華東師範大學
EAST CHINA NORMAL UNIVERSITY

proj0 – 面向操作系统课程的操作系统竞赛和实验

题目二：适合本校特点的实验指导教程

基于Rust与xv6的操作系统教学方案设计与实现

汇报队伍：SEIOS 队员：郑子攸，叶晨皓，周恒如 指导老师：郭建

Contents

总览

華東師範大學
EAST CHINA NORMAL UNIVERSITY

華東師範大學
EAST CHINA NORMAL UNIVERSITY

第一部分

项目介绍

1.1 项目简介

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- **Rust语言**是目前操作系统研究的重要方向，但现有的操作系统课程以C语言实现为主
- 本项目基于xv6-riscv-rust实现了一套操作系统实践课程，主要完成了如下工作
  - **系统内核架构分析与核心模块注释编写**
  - **Rust语言重写用户程序及测试用例**
  - **自动化评测方案移植**
  - **实验指导手册及参考实现编写**
- 对内核核心模块（内存管理，进程管理，文件系统）编写了**完善的文档注释**
- 使用**Rust重写部分xv6的用户程序以及测试用例**，计划后续使用Rust重写所有测试用例
- 基于**Rust语言实现了用户线程库**，用于设计多线程实验，**填补xv6中无法进行多线程实验的空缺**
- 移植xv6的评测方案，实现了**Rust环境下的学生实验全自动评测**
- 参考xv6的学生实验，已完成**6个实验的移植与实验指导手册及参考实现的编写**，计划后续完成全部14个实验移植

1.2 项目目标

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 填补基于Rust语言的教学操作系统课程目前存在的空白，**提供一种同时学习Rust与操作系统的可行路径**
- 为系统内核中的所有函数添加文档注释，**实现注释100%覆盖**，提升代码的可读性
- 针对操作系统的不同部分**设计难度由浅入深的实验题**，帮助学生逐步学习Rust语言与操作系统
- **设计完善的测试用例与测试脚本**，**实现自动化评测**，简化实验检查步骤并提高评分公平性
- **编写实验指导手册**，提供每一个实验题的题干、提示、参考实现以及详细的实验解答

華東師範大學
EAST CHINA NORMAL UNIVERSITY

第二部分

前期调研

2.1 调研内容

華東師範大學
EAST CHINA NORMAL UNIVERSITY

学校/项目	编程语言	实验难度	工具链和平台	实验内容与教学价值概要
MIT (xv6)	C	覆盖完善	RISC-V (QEMU模拟器)	覆盖完整操作系统功能（系统调用、内存管理、中断、文件系统等），实验内容深入全面，适合高年级本科生，强调OS完整性与工程实践能力。
清华大学 (rCore)	Rust	中等	RISC-V (QEMU, K210开发板)	从零构建类Unix OS，全面覆盖内核启动、内存/进程管理、文件系统及驱动；适合初学者，以Rust语言强调内存安全与现代OS理念。
中科院软件所 (RVOS)	C	较低	RISC-V (QEMU模拟器、 Docker环境)	实验聚焦于核心OS机制（引导启动、任务管理、中断处理等），难度梯度小，适合初学者建立OS基本概念与实践技能。
南京大学 (JYY OS)	C	较高	抽象机平台 (QEMU模拟 x86/RISC-V)	涉及深入的OS理论和实践（线程/进程管理、文件系统设计与实现），实验强度高、理论性强，适合高年级本科生与研究生深入研究。
华中科大 (PKE)	C	中等	RISC-V (Spike精确模拟, QEMU与FPGA硬件)	分模块逐步构建“代理内核”，覆盖特权级切换、内存管理、设备驱动与进程调度；强调软硬结合的工程实践，适合本科生掌握实际应用场景下的内核开发能力。

2.2 主要相似工作 – MIT (xv6)

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- MIT 6.828 课程通过一系列以 xv6 操作系统为平台的实验，系统地覆盖了现代操作系统的核心功能，包括用户程序与系统调用接口、异常与中断处理、虚拟内存与页表管理、内存优化策略（如惰性分配与写时复制）、线程与同步机制、文件系统结构以及设备驱动与网络通信。
- 每个实验从实际功能出发，要求学生亲自阅读、修改和扩展内核代码，循序渐进地构建完整的操作系统理解模型。这种“由简入深”的实践方式不仅锻炼了学生系统级编程能力，更加深了其对操作系统设计思想、内核机制和性能权衡的掌握，是操作系统教育中极具代表性的训练体系。这门课程共包含11组实验。

2.2 主要相似工作 – MIT (xv6)

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- **Lab1: Utilities** 实现常见 UNIX 工具掌握用户态程序构建与系统调用接口，奠定操作系统基础编程能力。
- **Lab2: System Calls** 添加新系统调用深入理解系统调用机制及用户态与内核态之间的交互与信息传递。
- **Lab3: Page Tables** 多级页表打印与内核页表隔离，深入掌握虚拟地址映射与页表管理策略。
- **Lab4: Traps** 实现调用栈回溯与用户级定时器功能，理解 trap 机制与用户态中断控制流切换。
- **Lab5: xv6 Lazy Page Allocation** 实现按需分配机制，掌握缺页异常处理与虚拟内存管理的延迟映射策略。
- **Lab6: Copy-on-Write Fork for xv6** 改造 fork() 为写时复制机制，深入掌握内存共享优化与页错误处理技术。
- **Lab7: Multithreading** 构建用户级线程库并进行并发测试，掌握线程切换、同步机制及并发程序设计。
- **Lab8: Locks** 实现并测试多核锁机制，强化对内核同步、竞态与死锁问题的理解与解决能力。
- **Lab9: File System** 扩展 xv6 文件系统支持大文件与符号链接，掌握文件索引结构与路径解析机制。
- **Lab10: Mmap** 实现内存映射文件访问，深入理解文件 I/O 与虚拟内存系统的集成原理。
- **Lab11: Networking** 开发网卡驱动与基础网络协议支持，掌握内核网络栈、DMA 通信与中断处理机制。

2.3 主要相似工作 – 清华大学 (rCore)

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- rCore 操作系统实验通过覆盖日志输出、系统调用、安全检查、进程调度、虚拟内存、文件系统、进程通信、并发同步及图形界面等核心模块，系统地培养学生用 Rust 开发操作系统的能力，强化其对内核结构和抽象机制的理解。
- 实验内容具备较强的工程实践性，能帮助学生掌握 Rust 在内核中的高安全性与高性能用法。然而，其入门门槛较高，难度递进陡峭，且实验文档在关键实现细节与调试指导上略显不足，容易使初学者在复杂模块中卡顿，适合有一定操作系统基础知识的专业学生进行学习。

2.3 主要相似工作 – 清华大学 (rCore)

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- **Lab1：彩色化 LOG** 通过实现彩色日志输出和内存布局显示，掌握操作系统的输出机制与调试可视化设计。
- **Lab2：sys_write 安全检查** 通过为 sys_write 添加地址合法性检查，理解系统调用中的内存隔离与安全边界。
- **Lab3：获取任务信息** 实现任务状态查询系统调用，掌握任务控制块扩展与内核任务监控机制。
- **Lab4：重写 sys_get_time** 在适配虚拟内存的基础上实现 mmap/munmap 系统调用，深入理解内存映射与权限控制。
- **Lab5：进程创建&&stride 调度算法** 通过实现新进程创建方式与 stride 调度算法，掌握进程生命周期管理与调度策略设计。
- **Lab6：硬链接** 实现硬链接与 inode 引用计数机制，深入理解文件系统中的元数据管理与目录结构映射。
- **Lab7：进程通信：邮箱** 通过构建邮箱机制实现进程间通信，掌握内核缓冲区管理与非阻塞通信接口设计。
- **Lab8：银行家算法** 通过用户态同步模型与 eventfd 系统调用实现，掌握并发控制与异步事件通知机制。
- **Lab9：支持图形显示的应用** 实现 GPU 驱动与帧缓冲设备，掌握图形显示路径与图形界面下的设备驱动开发方法。

