统的设计细节，思考其是否具有异步特性；在 VisionFive2 晨光 2 实体开发板环境下完成性能对比实验；（6-11 周）。**

3. 完成本科　毕业设计（论　）外　翻译。（第 1 周-第 7 周）

4. 完成毕业论　，提交软件及相关　档。（第 13 周-第 14 周）

5. 完成本科　毕业设计（论　）答辩。（第 15 周）

四、主要参考文献

2394-async_await - The Rust RFC Book.
https://rust-lang.github.io/rfcs/2394-async_await.html.
3185-static-async-fn-in-trait - The Rust RFC Book.
https://rust-lang.github.io/rfcs/3185-static-async-fn-in-trait.html.

五、指导教师签字： 陆慧梅

2024 年 2 月 23 日

六、题目审核负责人意见

通过

签字：

2024 年 2 月 24 日

编程语言：Rust
运行环境：Ubuntu 20.04 LTS
工作量：

- 块设备驱动模块
- 裸机测试程序
- 适配 Alien
- 适配 rCore

创新点：

- 使用内存安全、高效的 Rust
- 实现基本功能以外的 Feature
- 实现不依赖特定运行时的异步
- 驱动模块可跨操作系统使用

查重结果及盲评结果

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

毕业设计（论文）匿名评阅评语表 1

学生姓名	董若扬	学号	1120202944
学院	计算机学院	专业	计算机科学与技术
题目	跨操作系统的异步块设备驱动模块设计与实现		
评阅结果	良(A) 82.0		

评语：

论文分析了面向教学的简易操作系统的开发需求，设计完成了将驱动模块从操作系统中分离，重新封装成库，为操作系统开发者提供拆箱即用的异步块设备驱动模块，并进行了模块的正确性测试，结论可信。论文结构有些不合理，大量篇幅用于介绍现有知识体系而非作者工作，引用规范。设计方案恰当，有一定工作量，过程规范，表明学生具有扎实的专业基础知识和综合运用能力，已基本具备独立工程实现能力，论文基本达到本科毕业设计对应的毕业要求，同意参加论文答辩。

评阅人：
2024 年 5 月 22 日

毕业设计（论文）匿名评阅评语表 2

学生姓名	董若扬	学号	1120202944
学院	计算机学院	专业	计算机科学与技术
题目	跨操作系统的异步块设备驱动模块设计与实现		
评阅结果	良(A) 81.0		

评语：

选题针对多种面向教学的简易操作系统在不同开发板移植过程中，可移植性不足，系统复杂性高，教学工作难度大等问题，具有一定的应用价值，符合复杂工程问题要求。论文介绍了相关领域研究现状，分析了实现 virtio 块设备驱动需求，设计完成了遵循 virtio 协议的跨操作系统异步块设备驱动模块，针对 I/O 设备工作时的阻塞问题的异步解决方案，并进行了实际环境的测试，结论可信。

论文结构合理，引用规范，设计方案恰当，有一定的工作量与创新性，过程规范，表明学生具有较为扎实的综合运用能力，已基本具备独立工程实现能力，论文达到本科毕业设计对应的毕业要求，同意参加论文答辩。

评阅人：
2024 年 5 月 26 日

答辩版查重结果：2.6%
盲评 1：A（82）
盲评 2：A（81）
未申请评优

大纲

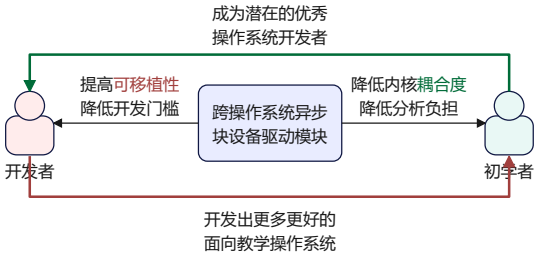
研究背景

社会对于国产操作系统的需求越来越高，在这种背景下涌现了许多开源操作系统，其中不乏一批被广泛用于教学研究，即帮助操作系统初学者入门学习操作系统设计的操作系统样例，例如基于 Rust 语言编写的rCore和Alien。如果这些操作系统需要部署在开发板上，则通常要求在操作系统中编写面向特定开发板的外设驱动模块。这给操作系统在不同型号的开发板间的迁移工作带来极大挑战，给教学工作带来了不必要的麻烦。同时，随着操作系统内核功能的不断增加，代码量也在不断膨胀，这增加了学习者理解和分析内核的难度和成本。因此，为了降低这种复杂性，并减少学习者的负担，有必要使操作系统的各个部分之间尽可能解耦。

