

北京理工大学

本科生毕业设计(论文)

指导手册

rCore 模块化改进的设计与实现

Design and implementation of modular improvement of rCore

学院：计算机学院
专业：计算机科学与技术
班级：07111703
学生姓名：石文龙
学号：1120173592
指导教师：陆慧梅

2023 年 北京理工大学教务部制

北京理工大学

本科生毕业设计（论文）任务书

学生姓名	石文龙	学号	1120173592
学院	计算机学院	班级	07111703
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	自拟题目	题目性质	软件开发
题目	rCore 模块化改进的设计与实现		

一、题目内容

本课题主要研究如何对模块化的 rCore 操作系统进行改进。我们希望能发挥 Rust 语言中 workspace/crate/traits 的先进设计理念，重构并形成模块化的 rCore, 这样操作系统将以模块化/Traits 的形式呈现出来，我们可以按照模块化/Traits 实现的方法来修改其中的内容。并且为已经模块化的 rCore 操作系统的每一个模块添加用户态的单元测试。

二、任务要求

1、了解用 Rust 写 rCore 操作系统相关领域背景知识，需要知道各章节实现了什么功能，需要完成各章节的实验代码，需要知道在当前的组织形式下，如何迁移各章节的成果，如何修改各章节的实现并同步到后续所有章节；

2、在指导老师的指导下阅读国内外文献和自学相关知识，对利用 Rust 将 rCore 模块化进行研究和分析，并且为现有的模块化 rCore 的每个模块增加用户态单元测试。模块化的作用是：A、章节模块化：所有章节不再被 git 分支隔离开，而是只有逻辑上的隔离关系，后一章节能够以依赖库的形式继承前一章节的成果。B、实现模块化：能在所有章节中复用的代码形成单独的 crate 甚至 package, crates 之间在调用方面有层次依赖关系，crates 的粒度尽量小。C、系统调用接口模块化：系统调用的分发封装到一个 crate，使得添加系统调用的模式不是为某个 match 增加分支，而是实现

北京理工大学本科生毕业设计指导手册

一个分发库要求的 `trait` 并将实例传递给分发库。对每个模块添加用户态单元测试能够在其他开发者完成一个模块的接口实现的时候，能够更快速有效的判断自己的接口实现是否正确。

3、完成毕业设计（论文）外文翻译，锻炼跨文化交流的语言和书面表达能力，能就专业问题，在跨文化背景下进行基本沟通和交流；

4、完成毕业设计论文并提交相关软件以及文档。

5、毕业设计开发环境

Ubuntu 虚拟机

Rust 版本管理器 `rustup` 和 Rust 包管理器 `cargo`

QEMU 7.0 模拟器

VSCode

三、进度安排

A.学习并掌握 `rust` 语言；（第 1 周-第 2 周）

B.学习 RISC-V；（第 3 周）

C.对现有的模块化 `rCore` 进行分析；（第 4 周-第 6 周）

D.完成 `rCore` 的模块化改进，主要指实现各模块的用户态单元测试；（第 7 周-第 14 周）

E.完成毕业论文，提交软件及相关文档；（第 14 周-第 15 周）

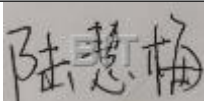
F.完成本科生毕业设计（论文）外文翻译；（第 1 周-第 15 周）

G.完成本科生毕业设计（论文）答辩；（第 1 周-第 15 周）

四、主要参考文献

杨德睿 单核环境的模块化 Rust 语言参考实现

五、指导教师签字：



2022 年 12 月 23 日

六、题目审核负责人意见

通过

签字:



2022 年 12 月 24 日

北京理工大学

北京理工大学

本科生毕业设计（论文）

开题报告

rCore 模块化改进的设计与实现

Design and implementation of modular improvement of rCore

学 院:	计算机学院
专 业:	计算机科学与技术
班 级:	07111703
学生姓名:	石文龙
学 号:	1120173592
指导教师:	陆慧梅

一、选题依据

（简述该选题的研究意义和背景，国内外研究概况和发展趋势等）

操作系统是计算机的灵魂，目前国外操作系统品牌几乎垄断了巨大的中国市场，其中在桌面端、移动端的市占率分别超过 94.75%、98.86%。根据 Gartner 的统计数据，2018 年中国的操作系统市场容量在 189 亿以上，其中国外操作系统品牌几乎在中国市场处于垄断地位。截至 2019 年 8 月，在中国的桌面操作系统市场领域，微软 Windows 的市占率 87.66%，苹果 OSX 的市占率为 7.09%，合计为 94.75%；在中国的移动操作系统市场领域，谷歌 Android 的市占率为 75.98%，苹果 iOS 的市占率为 22.88%，合计为 98.86%。