華東師範大學
EAST CHINA NORMAL UNIVERSITY

第三部分

技术方案

3.1 内核与文档注释

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- **基于Rust语言的系统内核**
  - 本项目的主要工作在于教学方案设计，因此没有重新实现类xv6的内核，内核部分参考了xv6-riscv-rust的实现
  - 系统内核绝大部分使用Rust语言实现，小部分使用RISC-V汇编实现
  - 系统结构与xv6保持一致，提供相同的系统调用接口，由内存管理、进程管理和文件系统等模块组成
- **内核文档注释**
  - 使用Rust文档注释格式，为系统内核核心模块（内存，进程，文件）的所有函数、结构体以及全局变量编写注释
  - 通过执行cargo doc指令，能够自动生成HTML格式的内核文档，便于学生快速查找阅读
  - 函数的文档注释内容完善，包含功能说明，参数，返回值，可能出错的情况
    - unsafe函数还额外包含安全性说明
  - 文档的编写是由手工编写与大模型生成相结合的，所有文档均经过人工查验，确保其内容的正确性

3.2 测试用例与测试脚本

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- **用户程序与测试用例**
  - 本项目参考了xv6的内核测试方案，通过用户程序实现对系统内核的测试，执行方法为在shell中执行usertests
  - 由于系统调用接口保持一致，参考并移植了部分C语言版本的xv6用户程序以及测试用例
  - 大部分用户程序与测试使用C语言实现，同时也实现了部分Rust版本的用户程序
  - 计划后续将C语言的用户程序全部使用Rust重新编写
- **自动化测试脚本**
  - 本项目参考了xv6的自动化测试脚本，使用Python实现了对学生实验的全自动评测与分数计算
  - 该脚本将评测点与评测库分离，能够自动收集评测点，灵活调整评测方案
  - 自动编译系统内核并重置文件系统，能够与QEMU-GDB通信，追踪内核输出，保证评测有效性

3.3 实验指导手册与参考实现

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- **实验指导手册与参考实现**
  - 实验指导手册提供了实验背景、实验题干、实现思路以及具体提示，可以通过向学生提供不同的内容以调整实验难度
  - 实验的设计综合了MIT 6.828课程近五年的所有实验，参考了其中难度合适的实验
  - 针对每个不同实验均编写的具体的实验指导，详细内容在后文中完整展现
  - 每个实验都提供了参考实现，以及参考实现的代码解析，详细内容在后文中完整展现
  - 参考实现均经过实机运行，能够在测试脚本下得到满分

華東師範大學
EAST CHINA NORMAL UNIVERSITY

第四部分

系统内核架构

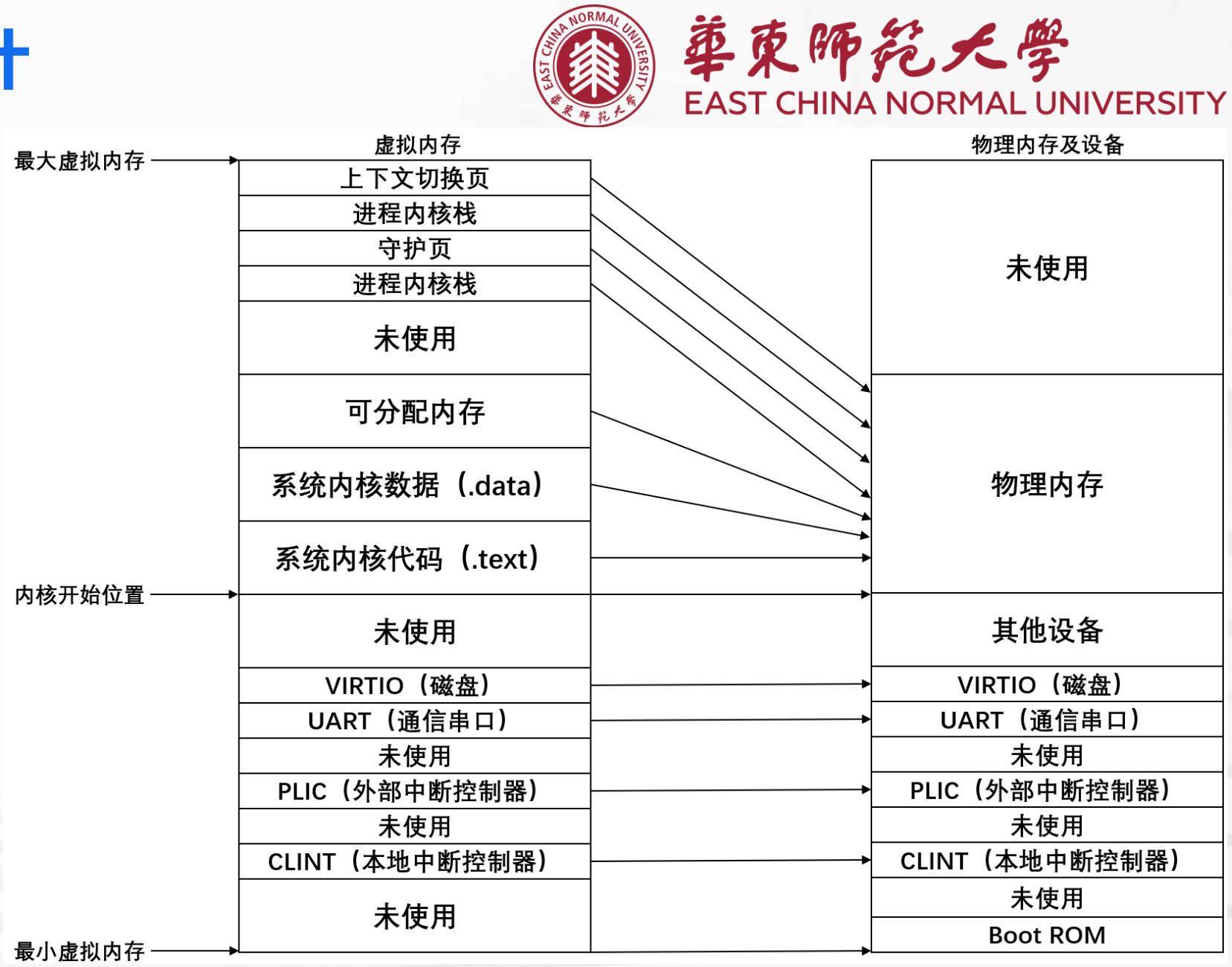
4.1 内核整体架构

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 内存管理模块：物理内存管理，虚拟内存管理
- 进程管理模块：中断处理，进程调度，进程控制，进程同步
- 文件系统模块：物理文件系统，虚拟文件系统

<table>
<tr><td rowspan="4">内核</td><td colspan="3">系统调用接口</td></tr>
<tr><td>内存管理模块</td><td>进程管理模块</td><td>文件管理模块</td></tr>
<tr><td>虚拟内存管理</td><td>进程控制与同步</td><td>虚拟文件系统</td></tr>
<tr><td>物理内存管理</td><td>中断处理与进程调度</td><td>物理文件系统</td></tr>
<tr><td>硬件驱动</td><td>RISC-V 页表</td><td>中断控制器</td><td>串口与磁盘驱动</td></tr>
</table>