研究目的和意义

为了应对这些挑战，本研究提出了将驱动模块从操作系统中剥离出来，以 crate 的形式与操作系统进行对接的解决方案。这种做法不仅便于驱动模块的开发和迭代，也能够增强其在不同系统中的可重用性。基于 Rust 的操作系统可以通过简单地在 Cargo.toml 文件中导入所需的驱动模块 crate 来实现这一目标。

这一研究将减轻操作系统开发者的开发负担，形成完善的操作系统驱动生态后会显著降低操作系统开发的门槛，相信将能极大促进开源操作系统的进展。另外本研究降低了系统的耦合性，将为初学者学习分析理解操作系统内核代码带来极大帮助，减轻学习者的负担。相信这一研究对于促进基于 Rust 语言的教学操作系统的发展具有重要的理论和实际意义。



大纲

主要研究工作

- 实现块设备的抽象接口
- 引入异步特性
- 支持更多Feature
- 完成Alien操作系统接入工作
- 完成rCore操作系统接入工作

大纲

Hypervisor

作为运行在硬件和操作系统层之间的一层，Hypervisor 使得计算环境能够在单个物理计算机上同时运行多个独立的操作系统，从而更有效地利用可用的计算能力、存储空间和网络带宽。

virtio 规范

virtio 协议是对 Hypervisor 中一组通用模拟设备的抽象，定义了虚拟设备的输入/输出接口。virtio 规范的主要目的是简化和统一虚拟机的设备模拟，并提高虚拟机环境下的 I/O 性能。基于 virtio 协议的 I/O 设备被称为 virtio 设备。

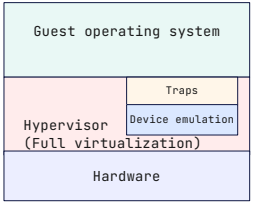
virtio 设备

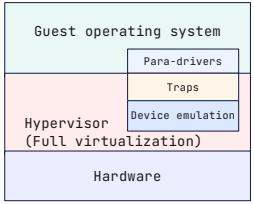
块设备（virtio-blk）、网络设备（virtio-net）、键盘鼠标类设备（virtio-input）、显示设备（virtio-gpu）等具有共性特征和独有特征。共性特征通过统一抽象接口进行设计，独有特征尽量最小化各种类型设备的抽象接口，从而屏蔽了各种 Hypervisor 的差异性，实现了 guest VM 和不同 Hypervisor 之间的交互过程。

virtio 虚拟机

虚拟机模拟外设的传统方案中（左侧），如果 guest VM 需要使用底层 host 主机的资源，那么 Hypervisor 必须截获所有的 I/O 请求指令，并模拟这些指令的行为。然而，这种方式会导致较大的性能开销。

虚拟机模拟外设的 virtio 方案中（右侧），当 guest VM 通过访问虚拟外设来使用底层 host 主机的资源时，Hypervisor 只需要处理少量的寄存器访问和中断，从而实现了高效的 I/O 虚拟化过程。