虽然当前中国的操作系统市场依旧是微软的 windows+intel 占据了主导地位，但是 windows 的闭源架构正面临着以 linux 为代表的开源操作系统的挑战。中国的操作系统国产化浪潮起源与二十世纪末，目前正依托于开源操作系统的开源生态以及政策东风正在快速崛起，涌现出了中标麒麟、银河麒麟、深度 Deepin、华为鸿蒙等各种国产操作系统。

所以在这个时期，我们进行操作系统的学习和研究是非常可行的。但是由于随着操作系统的功能实现的增多，其复杂程度也在增加；并且在对操作系统的学习过程中，因为各个章节是被 git 分支隔离开的，所以完成前一个章节的实验后，在下一个章节有关前一个章节的实现内容需要复制代码过来，同样的代码用一次就需要写一次，并且如果改了某一章节的内容，后续的所有章节相同的地方都需要重新改一边，代码的复用性非常不好。

rust 的模块化编程就能很好的解决上述问题，之前做不同章节的课后实验需要切换到对应的 lab 分支去写代码，现在只需要封装一个 crate 加入 workspace 就可以了，之前每个实验都会有大量的代码需要重复的写在每一个章节中，模块化之后这些重复的代码就可以直接引用了，大大提高了代码的复用性。防止修改了某一模块的内容，但是后续章节没有修改造成的错误出现，减少了开发者出错的可能性。

二、研究目标和内容

（研究目标、主要内容及关键问题等）

研究目标：在实验室工作的基础上，实现对于 rcore 内核模块化的改进与优化。主要完善实验室现有的模块化的 rcore 操作系统内核，针对现有的 rcore 操作系统内核模块化，提出了针对每一个章节模块的独立测试增加单元测试用例和在用户态对每一个模块进行单元测试的优化方向。

主要内容：首先需要了解 rust 语言的使用和 RISC-V 架构，并且完成用 rust 写操作系统内核的 5 个实验，理解实验室当前对内核模块化的成果，在当前成果的基础上各个章节的模块进行内核态和用户态的单元测试。

三、研究方案

（拟采用的研究方法、技术路线、实验方案及可行性分析等）

3.1 已有工作

目前实现的模块化,主要有在所有的章节中复用的代码形成了单独的 `package`, 各个章节对于这些复用的代码只需要在 `cargo.toml` 的依赖中添加上需要使用到的 `package` 就可以了, 不需要再像之前一样需要将这些已经写过的代码在每一章节中都重新写一遍。

成功利用了 `rust` 语言的 `workspace` 和 `crate`, 使得每一个章节是一个预期目标不同的 `package`, 做不同的章节的实验的时候, 只需要封装一个 `crate` 到对应的 `package` 中, 然后在运行的时候运行指定的 `package` 就可以了, 不需要像之前一样做不同章节的课后实验需要到不同的分支中写代码。

并且实现了系统调用接口的模块化, 即系统调用的分发封装到一个 `crate` 中, 这个 `crate` 就是 `syscall/src/kernel/mod.rs`。使得添加系统调用的模式不是为某个 `match` 增加分支, 而是实现一个分发库要求的 `trait` 并将实例传递给分发库。

3.2 改进方向

首先, 针对目前实验室的版本只是实现了操作系统内核模块化的基本功能的问题, 增加了每个章节的模块化完成之后的单元测试。增加了单元测试之后, 我们能够进行小而集中的测试能够在隔离的环境中一次测试一个模块或者测试模块的私有接口, 能够更加方便的检测出来是代码的哪个模块甚至是哪个函数出了问题。

继续完善 `rcore` 操作系统内核的模块化, 对已经存在的模块化进行自己的改进, 可以在分析了已有模块接口的优缺点的基础上, 只进行模块接口定义的改进; 也可以对模块的实现方法进行改进, 并且通过测试来分析自己的实现与已有模块的实现相比有哪些优缺点。

3.3 各章节单元测试

在 `Rust` 中一个测试函数的本质就是一个函数, 只是需要使用 `test` 属性进行标注或者叫做修饰, 测试函数被用于验证非测试代码的功能是否与预期一致。在测试的函数体里经常会进行三个操作, 即准备数据/状态, 运行被测试的代码, 断言结果。

