北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

操作系统的驱动隔离改进与实现

Driver Isolation Improvement and Implementation for Operating System

答辩人：罗博文

导　师：陆慧梅

时间：2024/5/27

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

目 录

CONTENTS

第一部分 ▷▷

研究背景

- 面临问题
- 现有方案
- 新的方案

面临问题

- 操作系统是使计算机工作的重要部分。操作系统安全与健壮性是计算机正常工作的基础。
- 由于设备可能出现硬件故障，驱动程序经常会产生错误而危及整个系统。
- 驱动和硬件上的漏洞可能被攻击者利用，对计算机系统造成危害。

- 提升驱动的安全性、减少其对系统可能造成的负面影响非常有必要。

现有方案

基于硬件的隔离

特权级划分、虚拟地址空间、可信执行环境等基础都是通过芯片结构的设计来进行隔离。
该方法可靠性较高，但在隔离区运行的代码有漏洞的情况下仍不安全。此外成本昂贵，通用性低也促使人们寻找新的方案。

基于软件的隔离

沙箱、虚拟机、容器技术等都是几乎不使用硬件机制对代码进行隔离的技术。
该方法更加轻量级，提出时间较晚，因此还有很多发展空间。

基于语言机制的隔离

伴随着Rust、Go等内存安全语言的出现而兴起。该方法可被分类为基于软件的。但是与常见的软件隔离机制相比，具有如下优点：

更可靠

开发者无需自行验证安全约束。编译器保证的约束降低了开发者出错的可能性。

更快速

对安全性的检查在编译期完成，因此没有运行时开销。

使用基于语言机制的内存隔离，设计了驱动程序与内核、设备交互的接口，实现了一系列驱动。能够有效减少漏洞，可被多个操作系统复用。

在操作系统的支持下，支持故障隔离、动态加载、自动重载特性。

第二部分 ▷▷

安全接口设计

- 与系统的交互
- 与设备的交互

与系统的交互

驱动与操作系统的所有交互都被限制在了这有限的四个trait中。必须的操作都可以借由其中方法完成。

可能的安全风险来源于操作系统对这些方法的实现。最小接口可以保证出现漏洞的风险最小。

```
2 implementations
pub trait VirtIoDeviceIo: Send + Sync + Debug {
    fn read_volatile_u32_at(&self, off: usize) -> VirtIoResult<u32>;
    fn read_volatile_u8_at(&self, off: usize) -> VirtIoResult<u8>;
    fn write_volatile_u32_at(&self, off: usize, data: u32) -> VirtIoResult<()>;
    fn write_volatile_u8_at(&self, off: usize, data: u8) -> VirtIoResult<()>;
    fn paddr(&self) -> PhysAddr;
    fn vaddr(&self) -> VirtAddr;
}

1 implementation
pub trait DevicePage: Send + Sync {
    fn as_mut_slice(&mut self) -> &mut [u8];
    fn as_slice(&self) -> &[u8];
    fn paddr(&self) -> PhysAddr;
    fn vaddr(&self) -> VirtAddr;
}

1 implementation
pub trait QueuePage<const SIZE: usize>: DevicePage {
    fn queue_ref_mut(&mut self, layout: &QueueLayout) -> QueueMutRef<SIZE>;
}

1 implementation
pub trait Hal<const SIZE: usize>: Send + Sync {
    fn dma_alloc(pages: usize) -> Box<dyn QueuePage<SIZE>>;
    fn dma_alloc_buf(pages: usize) -> Box<dyn DevicePage>;
    fn to_paddr(va: usize) -> usize;
}
```

与设备的交互：虚拟内存映射

驱动与设备进行交互的所使用的方法是虚拟内存映射。设备的寄存器按照右表的布局映射到设备地址空间处。驱动使用上文定义的接口，可以对该区域进行读写，以读取基本信息、对设备进行设置。