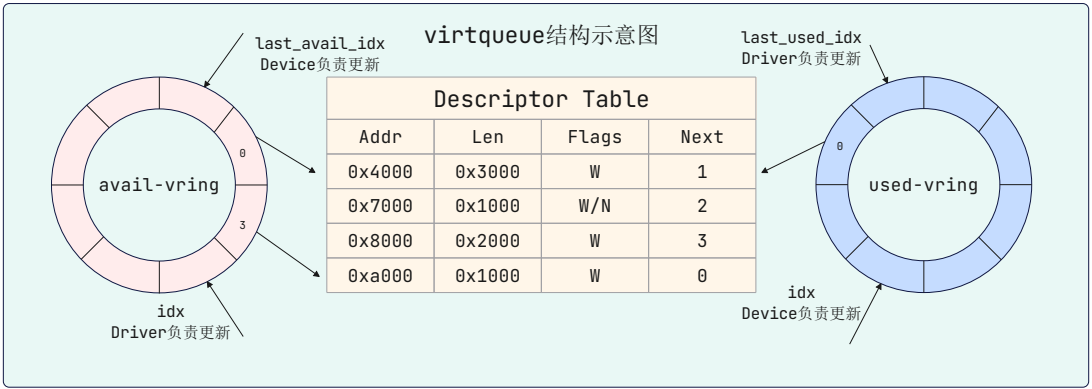


大纲

virtqueue 虚拟队列

为了实现批量数据传输，virtio 设备使用了 virtqueue 虚拟队列机制。每个 virtqueue 占用多个物理页。virtqueue 由三部分组成:

- 描述符表：描述符为组成元素的数组，每个描述符描述了一个内存 buffer 的 address/length。
- 可用环：记录了 virtio 设备驱动程序发出的 I/O 请求索引 (驱动更新的描述符索引的集合)，由设备进行读取。
- 已用环：记录了 virtio 设备发出的 I/O 完成索引 (设备更新的描述符索引的集合)，由驱动进行读取。



大纲

特征位

特征位用于表示 VirtIO 设备具有的各种特性和功能。驱动程序与设备之间进行特性协商，以形成一致的共识，从而正确地管理设备。

```
bitflags! {
    #[derive(Copy, Clone, Debug, Default, Eq, PartialEq)]
    struct BlkFeature: u64 {
        const GEOMETRY   = 1 << 4; //指示磁盘的几何结构
        const RO         = 1 << 5; //设备是只读的
        const BLK_SIZE   = 1 << 6; //设定磁盘块大小，影响调度策略
        const FLUSH      = 1 << 9; //设备支持刷新操作
        const TOPOLOGY   = 1 << 10;//指示磁盘的逻辑结构
        const CONFIG_WCE = 1 << 11;//0:write-through, 1:write-back
        const MQ         = 1 << 12;//设备支持多队列
    }
}
```

特征位的使用

在初始化时，根据协商的特征位，设备支持自定义更多配置信息。

```
let writeback = if negotiated_features.contains(CONFIG_WCE) {
            unsafe { volread!(config, writeback) }
        } else {
            1
        };
```

在调用函数时，根据特征位的协商结果，判断该驱动方法是否为设备所支持。

```
pub fn flush(&mut self) -> Result {
    if self.negotiated_features.contains(FLUSH) {
        ...
    } else {
        info!("device does not support flush");
        Ok(())
    }
}
```

设备状态域

在 virtio 设备初始化过程中，使用设备状态域来表示设备的状态。

1. EMPTY: 重置以确保设备处于初始状态
2. ACKNOWLEDGE: 操作系统设备已被识别
3. DRIVER: 操作系统已准备好驱动该设备
4. 写入操作系统和驱动程序能够处理的特性位子集至设备
5. FEATURES_OK: 指示驱动程序已完成特性协商
6. 此后，驱动程序不得接受新的特性位

状态	说明
ACKNOWLEDGE	驱动确认了一个有效的 virtio 设备
DRIVER	驱动知道如何驱动设备
FAILED	驱动无法正常驱动设备
FEATURES_OK	驱动已与设备就设备特性达成一致
DRIVER_OK	驱动加载完成，可以正常工作
DEVICE_NEEDS_RESET	设备触发了错误，需要重置

设备配置空间

配置中的信息是驱动相关的一些参数，当设备支持某些特征时，相关的参数就会被设置为配置中的数值。
基础配置信息有：capacity（容量），size_max（最大块数），seg_max（最大段数），blk_size（块大小）。