在各章节的 `src` 目录下, 每个文件都可以创建单元测试。标注了 `#[cfg(test)]` 的模块就是单元测试模块, 它会告诉 `[Rust]` 只在执行 `cargo test` 时才编译和运行代码。在 `library` 项目中, 添加任意数量的测试模块或者测试函数, 之后进入该目录, 在终端中输入 `cargo test` 运行测试。但是, 我们的内核是一个 `no_std` 应用, 没有标准库, 而 `rust` 的测试框架会隐饰的调用 `test` 库, 而 `test` 库使依赖标准库的, 所以我们

的内核无法使用默认框架。

因此我们需要自定义测试框架，并且幸运的使 Rust 支持通过使用不稳定的自定义测试框架来代替默认的测试框架，而且该功能不需要额外的依赖库，因此在我们的 `no_std` 内核中可以使用。它的工作原理是收集所有标注了 `#[test_case]` 属性的函数，然后将这个测试函数的列表作为参数传递给用户指定的 `runner` 函数。因此，它实现了对测试过程的最大控制。

完成了自定义测试框架之后，我们现在在 `_start` 函数结束之后进入了一个死循环，在每次运行完 `cargo test` 后需要我们手动退出 QEMU，非常麻烦。幸运的是 QEMU 支持一种名为 `isa-debug-exit` 的特殊设备，它提供了一种从客户系统（`guest system`）里退出 QEMU 的简单方式。我们可以通过将配置关键字 `package.metadata.bootimage.test-args` 添加到我们的 `Cargo.toml` 中来达到目的。

要在控制台上查看测试输出，我们还需要以某种方式将数据从内核发送到宿主系统，发送数据的一个简单的方式是通过串行端口，QEMU 可将通过串口发送的数据重定向到宿主机的标准输出或是外部文件中。并且为了查看 QEMU 的串行输出，我们需要使用 `-serial` 参数将输出重定向到 `stdout`。同时为了在 `panic` 时使用错误信息来退出 QEMU，我们可以使用条件编译在测试模式下使用不同的 `panic` 处理方式。由于我们使用 `isa-debug-exit` 设备和串行端口来报告完整的测试结果，所以我们不再需要 QEMU 窗口了。我们可以通过向 QEMU 传递 `-display none` 参数来隐藏弹出的窗口。

完成上述功能后我们已经有了一个可以工作的测试框架了，我们可以为我们的 VGA 缓冲区实现创建一些测试。

3.4 实施技术方案所需的条件

Ubuntu 虚拟机

Rust 版本管理器 `rustup` 和 Rust 包管理器 `cargo`

QEMU 7.0 模拟器

VSCode

四、研究计划及进度安排

12.23-12.30

了解课题研究内容，撰写开题报告

12.30-1.10

实现现有的模块化 `rcore` 内核

1.10-1.17

测试 `console` 模块的 `test_log` 函数能否打印出设定的信息，测试 `print` 宏和 `println` 宏，并写实验文档

1.17-1.24

测试 `linker` 模块的 `fmt` 和 `next` 函数，`kernel-context` 模块的 `execute`, `new`, `aligned_size` 函数，`multislot_portal.rs` 中 `calculate_size`，`init_transit` 函数，`lib.rs` 中 `empty`, `user`, `thread`, `x_x_mut` 等读取修改函数，`move_next`, `execute`, `build_sstatus` 函数，

syscall 模块的 handle 函数

1.25-2.2

测试 kernel-alloc 中 alloc 函数, kernel-vm 中 mapper.rs 中 new, ans, arrive, meet, block 函数, mod.rs 中 new root_ppn root map_extern maptranslate cloneself fmt 函数, visitor.rs 中 new ans arrive meet block 函数。

2.3-2.10

测试 task-manager 中 id.rs 进程和线程 new from_usize get_usize 函数, proc_manage.rs 中 new find_next set_manager make_current_suspend make_current_exited add current get_task wait 函数, proc_rel.rs 中 new add_child del_child wait_any_child wait_child 函数, 以及线程中与进程相同的函数。

2.11-2.18

测试 easy-fs 中 bitmap.rs 中 decomposition new alloc dealloc 函数 block_cache.rs 中 new addr_of_offset get_ref get_mut readmodify sync get_block_cache block_cache_sync_all 函数, efs.rs 中 create open root_inode get_disk_inode_pos get_data_block_id alloc_inode alloc_data dealloc_data 函数, file.rs 中 new lenread_write readable writable read write 函数, layout.rs 中 fmt initialize is_valid initialize is_dir is_file data_blocks _data_blocks total_blocks blocks_num_needed get_block_id increase_size clear_size read_at write_at 函数和 DirEntry 结构中的函数