Table 2 – Registers Summary

Address & Access type		Register		Bit number 7	6	5	4	3	2	1	0	
General Register Set												
000	R	Receiver Holding Register	RHR	Character Received								00
000	W	Transmitter Holding Register	THR	Character to be Transmitted								00
001	R/W	Interrupt Enable Register	IER	DMA Tx End	DMA Rx End	0	0	Modem Status	Receiver Line Status	THR Empty	Data Ready	00
010	R	Interrupt Status Register	ISR	FIFOs enabled	FIFOs enabled	DMA Tx End	DMA Rx End	Interrupt Identification Code			Interrupt Status	01
010	W	FIFO Control Register	FCR	Receiver's FIFO Trigger Level		0	Enable DMA End	DMA mode	Tx FIFO Reset	Rx FIFO Reset	FIFO enable	00
011	R/W	Line Control Register	LCR	DLAB	Set Break	Force Parity	Even Parity	Parity Enable	Stop Bits	Word Length		00
100	R/W	Modem Control Register	MCR	0	0	0	Loop back	Out 2 / Int. Enable[2]	Out 1	RTS	DTR	00
101	R	Line Status Register	LSR	FIFO data Error	Transmitter Empty	THR Empty	Break Interrupt	Framing Error	Parity Error	Overrun Error	Data Ready	60
110	R	Modem Status Register	MSR	CD	RI	DSR	CTS	delta CD	trailing edge RI	delta DSR	delta CTS	00
111	R/W	Scratch Pad Register	SPR	User Data								00