若协商特征包含GEOMETRY，则配置信息包含块设备的几何信息：cylinders（柱面数），heads（磁头数），sectors（单磁道扇区数）。

这些配置信息有助于操作系统和应用程序进行磁盘操作，如分区、格式化和文件系统管理。

若协商特征包含 TOPOLOGY，则配置信息包含块设备的拓扑结果信息：physical_block_size（物理块大小），alignment_offset（对齐偏移量），min_io_size（最小 I/O 规模），opt_io_size（最大 I/O 规模）。

这些配置信息有助于操作系统和应用程序优化存储访问和性能。

若协商特征包含CONFIG_WCE，则配置信息包含 writeback，用于表示缓存模式，0 表示write-through，1 表示write-back。若协商特征包含 MQ，则配置信息包含 num_queues，表示虚拟队列的数量。

设备配置空间

配置中的信息是驱动相关的一些参数，当设备支持某些特征时，相关的参数就会被设置为配置中的数值。
基础配置信息有：capacity（容量），size_max（最大块数），seg_max（最大段数），blk_size（块大小）。

若协商特征包含GEOMETRY，则配置信息包含块设备的几何信息：cylinders（柱面数），heads（磁头数），sectors（单磁道扇区数）。

这些配置信息有助于操作系统和应用程序进行磁盘操作，如分区、格式化和文件系统管理。

若协商特征包含 TOPOLOGY，则配置信息包含块设备的拓扑结果信息：physical_block_size（物理块大小），alignment_offset（对齐偏移量），min_io_size（最小 I/O 规模），opt_io_size（最大 I/O 规模）。

这些配置信息有助于操作系统和应用程序优化存储访问和性能。

若协商特征包含CONFIG_WCE，则配置信息包含 writeback，用于表示缓存模式，0 表示write-through，1 表示write-back。若协商特征包含 MQ，则配置信息包含 num_queues，表示虚拟队列的数量。

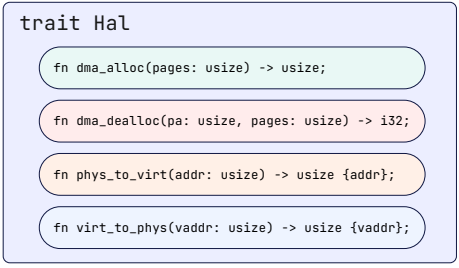
大纲

virtio-blk 设备结构

header 成员对应着 virtio 设备的共有属性，包括版本号、设备 ID、设备特征等信息。queue 是 virtio-blk 使用的虚拟队列。

```
pub struct VirtIOBlk<'a, H: Hal> {
    header: &'static mut VirtIOHeader,
    queue: VirtQueue<'a, H>,
    capacity: usize,
}
```

Hal(Hardware Abstraction Layer) 是 virtio_drivers 库中定义的 trait，用于抽象出与具体操作系统相关的操作，主要涉及内存分配和虚实地址转换等功能。



请求体 BlkReq & 响应体 BlkResp

BlkReq 结构体表示块设备的请求
BlkResp 结构体表示设备的响应状态

```
pub struct BlkReq {
  type_: ReqType,//请求的类型
  reserved: u32, //保留字段
  sector: u64,   //请求涉及的扇区号
}
```

状态	取值	状态	取值	状态	取值
In	0	Out	1	Flush	4
GetId	8	GetLifetime	10	Discard	11
WriteZeroes	13	SecureErase	14	Append	15
Report	16	Open	18	Close	20
Finish	22	Reset	24	ResetAll	26

```
pub struct BlkResp {
  status: RespStatus,
}
```

状态	取值	状态	取值
OK	0	IO_ERR	1
UNSUPPORTED	2	NOT_READY	3

大纲

Request & Request_Read & Request_Write

- **Request** 函数不带额外数据，用于刷新 flash 操作。
- **Request_read** 携带给定的数据，主要用于读操作。
- **Request_write** 携带给定的数据，主要用于写操作。

```
fn request(...


) -> Result {...
  add_notify_wait_pop(
    &[req],
    &mut [resp],
    &mut transport,
  )?;
  ...
}
```

```
fn request_read(...,
  data: &mut [u8]
) -> Result {...
  add_notify_wait_pop(
    &[req],
    &mut [data, resp],
    &mut transport,
  )?;
  ...
}
```

```
fn request_write(...,
  data: &[u8]
) -> Result {...
  add_notify_wait_pop(
    &[request, data],
    &mut [resp],
    &mut transport,
  )?;
  ...
}
```