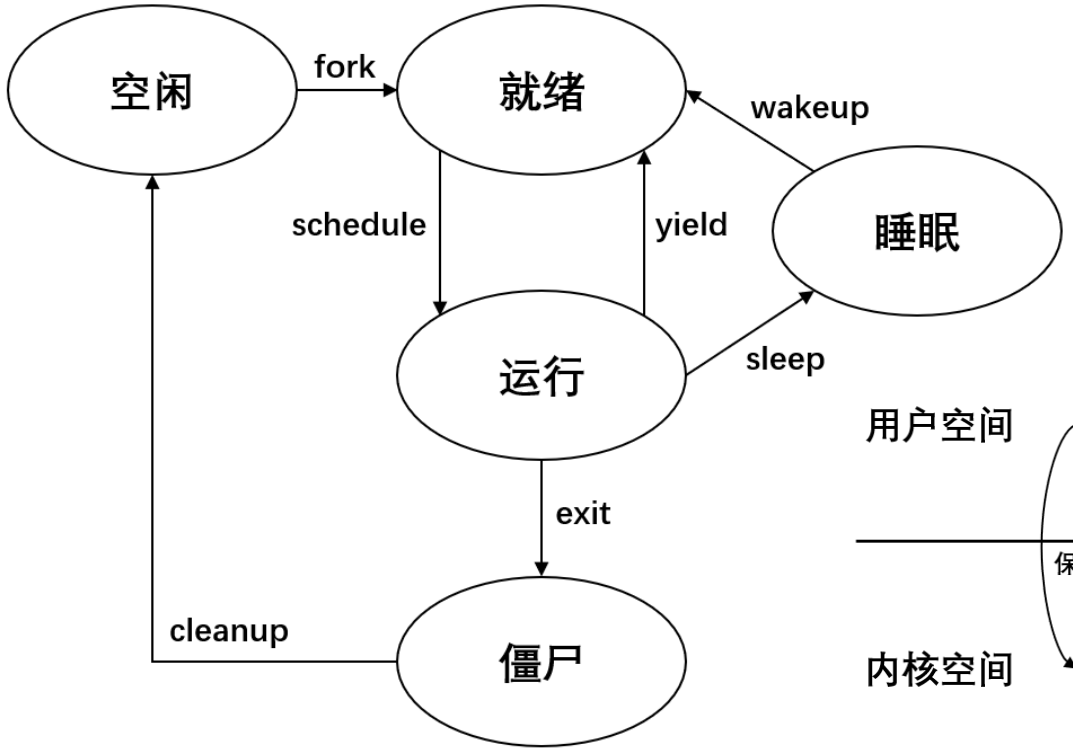
4.2 内存布局设计

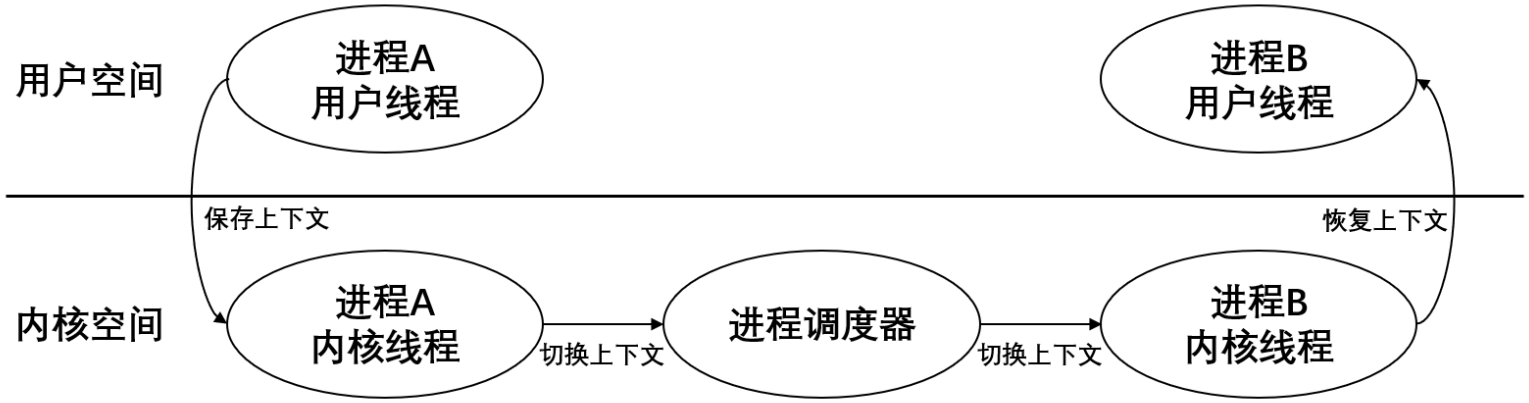


- 对硬件设备和系统内核采用直接映射，简化内核态内存访问的复杂性
- 进程内核栈以及上下文切换页存放在虚拟地址最高处，便于用固定偏移量寻址

4.3 进程管理设计

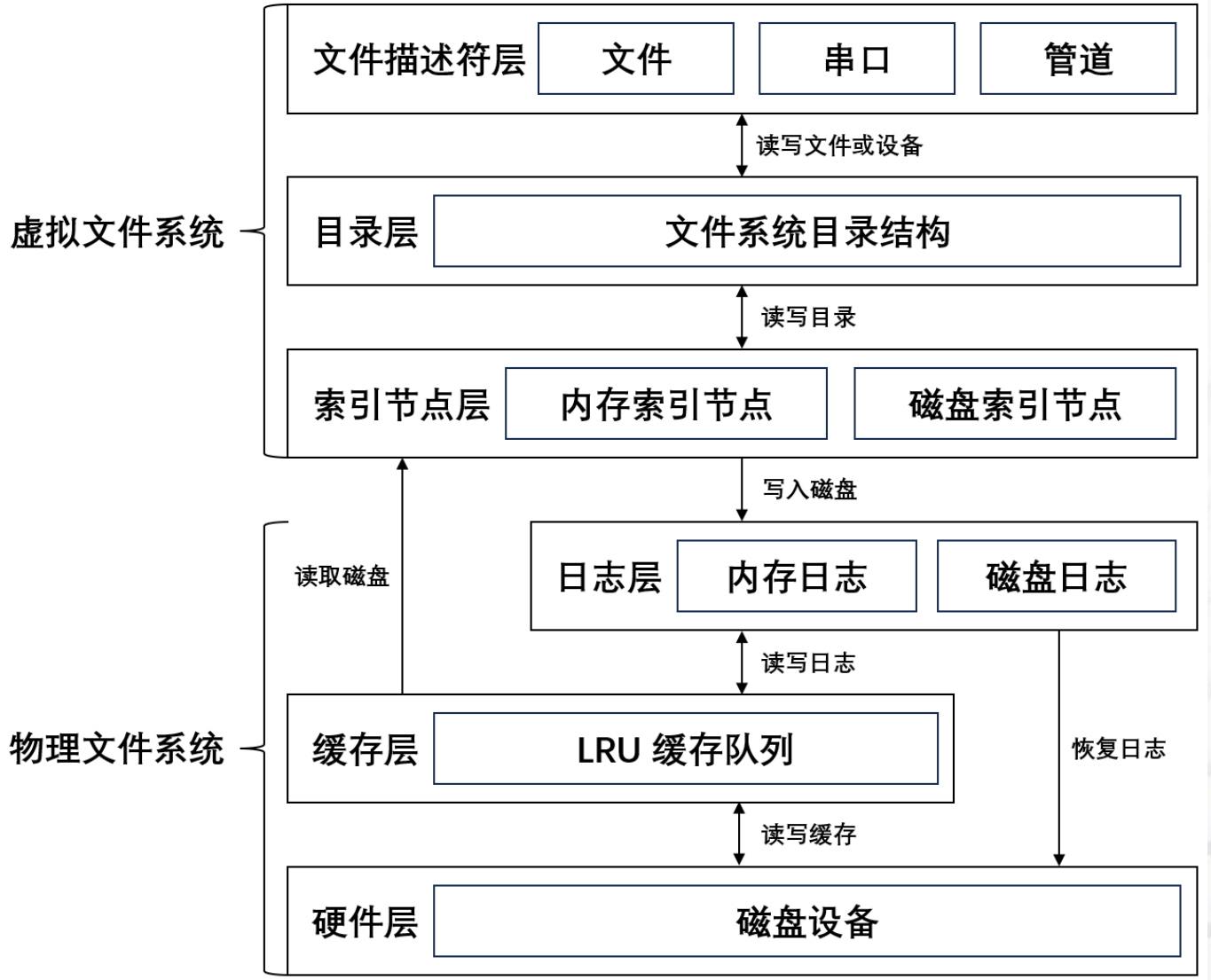
- 用户程序通过fork，wait等系统调用接口进行进程控制
- 进程在空闲，就绪，运行，睡眠，僵尸状态间切换
- 进程切换时首先进入其内核线程，再通过进程调度器切换至其他进程