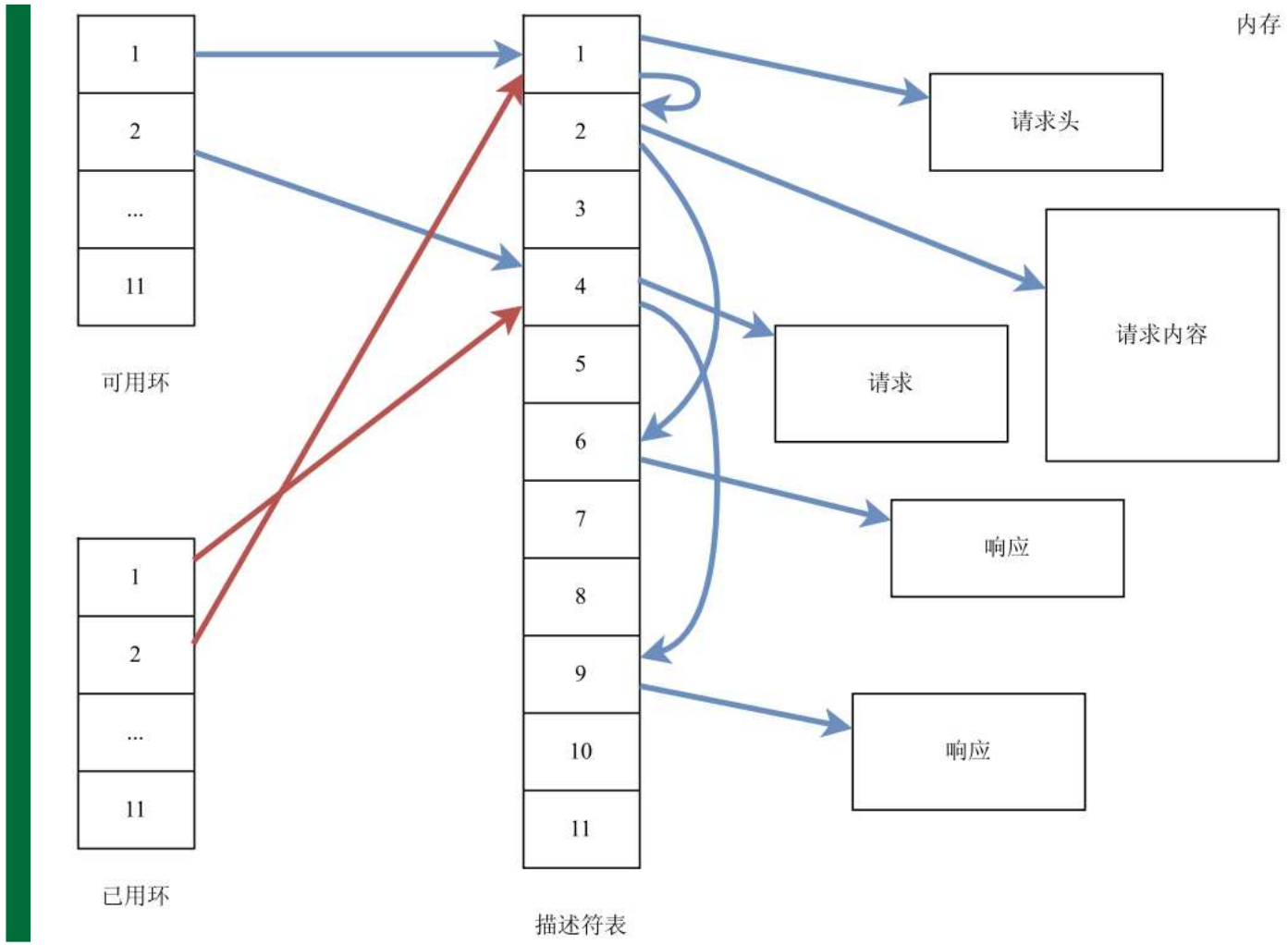
Table 4.1: MMIO Device Register Layout

Name Offset from base Direction	Function Description
MagicValue 0x000 R	**Magic value** 0x74726976 (a Little Endian equiv
Version 0x004 R	**Device version number** 0x2. **Note:** Legacy devices (see 4.
DeviceID 0x008 R	**Virtio Subsystem Device ID** See 5 Device Types for possible v depending on user's needs.
VendorID 0x00c R	**Virtio Subsystem Vendor ID**
DeviceFeatures 0x010 R	**Flags representing features the** Reading from this register returns *DeviceFeaturesSel* * 32 to (*Devic* 2.2 Feature Bits
DeviceFeaturesSel 0x014 W	**Device (host) features word sel** Writing to this register selects a se
DriverFeatures 0x020 W	**Flags representing device features understood and activated by the driver** Writing to this register sets 32 consecutive flag bits, the least significant bit depending on the last value written to *DriverFeaturesSel*. Access to this register sets bits *DriverFeaturesSel* * 32 to (*DriverFeaturesSel* * 32) + 31, eg. feature bits 0 to 31 if *DriverFeaturesSel* is set to 0 and features bits 32 to 63 if *DriverFeaturesSel* is set to 1. Also see 2.2 Feature Bits.
DriverFeaturesSel 0x024 W	**Activated (guest) features word selection** Writing to this register selects a set of 32 activated feature bits accessible by writing to *DriverFeatures*.
QueueSel 0x030 W	**Virtual queue index** Writing to this register selects the virtual queue that the following operations on *QueueNumMax*, *QueueNum*, *QueueReady*, *QueueDescLow*, *QueueDescHigh*, *QueueAvailLow*, *QueueAvailHigh*, *QueueUsedLow* and *QueueUsedHigh* apply to. The index number of the first queue is zero (0x0).
QueueNumMax 0x034 R	**Maximum virtual queue size** Reading from the register returns the maximum size (number of elements) of the queue the device is ready to process or zero (0x0) if the queue is not available. This applies to the queue selected by writing to *QueueSel*.
QueueNum 0x038 W	**Virtual queue size** Queue size is the number of elements in the queue. Writing to this register notifies the device what size of the queue the driver will use. This applies to the queue selected by writing to *QueueSel*.
QueueReady 0x044 RW	**Virtual queue ready bit** Writing one (0x1) to this register notifies the device that it can execute requests from this virtual queue. Reading from this register returns the last value written to it. Both read and write accesses apply to the queue selected by writing to *QueueSel*.
QueueNotify 0x050 W	**Queue notifier** Writing a value to this register notifies the device that there are new buffers to process in a queue. When VIRTIO_F_NOTIFICATION_DATA has not been negotiated, the value written is the queue index.