2.19-2.28

测试 signal-impl 的 default_action.rs 中 from into 函数, lib.rs 中 new fetch_signal fetch_and_remove, from_fork is_handling_signal set_action, get_action_ref update_mask handle_signals sig_return 函数, signal_set.rs 中 SignalSet 结构中的函数。

3.1-3.7

测试 sync 中 condvar.rs 中 new signal wait_no_sched, wait_with_mutex 函数, mutex.rs 中 new lock unlock 函数, semaphore.rs 中 new up down 函数, up.rs 中 new enter exit 函数 UPItrFreeCell 结构中 new exclusive_access exclusive_session 函数。

3.8-5.1

分析各模块接口的优缺点后修改模块的接口

5.1-6 月初

完成毕业论文, 参加答辩

五、创新点及预期研究成果

本项目基于实验室的 rcore 操作系统内核的模块化, 创新点在于增加了每一章的用户态和内核态的单元测试。

研究成果为: 一篇毕业论文, 一份软件成果。

六、参考文献

[1]杨德睿 单核环境的模块化 Rust 语言参考实现

[2]guomeng.2021 年操作系统的商业化应用 国内操作系统现状与前景分析.//2021 年 10 月 20 日中研网

[3]洛佳 使用 Rust 编写操作系统（四）：内核测试.//2019 年 11 月 07 日知乎

七、指导教师意见

同意该开题内容

签字:



2023 年 5 月 17 日

成绩: , 占比: 0.00%

八、开题审核负责人意见

同意

签字:



2023 年 5 月 18 日

北京理工大学

本科生毕业设计(论文)

中期报告

rCore 模块化改进的设计与实现

Design and implementation of modular improvement of rCore

学院:	计算机学院
专业:	计算机科学与技术
班级:	07111703
学生姓名:	石文龙
学号:	1120173592
指导教师:	陆慧梅

一、毕业设计（论文）主要研究内容、进展情况及取得成果

主要研究内容：

在实验室工作的基础上，实现对于应用于 Rust 语言的 rcore 内核模块化的改进与优化主要工作包括增加系统中的每一个模块的单元测试。

进展情况：

1. kernel-context 模块实现了内核上下文的控制，主要的结构包括：

LocalContext: 线程上下文。

PortalCache: 传送门缓存。

ForeignContext: 异界线程上下文即不在当前地址空间的线程上下文。

PortalText: 传送门代码。

MultislotPortal: 包含多个插槽的异界传送门。

lib.rs

```
pub struct LocalContext {  
    sctx: usize,  
    x: [usize; 31],  
    sepc: usize,  
    /// 是否以特权态切换。  
    pub supervisor: bool,  
    /// 线程中断是否开启。  
    pub interrupt: bool,  
}
```

该结构包含 14 个方法，其分别为：
`pub const fn empty()` 该方法的作用是创建空白上下文。
`pub const fn user(pc: usize)` 该方法的作用是初始化指定入口的用户上下文，切换到用户态时会打开内核中断。
`pub const fn thread(pc: usize, interrupt: bool)` 该方法的作用是初始化指定入口的内核上下文。
`pub fn x(&self, n: usize)` `pub fn a(&self, n: usize)` `pub fn ra(&self)` `pub fn sp(&self)` `pub fn pc(&self)` 该方法的作用分别是读取用户通用寄存器；读取用户参数寄存器；读取用户栈指针；读取用户栈指针；读取当前上下文的 pc。
`pub fn x_mut(&mut self, n: usize)` `pub fn a_mut(&mut self, n: usize)` `pub fn sp_mut(&mut self)` `pub fn pc_mut(&mut self)` 该方法的作用分别是修改用户通用寄存器；修改用户参数寄存器；修改用户栈指针；修改上下文的 pc。
`pub fn move_next(&mut self)` 该方法的作用是将 pc 移至下一条指令。
`pub unsafe fn execute(&mut self)` 该方法的作用是执行此线程，并返回 sstatus,将修改 sscratch、sepc、ssatus 和 stvec。

foreign/mod.rs

```
pub struct PortalCache {  
    a0: usize,      // (a0) 目标控制流 a0  
    a1: usize,      // 1*8(a0) 目标控制流 a1      (寄存，不用初始化)  
    satp: usize,    // 2*8(a0) 目标控制流 satp  
    sstatus: usize, // 3*8(a0) 目标控制流 sstatus
```

```

sepc: usize,      // 4*8(a0) 目标控制流 sepc
stvec: usize,     // 5*8(a0) 当前控制流 stvec    (寄存, 不用初始化)
sscratch: usize, // 6*8(a0) 当前控制流 sscratch (寄存, 不用初始化)