Flush & Read & Write

- 刷新 **flush** 操作通过调用request函数完成
- （阻塞）读 **read_blocks** 操作通过调用request_read函数完成
- （阻塞）写 **write_blocks** 操作通过调用request_write函数完成

```
pub fn flush(...

) -> Result {
  if negotiated_features
  .contains(FLUSH) {
    request(
      BlkReq {
        type_: Flush,...}

    )
  } else {...}
}
```

```
pub fn read_blocks(...,
  buf: &mut [u8]
) -> Result {


  request_read(
    BlkReq {
      type_: In,...},
    buf
  )

}
```

```
pub fn write_blocks(...,
  buf: &[u8]
) -> Result {
  if !negotiated_features
  .contains(RO) {
    request_write(
      BlkReq {
        type_: Out,...},
      buf
    )
  } else {...}
}
```

异步读

read_blocks_nb 函数向 virtio-blk 设备提交请求，并返回标识在链中第一个描述符的位置的令牌。一旦设备完成处理，调用者在读取响应之前使用相同的缓冲区调用 **complete_read_blocks** 函数传递令牌来完成读取操作。

```rust
pub fn read_blocks_nb(
  ...
) -> Result<u16> {
  *req = BlkReq {In, ...};
  let token = queue.add(
    &[req],
    &mut [buf, resp]
  )?;
  if queue.should_notify() {
    transport.notify(QUEUE);
  }
  Ok(token)
}
```

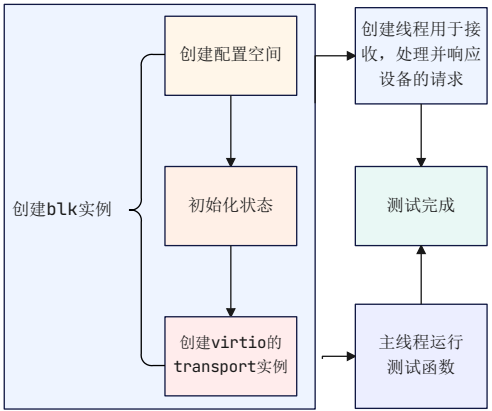
token

```rust
pub fn complete_read_blocks(...,
  token: u16,
) -> Result<()> {
  queue.pop_used(
    token,
    &[req],
    &mut [buf, resp]
  )?;
  ...
}
```

大纲

单元测试

在单元测试中，对块设备驱动程序中的函数和方法进行测试，以验证其在模拟输入环境下的输出正确性。总体的测试流程如图所示。



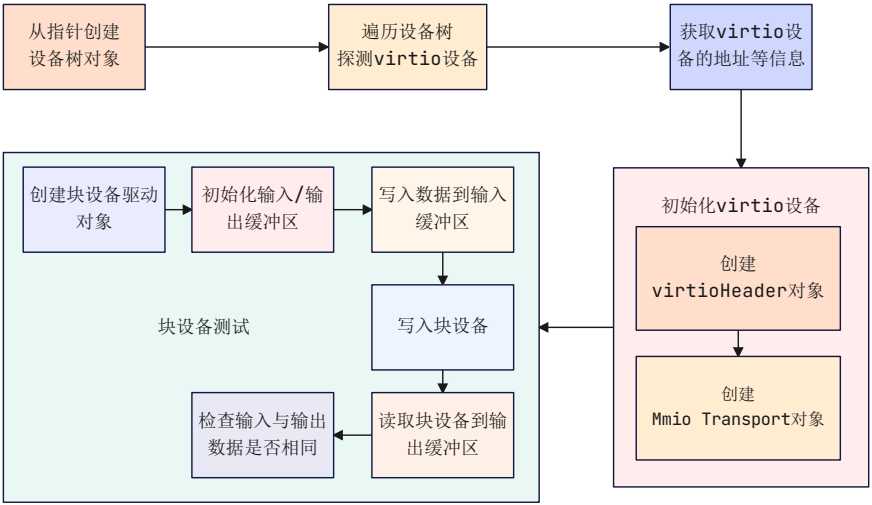

```rust
#[test]
fn test() {
  ...
  let mut config_space = BlkConfig {...};
  let state = Arc::new(Mutex::new(State {...}));
  let transport = FakeTransport {...};
  let mut blk = VirtIOBlk::
    <FakeHal, FakeTransport<BlkConfig>>::
    new(transport).unwrap();
  let handle = thread::spawn(move || {...});

  # Test here
  fn(...);

  handle.join().unwrap();
}
```