4.4 文件系统设计

- 文件系统采用六层设计
- 硬件层，缓存层，日志层形成物理文件系统
- 索引节点层，目录层，文件描述符层形成虚拟文件系统



華東師範大學
EAST CHINA NORMAL UNIVERSITY

第五部分

内核文档注释

5.1 文档注释的意义

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 在本项目中，我们针对基于 Rust 语言实现的类 xv6 教学操作系统的内核代码，**围绕内存管理、进程管理、文件系统三个核心模块，系统性地补充了文档注释。**
- 注释内容涵盖**函数功能说明、调用流程解释、参数与返回值说明**，并根据具体语义补充了**可能的错误场景与安全性提示。**该工作对于操作系统课程的教学与学生的实验实践具有显著意义。
- **意义一：降低理解操作系统内核的难度**
- 内核代码复杂且耦合度高，初学者难以直接理解调用逻辑。
- 调用链中存在大量未知函数，容易造成阅读断层与认知障碍。
- 文档注释提供函数功能说明与上下文解释，有助于快速理解整体逻辑。
- Rust 引入的所有权与生命周期机制进一步增加理解难度。
- 注释中对 Rust 类型系统特性的解释，有效降低语言本身的学习门槛。

5.1 文档注释的意义

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- **意义二：提升内核调试的效率**
- 1. 内核态调试难度大，传统工具在处理底层问题时效果有限。
- 2. 实验中常见的 bug 多涉及汇编、中断或内存操作，定位困难。
- 3. 注释中提供异常场景与错误提示，有助于学生快速建立调试思路，提升调试效率，减少实验挫败感。
- **意义三：帮助学生参与内核开发**
- 1. 传统教学偏重接口使用，学生难以深入掌握内核实现细节，注释提升了内核代码的可读性与可维护性。
- 2. 为学生提供可理解、可修改的开发基础，鼓励学生从完成实验转向探索与优化内核，培养开发者思维。
- **意义四：增加对 Unsafe 代码的理解**
- 1. Rust 提供内存与并发安全保障，但内核开发中仍需使用 `unsafe`。
- 2. 注释中指出安全风险与语言层面的防御机制，帮助学生建立系统级编程中的安全意识。
- 3. 强化操作系统与嵌入式领域开发的基础能力。

5.2 文档注释的结构

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 本项目在实现内核文档时，参照上述标准库文档实现，根据不同函数的特性，包含了不同的文档段落
- 对于普通函数，文档内容包含
  - **函数功能说明**：使用简短的语言简述函数的功能，帮助学习者快速了解函数功能
  - **流程解释**：详细说明函数的执行流程及其具体实现，帮助无法读懂具体代码的学习者理解细节
  - **参数与返回值说明**：解释函数的输入与输出值
  - **可能的错误场景**：对Option，Result出现失败变体，或函数可能发生panic的情况进行说明，帮助学习者了解出错的可能情况
- 对于Unsafe函数，额外添加
  - **安全性说明**：详细说明该Unsafe函数需要人为保证的前提条件，否则可能造成系统进入不安全的状况
  - 有助于帮助学生快速了解使用Rust语言进行操作系统开发时需要注意的Unsafe代码使用问题

5.3 文档注释的实现

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 本项目采用手工编写与大模型优化的方式编写文档注释，具体实现流程如下：
- 1. 首先阅读需要编写文档的代码，并找出其在内核中被调用的位置
- 2. 编写其功能说明、参数、返回值、以及可能的错误场景
- 3. 将上述内容以及函数代码输入到大模型中，命令其检查并完善函数的流程解释
- 4. 若函数是Unsafe的，则额外添加其安全性说明
- 5. 人工检查所有内容，并添加到函数顶部
- 6. 执行cargo doc --no-deps --document-private-items,利用rustdoc工具产生HTML版本的文档

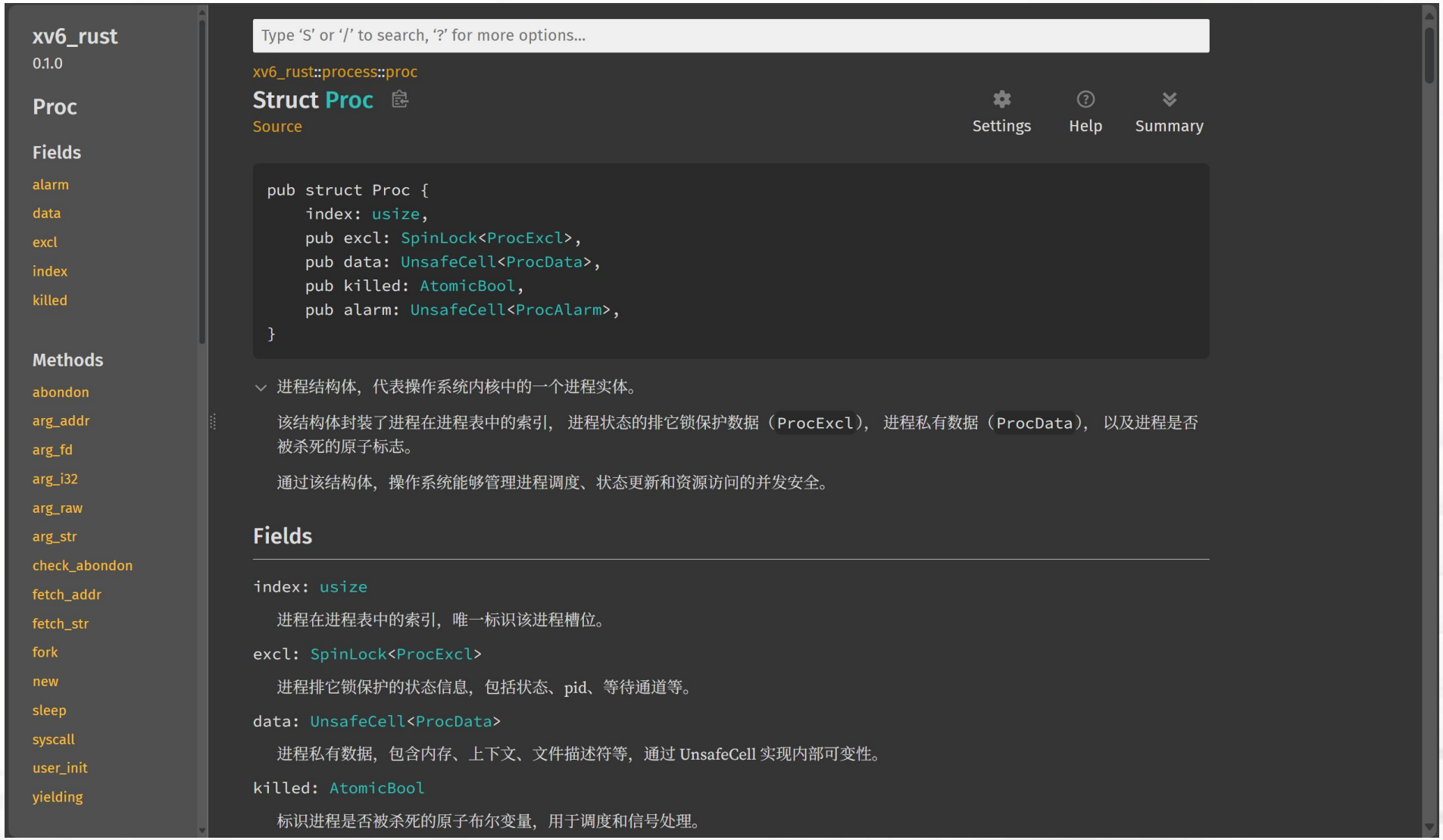
5.4 文档注释的覆盖范围

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 目前，本项目的文档注释已经基本覆盖内存管理模块、进程管理模块和文件系统模块，具体的文件与函数覆盖如下所示

```
.
├── fs
│   ├── bio.rs
│   ├── block.rs
│   ├── file
│   │   ├── mod.rs
│   │   └── pipe.rs
│   ├── inode.rs
│   └── log.rs
├── mm
│   ├── kalloc.rs
│   ├── kvm.rs
│   └── pagetable.rs
├── process
│   ├── cpu.rs
│   ├── mod.rs
│   ├── proc
│   │   ├── elf.rs
│   │   └── mod.rs
│   └── trapframe.rs
└── rmain.rs
```