```

```

}

```

该结构是传送门缓存,即映射到公共地址空间,在传送门一次往返期间暂存信息。该结构的方法一共有 2 种, 分别是 `pub fn init(&mut self, satp: usize, pc: usize, a0: usize, supervisor: bool, interrupt: bool)` 该方法的作用是初始化传送门缓存。 `pub fn address(&mut self)` 该方法的作用是返回缓存地址。

```

pub struct ForeignContext {
    /// 目标地址空间上的线程上下文。
    pub context: LocalContext,
    /// 目标地址空间。
    pub satp: usize,
}

```

该结构的作用是异界线程上下文,即不在当前地址空间的线程上下文。该结构一共有 1 中方法, 其是 `pub unsafe fn execute(&mut self, portal: &mut impl ForeignPortal, key: impl SlotKey)` 该方法的作用是执行异界线程。

```

struct PortalText(&'static [u16]);

```

该结构的作用是传送门代码。该结构一共有 3 种方法, 分别是: `pub fn new()` `pub fn aligned_size(&self)` `pub unsafe fn copy_to(&self, address: usize)`

```

foreign/multislot_portal.rs

```

```

pub struct MultislotPortal {
    slot_count: usize,
    text_size: usize,
}

```

该结构的含义是包含多个插槽的异界传送门。该结构有 2 个方法, 分别是: `pub fn calculate_size(slots: usize)` 该方法的作用是计算包括 slots 个插槽的传送门总长度。 `pub unsafe fn init_transit(transit: usize, slots: usize)` 该方法的作用是初始化公共空间上的传送门。其中参数 transit 必须是一个正确映射到公共地址空间上的地址。

kernel-context 模块的测试方法就是调用 LocalContext 结构,然后一次调用 LocalContext 结构的几个方法:测试 empty().测试 user()函数,初始化指定入口的用户上下文。thread()函数,初始化指定入口的内核上下文。测试读取类函数.测试 move_next,将 pc 移至下一条指令。测试修改类函数.然后使用 assert_eq()函数来比较方法的返回值和预期的结果是否一样,若结果一样则说明函数方法没有问题,测试通过。

2. linker 板块为内核提供链接脚本的文本, 以及依赖于定制链接脚本的功能。build.rs 文件可依赖此板块, 并将 [SCRIPT] 文本常量写入链接脚本文件定义内核入口, 即设置一个启动栈, 并在启动栈上调用高级语言入口。

```

macro_rules! boot0

```

KernelLayout 结构: 代表内核地址信息;

KernelRegion 结构：内核内存分区。

KernelRegionIterator 结构：内核内存分区迭代器。

KernelLayout 的结构为： `pub struct KernelLayout { text: usize, rodata: usize, data: usize, sbss: usize, ebss: usize, boot: usize, end: usize, }` 该结构有 6 个方法，分别为 `pub fn locate()` 该方法作用为定位内核布局。 `pub const fn start(&self)` 该方法作用为得到内核起始地址。 `pub const fn end(&self)` 该方法作用为得到内核结尾地址。 `pub const fn len(&self)` 该方法作用为得到内核静态二进制长度。 `pub unsafe fn zero_bss(&self)` 该方法作用为清零 .bss 段。 `pub fn iter(&self)` 该方法作用为得到内核区段迭代器。

KernelRegion 结构为： `pub struct KernelRegion { /// 分区名称。 pub title: KernelRegionTitle, /// 分区地址范围。 pub range: Range, }` 该结构的含义是内核内存分区。该结构存在 `fmt` 方法。 `fn fmt(&self, f: &mut fmt::Formatter<_>)` 该方法的作用是使用给定的格式化程序格式化值。

KernelRegionIterator 结构为： `pub struct KernelRegionIterator<'a> { layout: &'a KernelLayout, //内核内存分区名称 next: Option, }` 该结构的含义是内核内存分区迭代器。该结构存在 `next` 方法。 `fn next(&mut self)` 该方法的作用是得到迭代器中下一位的值。

app.rs

AppMeta：应用程序元数据。

AppIterator：应用程序迭代器。

AppMeta 结构为： `pub struct AppMeta { base: u64, step: u64, count: u64, first: u64, }` 该结构的含义是应用程序元数据。该结构有 2 个方法，分别为： `pub fn locate()` 该方法的作用是定位应用程序。 `pub fn iter(&'static self)` 该方法的作用是遍历链接进来的应用程序。

AppIterator 结构为： `pub struct AppIterator { meta: &'static AppMeta, i: u64, }` 该结构的含义是应用程序迭代器。该结构有一个 `next` 方法： `fn next(&mut self)` 该方法的作用是对应用程序进行迭代。