与设备的交互：虚拟队列

虚拟队列是VirtIO设备接受操作请求、返回结果的数据结构。驱动按照规定好的结构放置请求、进行连接（指针赋值），蓝线为驱动进行的连接，设备按其完成操作。

红线是设备返回结果时需要进行的连接。驱动按其读取结果。



第三部分 ▷▷

驱动实现

- 各设备特点
- 复用性

各设备特点

块设备

模式: 系统 → 驱动

与VirtIO操作模式一致，因此可以直接实现。

输入设备

模式: 驱动 → 系统

与VirtIO操作模式相反，需要先把请求全部发出，否则驱动无法返回事件。

网卡

模式: 驱动 → 系统

同左边。
需要传输大量数据，因此发出的请求需为数据分配空间。

复用性

使用Rust包（crate）的形式发布在github上。可以被操作系统直接引入。

使用时，操作系统需要为接口实现若干方法。

为了让开发者能够无缝切换，在导出的方法上尽力做到与不安全驱动保持一致。

第四部分 ▷▷

测试

- 功能测试
- 性能测试
- 总结

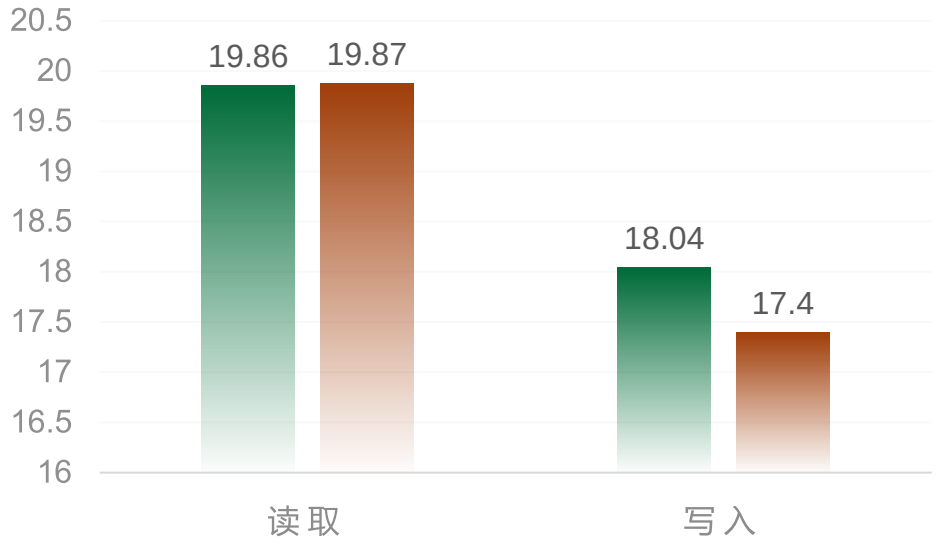
功能测试

- 为验证驱动实用性，在Alien系统中以隔离域形式加入该驱动。
- Alien 是一个着眼于使用语言机制提高操作系统的健壮性与安全性，并加入多种实验性新功能的微内核操作系统。它使用了多种设计模式和约束。如模块化OS各组件、严格的隔离机制、各部分的动态加载与替换等。
- 隔离域是Alien内的单元，需要满足故障隔离、内存隔离、代理调用等约束。满足之后，隔离域支持动态加载、无感恢复等特性。

块设备性能对比

绿色为设计的安全驱动。褐色为对比不安全驱动（rCore的VirtIO驱动）。

可以看出，两者性能没有明显差异。在写入时甚至略快于不安全驱动。



总结

通过展示驱动的设计，并对其进行多方面测试，可以看出安全驱动取代现有不安全驱动的巨大潜力。

Rust语言与其带来的新兴安全约束模式还有很多的可探索空间。希望本项目能够为基于语言机制的安全程序设计提供参考，推动系统编程领域的安全改进。

谢谢各位老师
敬请批评指正

Thanks for your listening

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

答辩人：罗博文

导　师：陆慧梅

时间：2024/5/27