5.5 文档注释的效果


xv6_rust
0.1.0
Proc
Fields
alarm
data
excl
index
killed
Methods
abondon
arg_addr
arg_fd
arg_i32
arg_raw
arg_str
check_abondon
fetch_addr
fetch_str
fork
new
sleep
syscall
user_init
yielding
Type 'S' or '/' to search, '?' for more options...
xv6_rust::process::proc
Struct Proc
Source
Settings
Help
Summary
pub struct Proc {
index: usize,
pub excl: SpinLock<ProcExcl>,
pub data: UnsafeCell<ProcData>,
pub killed: AtomicBool,
pub alarm: UnsafeCell<ProcAlarm>,
}
进程结构体，代表操作系统内核中的一个进程实体。
该结构体封装了进程在进程表中的索引， 进程状态的排它锁保护数据（ProcExcl）， 进程私有数据（ProcData）， 以及进程是否被杀死的原子标志。
通过该结构体，操作系统能够管理进程调度、状态更新和资源访问的并发安全。
Fields
index: usize
进程在进程表中的索引，唯一标识该进程槽位。
excl: SpinLock<ProcExcl>
进程排它锁保护的状态信息，包括状态、pid、等待通道等。
data: UnsafeCell<ProcData>
进程私有数据，包含内存、上下文、文件描述符等，通过 UnsafeCell 实现内部可变性。
killed: AtomicBool
标识进程是否被杀死的原子布尔变量，用于调度和信号处理。

5.5 文档注释的效果

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 目前，本项目的文档注释已经基本覆盖内存管理模块、进程管理模块和文件系统模块，使用tokie统计注释行数，目前文档注释总量为3435行。

Language	Files	Lines	Code	Comments	Blanks
GNU Style Assembly	6	390	341	6	43
C	23	5123	4325	251	547
C Header	12	644	459	97	88
Makefile	1	130	97	7	26
Markdown	3	198	0	177	21
Perl	1	40	33	2	5
Python	2	672	514	47	111
TOML	2	28	22	1	5
Rust	49	7018	5648	275	1095
|- Markdown	45	4223	0	3435	788
(Total)		11241	5648	3710	1883
Total	99	14243	11439	863	1941

華東師範大學
EAST CHINA NORMAL UNIVERSITY

第六部分

Rust语言用户程序与线程库

6.1 Rust语言用户程序

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 在本项目中，操作系统的整体架构与 MIT xv6 保持一致，采用相同的系统调用接口与用户态启动方式。
- 配合通过用户程序以系统调用的方式与内核交互，可以有效地验证系统各项功能的正确性。
- 为进一步统一语言生态、提升教学一致性，已经使用 Rust 重写部分测试用例，并已实现初步成功运行。
- 未来计划将全部用户程序迁移为 Rust 实现，构建一个全栈基于 Rust 的教学操作系统与测试平台。

6.2 用户程序库基础结构

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 由于在操作系统基础上编写用户程序，无法直接使用Rust的标准库。
- 本用户程序库（在no_std环境下）结合操作系统**系统调用接口**、alloc官方库，core官方库和部分**第三方库**进行进行实现。

6.2 用户程序库基础结构

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 用户程序入口
  - 通过link.ld进行用户代码链接。 通过ENTRY(_start)来指定用户程序入口函数 。
  - OUTPUT_ARCH(riscv)指定目标架构为RISC-V。
  - ENTRY(_start)指定程序的入口点为_start符号（用户程序启动代码）。
  - BASE_ADDRESS = 0x10000表示程序加载的基地址为0x10000。
  - .text 段包含所有函数代码。
  - .rodata 段存储字符串常量。
  - .data 和 .bss 分别存储初始化和未初始化的全局变量。

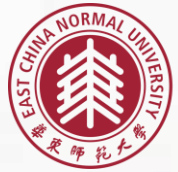
```
OUTPUT_ARCH(riscv)
ENTRY(_start)

BASE_ADDRESS = 0x10000;

SECTIONS
{
    . = BASE_ADDRESS;
    .text : {
        *(.text.entry)
        *(.text .text.*)
    }
    . = ALIGN(4K);
    .rodata : {
        *(.rodata .rodata.*)
        *(.srodata .srodata.*)
    }
    . = ALIGN(4K);
    .data : {
        *(.data .data.*)
        *(.sdata .sdata.*)
    }
    .bss : {
        *(.bss .bss.*)
        *(.sbss .sbss.*)
    }
    /DISCARD/ : {
        *(.eh_frame)
        *(.debug*)
    }
}

```

6.2 用户程序库基础结构

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 用户栈中的堆内存分配
  - 基于**buddy_system_allocator**库实现的用户栈中内存分配器初始化配置
- _start
  - 1.初始化用户栈中的堆内存
  - 2.从内核初始化的进程用户栈中获取参数**argc: usize, argv: usize**（内核参数设置符合函数调用规范） seios操作系统内核已经将参数压入用户栈中，以下start_中将符合C语言标准的参数转换为符合Rust语言标准的参数（处理字符串）
  - 3.执行用户编写的主函数**fn main(argc:usize, argv:&[&str]) -> i32**
  - 4.退出，执行exit系统调用，保存返回值并退出进程。

6.2 用户程序库基础结构



華東師範大學
EAST CHINA NORMAL UNIVERSITY

```rust
#[no_mangle]
#[link_section = ".text.entry"]
pub extern "C" fn _start(argc: usize, argv: usize) -> ! {
    unsafe {
        HEAP.lock() MutexGuard<'_, Heap>
            .init(start: HEAP_SPACE.as_ptr() as usize, USER_HEAP_SIZE);
    }
    let mut v: Vec<&'static str> = Vec::new();
    for i: usize in 0..argc{
        let str_start: usize = unsafe {
            ((argv + i * core::mem::size_of::<usize>()) as * const usize).read_volatile()
        };
        let len: usize = (0usize..) RangeFrom<usize>
            .find(|i: &usize| unsafe{((str_start + *i) as *const u8).read_volatile() == 0}) Option<usize>
            .unwrap();
        v.push(
            core::str::from_utf8(
                unsafe {
                    core::slice::from_raw_parts(data: str_start as *const u8, len)
                }
            ) Result<&str, Utf8Error>
            .unwrap()
        );
    }
    exit(exit_code: main(argc,_argv: v.as_slice()));
}
```

6.3 用户程序库基础结构

華東師範大學
EAST CHINA NORMAL UNIVERSITY

为用户程序实现的 panic 处理函数，使用rust提供的属性注释`#[panic_handler]` 来定义当程序发生不可恢复错误（panic）时的处理行为。该标记该函数为全局panic处理器，替代标准库的默认实现。当程序触发panic!时，此函数会被调用。

该错误处理函数先输出错误发生的信息,再调用kill(getpid())终止当前进程。

```rust
#[panic_handler]
fn panic_handler(panic_info: &core::panic::PanicInfo) -> ! {
    // 打印 panic 位置（文件 + 行号）
    if let Some(location) = panic_info.location() {
        println!(
            "Panicked at {}:{}:{}",
            location.file(),
            location.line(),
            location.column()
        );
    }
    // 打印 panic 消息（如果有）
    println!("Error: {}", panic_info.message());
    kill(getpid());
    unreachable!()
}
```

6.4 系统调用接口

根据Riscv结构，使用ecall进行内核陷入

```rust
#![no_std]
/// the syscall on RISCV chips which support 6 parameters
pub fn syscall(id: usize, args: [usize; 6]) -> isize {
    let mut ret: isize;
    unsafe {
        core::arch::asm!(
            "ecall",
            inlateout("x10") args[0] => ret,
            in("x11") args[1],
            in("x12") args[2],
            in("x13") args[3],
            in("x14") args[4],
            in("x15") args[5],
            in("x17") id
        );
    }
    ret
}
```