定义了实现 VmMeta 特征的 SV39 结构和实现了 PageManager 特征的 Sv39Manager 结构体。

需要赋值一个物理页 `range: Range`;

linker 模块测试了 KernelLayout 结构的集中方法,首先需要构建依赖环境,然后调用 KernelLayout 结构的方法,并且对内核内存分区迭代器 KernelRegionIterator 结构的测试,最后使用 `assert_eq()`函数将方法的返回值和预期结果进行比较,如果比较通过,则测试也通过。

3. kernel-vm 模块的主要内容是内核虚拟存储的管理。

space/mod.rs

AddressSpace：地址空间结构。

`pub struct AddressSpace { /// 虚拟地址块 pub areas: Vec<_, page_manager: M, }` 该结构共有 7 个方法,分别为： `pub fn new()` 该方法的作用是创建新地址空间。 `pub fn root_ppn(&self)` 该方法的作用是得到地址空间根页表的物理页号。 `pub fn root(&self)` 该方法的作用是得到地址空间根页表 `pub fn map_extern(&mut self,`

range: Range, pbase: PPN, flags: VmFlags) 该方法的作用是向地址空间增加映射关系。 pub fn map(&mut self, range: Range, data: &[u8], offset: usize, mut flags: VmFlags,) 该方法的作用是分配新的物理页, 拷贝数据并建立映射。 pub fn translate(&self, addr: VAddr, flags: VmFlags) 该方法的作用是检查 flags 的属性要求, 然后将地址空间中的一个虚地址翻译成当前地址空间中的指针。 pub fn cloneself(&self, new_addrspace: &mut AddressSpace) 该方法的作用是遍历地址空间, 将其中的地址映射添加进自己的地址空间中, 重新分配物理页并拷贝所有数据及代码。

space/mapper.rs

```
pub(super) struct Mapper<'a, Meta: VmMeta, M: PageManager> {  
    space: &'a mut AddressSpace,  
    range: Range,>,  
    flags: VmFlags,  
    done: bool,  
}
```

该结构有 5 个方法, 分别是: pub fn new(space: &'a mut AddressSpace, range: Range, flags: VmFlags,) 该方法的作用是创建一个新的 Mapper。 pub fn ans(self) 该方法的作用是得到 Mapper 结构的 done 值。 fn arrive(&mut self, pte: &mut Pte, target_hint: Pos) fn meet(&mut self, _level: usize, pte: Pte, _target_hint: Pos,) fn block(&mut self, _level: usize, pte: Pte, _target_hint: Pos)

kernel-vm 模块的测试,首先需要构建依赖环境,即要定义并实现一个满足 VmMeta 特征的结构 SV39,然后要定义并实现满足 PageManager 特征的结构 SV39Manage; 最后, 还需要定义内核地址信息 KernelLayout, 并且实现 KernelLayout 结构的方法; 至此,kernel-vm 模块测试所需要的的依赖环境构建完成。

测试的时候, 首先自己初始化一个 KernelLayout 结构, 然后调用 AddressSpace 结构和 Mapper 结构的方法, 最后将预期结果和实际结果进行比较, 如果两者相符, 则说明 kernel-vm 模块中的 mapper 模块测试成功; 在测试 space 是可以依次测试: 创建新地址空间。地址空间根页表的物理页号。地址空间根页表。向地址空间增加映射关系。检查 flags 的属性要求, 然后将地址空间中的一个虚地址翻译成当前地址空间中的指针。遍历地址空间, 将其中的地址映射添加进自己的地址空间中, 重新分配物理页并拷贝所有数据及代码。并且在每次调用之后, 将预期结果和实际结果进行比较, 如果结果相同, 则 space 模块测试完成。

4. task-manage 模块的作用是实现父进程, 子进程和线程之间的调度管理。

任务 id 类型, 自增不回收, 任务对象之间的关系通过 id 类型来实现

ProcId

ThreadId

CoroId 结构 ProcId 的方法有 3 个, 分别是 new(),from_usize(),get_usize;

new()方法创建了一个进程编号自增的进程 id 类型,

from_usize()根据输入的 usize 类型参数可以获得一个以参数为 id 的 ProcId,

get_usize()方法需要输入的参数是一个 ProcId 的引用, 返回该 ProcId 结构对应的 id。

结构 `ThreadId` 的方法与 `ProcId` 的方法相同。

任务对象管理 `manage trait`，对标数据库增删改查操作

`insert`

`delete`

`get_mut`

任务调度 `schedule trait`，队列中保存需要调度的任务 `Id`

`add`: 任务进入调度队列

`fetch`: 从调度队列中取出一个任务

封装任务之间的关系，使得 `PCB`、`TCB` 内部更加简洁

`ProcRel`: 进程与其子进程之间的关系

`ProcThreadRel`: 进程、子进程以及它地址空间内的线程之间的关系 `ProcRel` 结构包含父进程 `id`，子进程列表和已经结束的进程列表。`ProcRel` 结构有 `new()` 方法，`add_child()`、`del_child()`、`wait_any_child()`、`wait_child()` 5 种方法。