大纲

裸机环境集成测试-流程图



大纲

对接 virtio-blk 的准备-1

在 Cargo.toml 中添加本项目的 virtio-blk 依赖。

```
virtio-drivers = { git = "https://github.com/semidry/virtio_crate.git" }
```

定义 Trait，用于满足对块设备驱动的 I/O 访问要求。

```
pub trait BlkDevice: Send + Sync + Any {
    fn read_block(&self, block_id: usize, buf: &mut [u8]);
    fn write_block(&self, block_id: usize, buf: &[u8]);
    fn handle_irq(&self);
}
```

建立用于表示 virtio_blk 设备的全局变量 BLOCK_DEVICE。

```
pub static BLK_DEVICE: Once<Arc<dyn BlkDevice>> = Once::new();
```

对接 virtio-blk 的准备-2

创建结构体 VirtIOBlkDeviceWrapper 用于表示块设备，其中包含一个实现了 BlkDevice Trait 的设备。为其实现方法以及实现 VfsFile 和 VfsInode 特征。
VfsFile 特征定义了与文件操作相关的方法，例如从指定偏移量读取数据到缓冲区，将缓冲区的数据写入指定偏移量。
VfsInode 特征定义了与索引节点相关的操作，例如返回或设置索引节点的类型（如字符设备），获取索引节点的属性。
这些特征方法应由该操作系统的开发者自行决定实现方式，这里仅进行最简单的实现。

```
pub struct VirtIOBlkDeviceWrapper {
    blk: Mutex<VirtIOBlk<HalImpl, MmioTransport>>,
}
impl VirtIOBlkDeviceWrapper {...}
impl BlkDevice for VirtIOBlkDeviceWrapper {...}
```

在 Makefile 中为 qemu 模拟器添加虚拟块设备

```
define boot_qemu
        ...
        -drive file=$(IMG),if=none,format=raw,id=x0 \
        -device virtio-blk-device,drive=x0 \
        ...
endef
```

在 Alien 中启动块设备驱动

在内核态的主函数中，devices::init_device() 是启动设备的切入点。在初始化函数中，先获取设备树块（DTB）的指针并转换为字节指针以创建 Fdt（Flattened Device Tree）对象。接着对设备树进行遍历探测设备，对块设备执行 init_block_device() 进行初始化。
完成上述操作后，qemu 模拟器中设备树上的块设备以及块设备驱动已成功加载，并能够被操作系统所识别，如图所示，加载出块设备的地址并初始化成功，就可以对块设备进行操作了。然后进入了 Alien 的控制台，表明 Alien 正常启动。

```
[0] Init blk device, base_addr:0x10008000,irq:8
[0] Init blk device success
```

```
[0] Initrd populate success
[0] Init filesystem success
[0] ++++ setup interrupt ++++
[0] ++++ setup interrupt done, enable:true ++++
[0] Init task success
[0] Begin run task...
[0] kthread_init start...
Init process is running
Alien:/#
```