6.4 系统调用接口

華東師範大學
EAST CHINA NORMAL UNIVERSITY

SYSCALL_FORK(syscall_id = 1) 功能：创建当前进程的一个子进程，复制父进程的内存、TrapFrame、打开文件、当前工作目录等信息，并将子进程状态设置为可运行
SYSCALL_EXIT(syscall_id = 2) 功能：终止当前进程，释放其资源，并向父进程传递退出状态
SYSCALL_WAIT(syscall_id = 3) 功能：等待任意子进程终止，并获取其退出状态
SYSCALL_PIPE(syscall_id = 4) 功能：创建一个管道，用于进程间通信（IPC）
SYSCALL_READ(syscall_id = 5) 功能：从文件描述符读取数据。
SYSCALL_KILL(syscall_id = 6) (未实现sig：信号编号) 功能：向指定进程发送信号
SYSCALL_EXEC(syscall_id = 7) 功能：加载并执行新程序，替换当前进程的地址空间
SYSCALL_FSTAT(syscall_id = 8) 功能：获取文件描述符对应的文件状态信息。
SYSCALL_CHDIR(syscall_id = 9) 功能：更改当前进程的工作目录。
SYSCALL_DUP(syscall_id = 10) 功能：复制文件描述符

6.4 系统调用接口

華東師範大學
EAST CHINA NORMAL UNIVERSITY

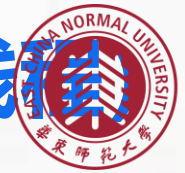
SYSCALL_GETPID(syscall_id = 11) 功能： 获取当前进程的 PID
SYSCALL_SBRK(syscall_id = 12) 功能： 调整进程的堆内存大小
SYSCALL_SLEEP(syscall_id = 13) 功能： 使进程休眠指定时间
SYSCALL_UPTIME(usize = 14) 功能： 获取系统启动后的时钟周期数
SYSCALL_OPEN(usize = 15) 功能： 打开或创建文件
SYSCALL_WRITE(usize = 16) 功能： 向文件描述符写入数据。
SYSCALL_MKNOD(usize = 17) 功能： 创建设备文件或特殊文件
SYSCALL_UNLINK(usize = 18) 功能： 删除文件链接
SYSCALL_LINK:(usize = 19) 功能： 创建硬链接
SYSCALL_MKDIR(usize = 20) 功能： 创建目录
SYSCALL_CLOSE:(usize = 21) 功能： 关闭文件描述符
SYSCALL_GETMTIME(usize = 22) 功能： 获取riscv处理器的计时器寄存器

6.5 用户线程库（不依赖内核线程）

華東師範大學
EAST CHINA NORMAL UNIVERSITY

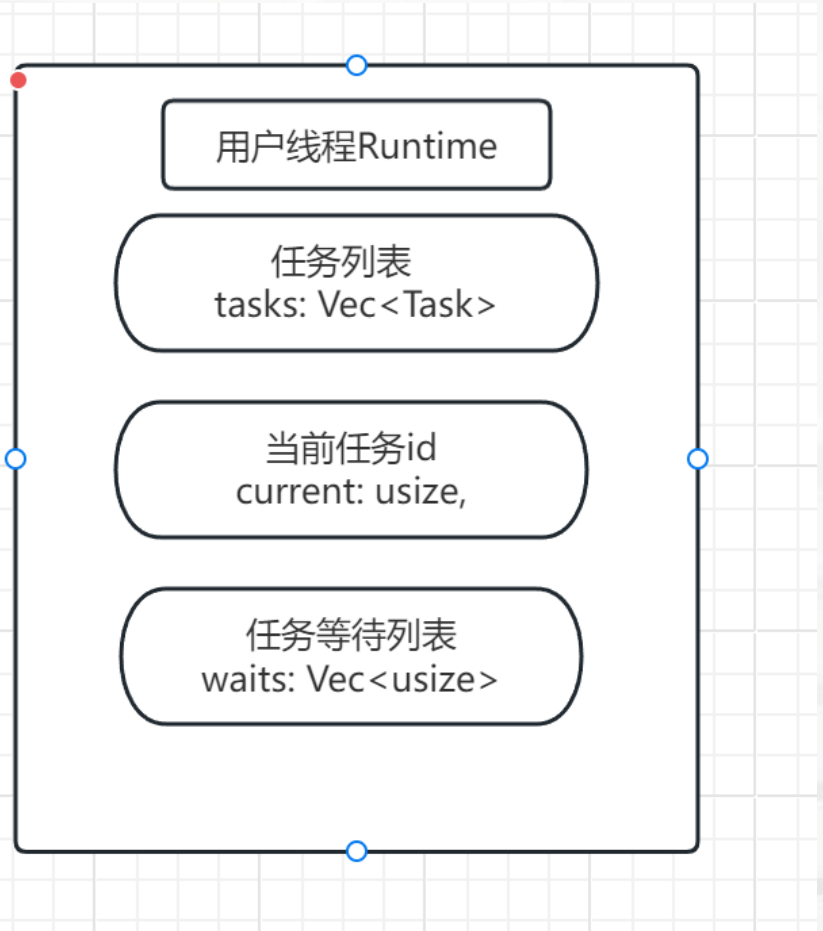
- 用户空间线程库（User-Level Thread Library）是一种在用户态实现线程管理的机制。
- 与操作系统内核管理的线程（内核线程）不同，用户线程的创建、调度、同步等操作完全由库在用户空间处理，无需频繁陷入内核，从而减少上下文切换的开销。
- **这可以算是一个简易的内核，可用于教学，便于学生后续理解内核切换任务的过程。**

6.5 用户线程库实现（不依赖内核线程）



華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 线程调度器结构
  - 管理所有用户线程
  - 记录当前正在执行的线程
  - 维护任务间的等待依赖（等待特定或任意线程完成）



6.5 用户线程库实现（不依赖内核线程）

用户线程包含：

1.线程唯一编号

2.栈指针

3.线程上下文

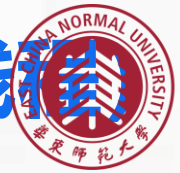
4.状态（空闲、运行、可调度、等待）

5.Runtime指针

```rust
pub struct Task {
    pub id: usize,
    pub stack: Vec<u8>,
    pub ctx: TaskContext,
    pub state: TaskState,
    pub r_ptr: u64
}
```

```rust
#[derive(Debug, Default, Clone, Copy)]
#[repr(C)] // not strictly needed but Rust ABI is not guaranteed to be stable
4 implementations
pub struct TaskContext {
    pub x1: u64,  //ra: return address
    pub x2: u64,  //sp
    pub x8: u64,  //s0,fp
    pub x9: u64,  //s1
    pub x18: u64, //x18-27: s2-11
    pub x19: u64,
    pub x20: u64,
    pub x21: u64,
    pub x22: u64,
    pub x23: u64,
    pub x24: u64,
    pub x25: u64,
    pub x26: u64,
    pub x27: u64,
    pub nx1: u64, //new return address
    pub r_ptr: u64,// self ptr
    pub params: u64
}
```

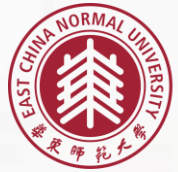
6.5 用户线程库实现（不依赖内核线程）



華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 线程切换机制
  - 线程通过主动让出执行权参与调度
  - 使用上下文切换技术保存和恢复寄存器状态
  - 避免无效的切换（如切换到自身或空闲线程）
  - 调度器寻找状态为“可调度”的下一个线程执行