`new()` 方法需要输入父进程 `ProcId`，返回一个 `ProcRel` 结构，其中父进程 `ProcId` 是输入的参数，子进程列表和已经结束的进程列表使新创建的动态数组。

`add_child()` 方法的参数是一个 `ProcRel` 的可变引用和一个子进程 `ProcId`，该方法的作用是将参数子进程 `id` 放入到输入的 `ProcRel` 的子进程列表中。

`del_child()` 方法的参数有一个 `ProcRel` 的可变引用、一个子进程 `ProcId` 和一个退出码 `exit_code`，该方法的作用是：令子进程结束，子进程 `Id` 被移入到 `dead_children` 队列中，等待 `wait` 系统调用来处理。

`wait_any_child()` 方法的参数是一个 `ProcRel` 的可变引用，该方法的作用是：等待任意一个结束的子进程，直接弹出 `dead_children` 队首，如果等待进程队列和子进程队列为空，返回 `None`，如果等待进程队列为空、子进程队列不为空，则返回 `-2`。

`wait_child` 方法的参数有一个 `ProcRel` 的可变引用和一个子进程 `ProcId`，该方法的作用是：等待特定的一个结束的子进程，弹出 `dead_children` 中对应的子进程，如果等待进程队列和子进程队列为空，返回 `None`，如果等待进程队列为空、子进程队列不为空，则返回 `-2`。

封装任务之间的调度方法

`PManager`: 管理进程以及进程之间的父子关系

`PThreadManager`: 管理进程、子进程以及它地址空间内的线程之间的关系

`PManager` 结构为：

```
pub struct PManager + Schedule> { // 进程之间父子关系
rel_map: BTreeMap, // 进程对象管理和调度
manager: Option, // 当前正在运行的进程 ID
current: Option, phantom_data: PhantomData
}, }
```

该结构办函的方法共有 9 个，分别为：

```
pub const fn new() -> Self
```

 此方法用于新建 `PManager`

```
pub fn find_next(&mut self) -> Option<&mut P>
```

 此方法用于找到一个进程

```
pub fn set_manager(&mut self, manager: MP)
```

 此方法用于设置 `manager`

pub fn make_current_suspend(&mut self) 此方法用于阻塞当前进程 pub fn make_current_exited(&mut self, exit_code: isize) 此方法用于结束当前进程，只会删除进程的内容，以及与当前进程相关的关系 pub fn add(&mut self, id: ProcId, task: P, parent: ProcId) 此方法用于 添加进程，需要指明创建的进程的父进程 Id pub fn current(&mut self) -> Option<&mut P> 此方法用于获取当前进程 #[inline] pub fn get_task(&mut self, id: ProcId) -> Option<&mut P> 此方法用于获取某个进程 pub fn wait(&mut self, child_pid: ProcId) -> Option<(ProcId, isize)> 此方法用于 wait 系统调用，返回结束的子进程 id 和 exit_code，正在运行的子进程不返回 None，返回 (-2, -1)

测试 id 模块 ProcId 结构时，只需要依次调用 ProcId 模块里各结构的方法，然后将预期结果和实际结果进行比较就可以了。proc_rel 模块和 id 模块的测试方法基本一样，只需要依次：创建一个进程时同时创建进程关系；添加子进程前测试等待子进程结束的函数；添加子进程；测试等待子进程结束的函数；子进程结束，子进程 Id 被移入到 dead_children 队列中；测试等待子进程结束的函数；最后将与其结果和实际结果进行比较就可以了。

测试 proc_manage 模块里的 PManager 结构的时候，首先需要构建需要的依赖环境，即定义一个进程结构 Process，并且实现 Process 的 new() 方法，然后定义一个 ProcManager 结构，ProcManager 结构需要实现 Manage 特征和 Schedule 特征，至此，proc_manage 模块的依赖环境完成了，然后就可以依次：新建 PManager；设置 manager；添加进程；获取指定进程；并且将预期结果和实际结果进行比较。