大纲

对接 virtio-blk 设备初始化过程

在 Cargo.toml 中添加本项目的 virtio-blk 依赖
建立表示 virtio_blk 设备的全局变量 BLOCK_DEVICE，以便内核能够识别和使用该设备。

```
pub struct VirtIOBlock {
    virtio_blk: UPIntrFreeCell<VirtIOBlk<'static, VirtioHal>>,
    condvars: BTreeMap<u16, Condvar>,
}
```

condvars 条件变量，用于在进程等待 I/O 读或写操作完成之前挂起进程。

VirtioHal 实现了 virtio_drivers 模块定义的 Hal trait，使得 VirtIOBlk 类型能够得到操作系统的服务。

```
impl Hal for VirtioHal
    fn dma_alloc(pages: usize) -> usize;
    fn dma_dealloc(pa: usize, pages: usize) -> i32;
    fn phys_to_virt(addr: usize) -> usize {addr};
    fn virt_to_phys(vaddr: usize) -> usize {vaddr};
```

对接 virtio-blk 设备

操作系统还需要对整体的中断处理过程进行调整，以支持基于中断方式的块读写操作。
有了基于中断方式的块读写操作，当某个线程/进程由于块读写操作无法继续执行时，操作系统可以切换到其它处于就绪态的线程/进程执行，从而提升计算机系统的整体执行效率。

```
impl BlockDevice for VirtIOBlock {
    fn handle_irq(&self) {...}
    fn read_block(...) {
        // 如果是中断方式
        if *DEV_NON_BLOCKING_ACCESS
          .exclusive_access() {
            ...
            let task_cx_ptr = ...;
            schedule(task_cx_ptr);
        } else {
        // 如果是轮询方式
            ...
        }
    }
    //write_block与read\_block处理类似
    fn write_block(...){...}
}
```

运行 rCore

进入 os 目录，使用命令 `make run` 启动 rCore，如图可以看到 rCore 进入了控制台并且文件系统成功加载出了一系列文件，证明文件系统对接的块设备驱动程序工作正常。输入命令打开对应文件（例如 pipetest），可以看到 rCore 正常运行，并回到了控制台命令行。说明该 virtio-blk 块设备驱动程序可以在 rCore 上正常工作。

```
pipetest
adder_peterson_yield
huge_write_mt
gui_snake
inputdev_event
cat
eisenberg
**************/
Rust user shell
>> pipetest
Read OK, child process exited!
pipetest passed!
>>
```

大纲

可继续改进的部分

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

进一步提高模块化程度

由于本研究遵循 virtio 协议，因此应能与其他 virtio 协议驱动协调工作。然而，与其他 virtio 驱动共同工作时，势必会产生一些冗余代码。未来的研究希望能分离出 virtio 协议依赖部分，使操作系统开发者在使用多种 virtio 驱动时无需重复引入冗余代码。

实现更多 Feature

virtio 规定的许多功能尚未在本研究中实现，这使得对块设备的许多高级特性，例如分区块设备及其独特的空间布局和拓扑信息，无法获得更优支持，可能会阻碍操作系统在调度时的进一步优化。未来的研究将分析块设备的物理特性，逐步实现 virtio 支持的高级特性。

支持多队列

I/O 命令被挂起后携带唯一的标识符（token）在循环队列中排队。对于零散的 I/O 操作，这种方法对 CPU 利用率的提升较为显著，但当 I/O 操作较为密集时，CPU 仍会因为 I/O 操作未完成而不得不等待。因此，将来可以在队列上做进一步优化，例如多队列。该改进不属于异步本身，但可以提升异步效果。

总结

本研究设计并实现了一个遵循 virtio 协议的跨操作系统异步块设备驱动模块。该模块支持更多块设备特性，既能为块设备提供通用服务，又能依据块设备支持的高级特性提供更多支持。同时，该解决方案不依赖任何特定的异步运行时，提供了一种面向底层设备的异步解决方案。模块化设计极大提高了其可移植性。

本研究或能减轻操作系统开发者的开发负担，使其只需导入依赖即可在操作系统中直接使用块设备驱动，从而增强了操作系统的可移植性，降低了开发成本。同时，得益于模块化和低耦合性的设计，本研究也能为操作系统的学习者减轻分析和理解操作系统内核的难度。

本研究仍有大量改进工作可以继续完成，其前景十分广阔，具有巨大的实用性和经济价值。遵循 virtio 协议，是对跨操作系统外设驱动问题提出的一种具有实际意义的可持续改进的现实解决方案。

感谢各位专家老师
请您批评指正！