6.5 用户线程库（不依赖内核线程）



華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 这里在用户创建的线程中的使用yield_task主动让出处理器资源和使用guard退出任务。

```rust
pub fn yield_task(r_ptr: *const Runtime) {
    unsafe {
        let rt_ptr: *mut Runtime = r_ptr as *mut Runtime;
        (*rt_ptr).t_yield();
    };
}
```

```rust
#[inline(never)]
fn t_yield(&mut self) -> bool {
    let mut pos: usize = (self.current + 1) % MAX_TASKS;
    let mut temp: usize = 0usize;
    while self.tasks[pos].state == TaskState::Sleep || self.tasks[pos].state == TaskState::Available {
        pos += 1;
        if pos == self.tasks.len() {
            pos = 1;
            if temp == 1 {
                pos = 0;
            }
            temp = 1;
        }
        if pos == 0 && pos == self.current {
            if !self.waits.iter().any(|&x: usize| x != 0){
                return false;
            }
        }
    }
    if self.tasks[self.current].state != TaskState::Sleep {
        if self.tasks[self.current].state != TaskState::Available {
            self.tasks[self.current].state = TaskState::Ready;
        }
    }

    self.tasks[pos].state = TaskState::Running;
    let old_pos: usize = self.current;
    self.current = pos;
    if old_pos == pos {
        return  self.tasks.len() > 0
    }
    unsafe {
        switch(old: &mut self.tasks[old_pos].ctx, new: &self.tasks[pos].ctx);
    }

    // NOTE: this might look strange and it is. Normally we would just mark this as `unreachable!()` but our compiler
    // is too smart for it's own good so it optimized our code away on release builds. Curiously this happens on windows
    // and not on linux. This is a common problem in tests so Rust has a `black_box` function in the `test` crate that
    // will "pretend" to use a value we give it to prevent the compiler from eliminating code. I'll just do this instead,
    // this code will never be run anyways and if it did it would always be `true`.
    self.tasks.len() > 0
} fn t_yield
```

6.5 用户线程库实现

華東師範大學
EAST CHINA NORMAL UNIVERSITY

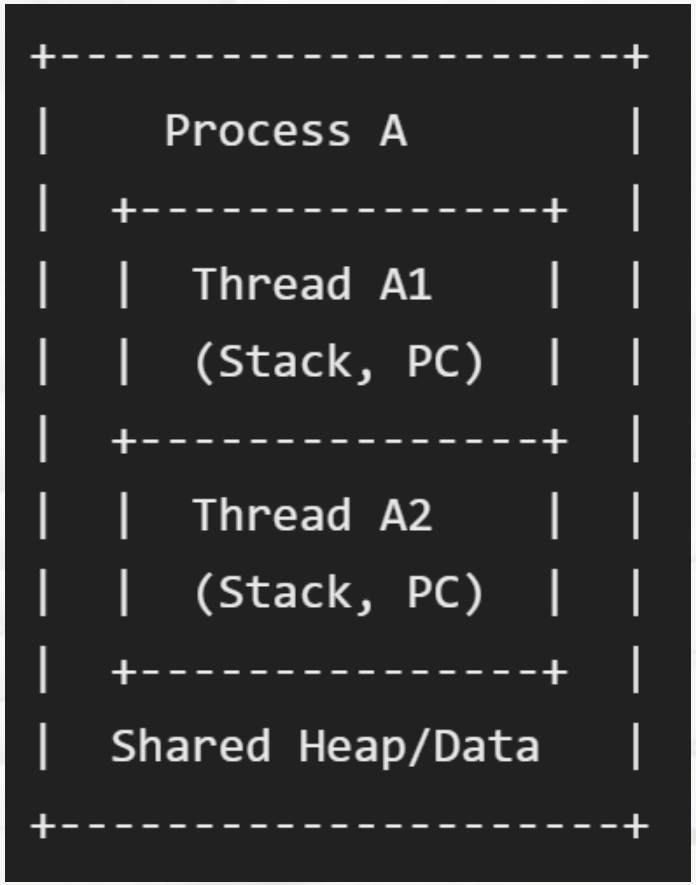
- 线程创建与启动
  - 从空闲任务池中选择任务槽位
  - 初始化其上下文、栈指针与传入参数
  - 设置线程初始状态为可调度，等待调度器启动执行
- 线程同步机制
  - 线程可以选择挂起，等待特定线程或任意线程完成
  - 其他线程可通过发送信号唤醒被等待的线程
  - 实现了线程间的简单同步依赖关系
- 线程结束与资源回收
  - 执行完毕后进入“可重用”状态
  - 自动唤醒所有等待当前线程完成的其他线程
  - 返回控制权给调度器，继续执行其他任务

6.6 内核线程

共享资源：线程共享进程的地址空间、文件描述符等。

独立执行流：每个线程有自己的栈、寄存器和程序计数器（PC）。

切换开销小：无需切换虚拟内存，仅需切换栈和寄存器。



6.6 内核线程

以下为内核进程和线程具体设计的结构

```rust
pub struct ProcessControlBlock {
    pub pid: usize,
    pub is_zombie: bool,
    pub page_table: PageTable,
    pub parent: Option<Weak<ProcessControlBlock>>,
    pub children: Vec<Arc<ProcessControlBlock>>,
    pub exit_code: i32,
    pub fd_table: [Option<Arc<File>>; NFILE],
    pub tasks: Vec<Option<Arc<TaskControlBlock>>>,
}
```

```rust
pub struct TaskControlBlock {
    pub tid: usize,
    pub process: Weak<ProcessControlBlock>,
    pub kstack: KernelStack,
    // mutable
    pub ustack_base: usize,
    pub process: Weak<ProcessControlBlock>,
    pub trap_cx_ptr: usize,
    pub task_cx: TaskContext,
    pub task_status: TaskStatus,
    pub exit_code: Option<i32>,
}
```

6.6 内核线程（待实现）

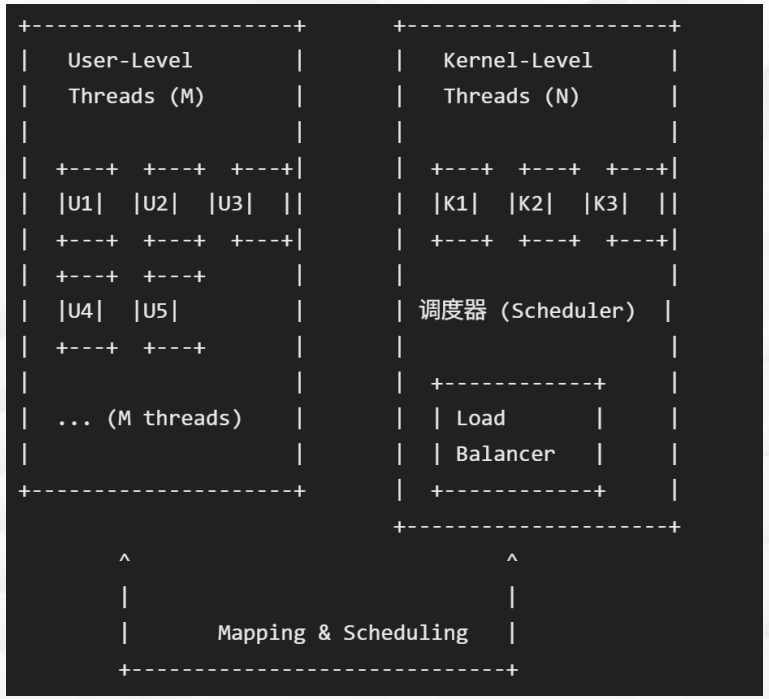
- *const* SYSCALL_THREAD_CREATE: usize; //线程创建
- *const* SYSCALL_GETTID: usize; //获取线程号
- *const* SYSCALL_WAITTID: usize; //等待其他线程

6.7 用户线程库实现（结合内核线程实现）

后续实现了内核线程之后，将结合内核线程调用和用户线程管理（Runtime）用实现减少内核态切换开销，支持高并发。

M:N模型： 用户线程（轻量级）由线程库管理，内核线程（重量级）由OS调度。

挑战： 需要手动处理阻塞、调度和同步问题。



華東師範大學
EAST CHINA NORMAL UNIVERSITY

第七部分

实验指导与参考实现

7.1 测试用例设计

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 在本项目中，操作系统的整体架构与 MIT xv6 保持一致，采用相同的系统调用接口与用户态启动方式。
- 我们成功地将 xv6 的用户程序 usertests 以及各实验对应的测试程序（如 alarmtest、bttest 等）移植到了该 Rust 实现的类 xv6 教学操作系统上。
- 通过用户程序以系统调用的方式与内核交互，可以有效地验证系统各项功能的正确性。
- 这些测试程序仍主要以 C 语言实现。为进一步统一语言生态、提升教学一致性，我们已开始尝试以 Rust 重写部分测试用例，并已实现初步成功运行。
- 未来计划将全部用户程序迁移为 Rust 实现，构建一个全栈基于 Rust 的教学操作系统与测试平台。