二、存在的问题和拟解决方案

1. console 模块： 需要实现 put_char 函数； 在 Cargo.toml 里直接添加 sbi-rt 会在测试的时候直接关于寄存器报错，自己实现 sbi 的 console_putchar 在 asm 中关于寄存器 x10 会报错：invalid register x10: unknown register 无效寄存器 x10：未知寄存器。
2. kernel-alloc 模块： 完成了 init 函数的测试 测试 transfer() 时，参数正确还是失败，准备分开测试一下其中的代码。Heap::new() 创建的是空分配器，总容量为 0，所以分配失败。测试 alloc() 时，会运行 handle_alloc_error(layout)。
3. sync 模块： 测试 up.rs 模块时，调用 sstatus::read().sie() 函数会出错，测试 exit 方法使要避免 nested_level 为 0 的情况，否则会发生错误。测试 condvar.rs 时测试 signal() 方法前需要向新建的 Condvar 里添加数据否则会 panic。
4. easy-fs 模块 对于实现 BlockDevice 特征的结构定义成功了，但是在测试的时候会报错：无效内存引用。
5. signal-imple 模块： 在进行测试的时候会对 kernel-context 的线程切换核心部分报错 unsafe extern "C" fn execute_naked() { core::arch::asm!() }

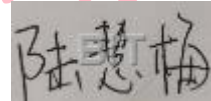
三、下一步研究任务与进度安排

- 4.6-4.15 使完成了部分的4个模块能够进行演示。
4.16-4.23 完成 easy-fs 模块的测试。
4.24-4.30 完成 syscall 和 signal-imple 模块的测试。

四、指导教师意见

石文龙同学按照毕设的工作安排有序地推进论文工作。后续希望更加有效地完成剩余任务。

签字:



2023年4月22日

成绩: , 占比: 0.00%

五、中期审核负责人意见

通过

签字:



2023年5月22日

毕业设计（论文）指导教师评语表

指导教师对毕业设计（论文）的评语：

石文龙同学毕设的论文题目是rCore模块化改进的设计与实现，希望能够完成rCore操作系统的模块化，实现代码的隔离和复用，降低操作系统课程实验的难度。论文的选题具有很强的实用价值。

毕业论文的主要工作包括相关基础知识的学习、工作环境的配置、对已有 rCore 模块化的工作进行分析；将 rCore 的代码从一个仓库拆分到不同的 github 仓库中；对各个模块进行用户态单元测试；最后也给出了将各个独立模块组合起来的方法。

石文龙同学很好地完成了毕业设计的任务，虽然刚开始困难重重，但他克服了诸多困难，终于啃下了一个个模块的独立测试并完成了整合。论文结构合理，引用规范，设计方案恰当，有一定工作量，过程规范。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。

是否有校外指导教师：○是 ●否；若选择“是”，校外指导教师对毕业设计（论文）的评语：

北京理工大学本科生毕业设计（论文）评语表

成绩： ， 占比 0.00%

指导教师

陆慧楠

2023年5月22日

北京理工大学

毕业设计（论文）匿名评阅评语表 1

学生姓名	石文龙	学号	1120173592
学院	计算机学院	专业	计算机科学与技术
题目	rCore 模块化改进的设计与实现		
评阅结果	中 (B) 77.0		
<p>评语：</p> <p>选题针对教学操作系统模块化改进问题，具有一定的应用价值，论文介绍了 rCore 模块化相关领域研究现状。</p> <p>论文阐述了 Rust 编程和 RISC-V 处理器的相关内容，配置了实验所需要的环境，对实验室模块化的工作进行分析。将操作系统的功能模块存放到不同的 github 仓库中，在只有一个模块的情况下也能够进行用户态单元测试，降低了其他人完成此实验的工作难度。</p> <p>论文结构合理，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。</p>			
<p>评阅人：</p> <p>2023 年 5 月 21 日</p>			

毕业设计（论文）匿名评阅评语表 2

学生姓名	石文龙	学号	1120173592
学院	计算机学院	专业	计算机科学与技术
题目	rCore 模块化改进的设计与实现		
评阅结果	及格(C) 67.0		
<p>评语：</p> <p>选题针对 rCore 模块化改进设计与实现问题，具有一定的研究意义，论文没有介绍相关领域研究现状。本课题论文分析了 rCore 各个模块功能，设计并完成了 rCore 模块化拆分，并进行了单元测试实验。整体论文结构合理，引用规范，研究方案恰当，有一定的工作量，且过程规范，表明学生具有较为扎实的专业基础知识和综合运用能力，无法界定具备工程实现能力，论文基本达到本科毕业设计对应的毕业要求，同意该学生参加论文答辩。</p>			
<div>评阅人：</div> <div>2023 年 5 月 22 日</div>			

北京海工