7.1 测试用例设计

- **综合性回归测试：usertests**
  - 含约 60 个子测试用例，覆盖进程管理、内存管理、文件系统、系统调用边界条件等基础功能模块。
  - 可用于持续验证核心功能的完整性，确保学生在实现新功能时未破坏已有行为逻辑。
  - 作为教学系统的“回归测试基准”，可嵌入自动化测试流程。
- **实验对应专项测试程序**
  - 每个实验新增机制配套一个专门测试程序。
  - 专项测试程序能更精确地验证功能实现是否符合实验设计目标，补充通用测试的不足。
  - 构成“覆盖广 + 针对强”的测试闭环设计，提升教学准确性与系统鲁棒性。

7.2 测试用例函数

- 下面简单介绍移植自xv6的60个测试用例的一部分：

```
void copyin(char *s);          // 测试将无效指针传递给从用户空间读取数据的系统调用
void copyout(char *s);         // 测试将无效指针传递给向用户空间写入数据的系统调用
void copyinstr1(char *s);      // 测试将非法字符串指针传递给系统调用
void copyinstr2(char *s);      // 测试当参数字符串长度恰好等于内核缓冲区大小时的处理
void copyinstr3(char *s);      // 测试字符串参数跨越用户空间最后一页边界时的行为
void rwsbrk();                 // 测试进程归还内存后对已释放地址执行读写操作的处理
void truncate1(char *s);       // 测试使用 O_TRUNC 截断文件后已打开文件描述符读取行为是否正确
void truncate2(char *s);       // 测试文件被截断后对原文件描述符执行写操作是否正确失败
void truncate3(char *s);       // 测试多进程同时对同一文件进行截断和写入操作的情况
void iputtest(char *s);        // 测试进程切换到新目录后删除该目录时内核对引用计数的处理
void exitiputtest(char *s);    // 测试进程退出时能否正确释放其当前工作目录的引用（iput）
void openiputtest(char *s);    // 测试尝试写打开目录出错时内核是否正确回收 inode
void opentest(char *s);        // 测试打开存在的文件成功和打开不存在的文件失败的情况
```

7.3 测试脚本设计

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 自动化测试脚本通过执行 make grade 命令触发测试流程，评测的整体过程包括以下几个关键步骤：
- **1. 清理环境并重置文件系统镜像：**首先清除之前编译生成的中间文件，并将文件系统镜像恢复为干净状态，以确保测试在一致的初始环境下运行。
- **2. 重新编译内核与用户程序：**执行完整的 make 构建流程，编译最新的内核和用户态测试程序，并生成包含测试程序的初始文件系统镜像。
- **3. 通过QEMU启动系统并执行测试程序：**启动带有 GDB 调试端口的 QEMU 模拟器，进入 xv6 shell 后，根据测试用例预设内容自动执行测试命令（包括实验相关测试程序与通用的用户程序测试）。
- **4. 采集QEMU输出并进行内容匹配：**通过与 GDB 建立连接，实时捕获 xv6 运行期间的标准输出，并与预期的输出模式进行匹配校验。
- **5. 统计得分并输出测试结果：**根据每个测试点是否通过来累计得分，最终输出整体得分情况与详细的测试通过/失败信息，供用户评估实验完成度。

7.4 测试点例子

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 一个修改过的测试点示例如下所示，该测试点用于backtrace功能的测试，测试流程是首先在内核中执行bttest，获取内核输出的几个栈地址，然后在内核的可执行文件中使用addr2line工具查找这几个栈地址是否是目标文件中的正确地址。

```
@test(10, "backtrace test")
def test_backtracetest():
    r.run_qemu(shell_script([
        'bttest'
    ]))
    a2l = addr2line()
    matches = re.findall(BACKTRACE_RE, r.qemu.output, re.MULTILINE)
    assert_equal(len(matches), 3)
    files = ['syscall.rs', 'mod.rs', 'trap.rs']
    for f, m in zip(files, matches):
        result = subprocess.run([a2l, '-e', 'target/riscv64gc-unknown-none-elf/debug/xv6-rust', m],
stdout=subprocess.PIPE)
        if not f in result.stdout.decode("utf-8"):
            raise AssertionError('Trace is incorrect; no %s' % f)
```

7.5 测试脚本运行效果

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 运行一个自动测试脚本，若所有测试均通过的情况如下所示，可以看到各个测试点的通过情况和分数汇总：

```
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 4.22s
make[1]: Leaving directory '/workspaces/os-competition/xv6-rust'
== Test backtrace test == backtrace test: OK (2.6s)
== Test running alarmtest == (2.5s)
== Test   alarmtest: test0 ==
  alarmtest: test0: OK
== Test   alarmtest: test1 ==
  alarmtest: test1: OK
== Test   alarmtest: test2 ==
  alarmtest: test2: OK
== Test usertests == usertests: OK (224.2s)
Score: 79/79
```

7.5 测试脚本运行效果

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 若存在学生实验不符合要求的情况，测试点不通过的输出如下所示：

```
== Test backtrace test == backtrace test: FAIL (2.9s)
    got:
      0
    expected:
      3
    QEMU output saved to xv6.out.backtracetest
== Test running alarmtest == (26.3s)
== Test   alarmtest: test0 ==
  alarmtest: test0: FAIL
    ...
         test1 failed: too few calls to the handler
         test2 start
         ..........................................................................
..........................................................................
..........................................................................
..........................................................
         test2 failed: alarm not called
         $ qemu-system-riscv64: terminating on signal 15 from pid 81378 (make)
    MISSING '^test0 passed$'
```

7.6 已完成的实验设计

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 目前已完成6个实验的移植、修改与实现，分别是：
- **1. 系统调用跟踪**
- 为内核增加一个系统调用追踪功能，通过新增的 trace(mask) 系统调用按位指定需要追踪的系统调用类型。内核在被追踪的系统调用即将返回时，打印出该进程的 PID、系统调用名及其返回值，从而辅助调试后续实验。
- **2. 加速系统调用**
- 在每个进程创建时，将一个只读页映射到用户空间的 USYSCALL 虚拟地址，并在该页起始位置写入包含当前进程 PID 的 struct usyscall 结构体。用户空间通过 ugetpid() 访问该映射页即可获取进程 PID，实现无需陷入内核的高效用户态系统信息查询。
- **3. 打印页表**
- 实现一个 vmprint(pagetable_t) 函数，用于按指定格式打印页表内容，并在 exec.c 中对第一个进程（PID 为 1）在返回前调用该函数输出其页表。该功能用于加深对多级页表结构的理解，并通过页表打印测试验证实现正确性。

7.6 已完成的实验设计

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- **4. 检测某一页是否被访问过**
- 实现 pgaccess() 系统调用，用于报告一组用户页是否被访问过。它接收起始虚拟地址、页数以及用于写入访问结果的用户空间位掩码缓冲区，返回值按位表示每页的访问状态，有助于实现如页面置换等高级功能。
- **5. 回溯调用栈**
- 实现一个 backtrace() 函数，用于输出当前内核栈的调用地址，并在 sys_sleep 中调用该函数进行回溯测试。通过 addr2line 工具可将输出地址解析为源代码位置，从而实现内核调试中常用的调用路径追踪功能。
- **6. 周期性任务调用**
- 该实验要求为内核增加定时报警机制，使进程在占用一定 CPU 时间后自动触发用户定义的处理函数，实现类似用户级中断的效果。该机制可用于周期性任务或资源限制场景，正确实现需通过 alarmtest 和 usertests 验证。

7.6 已完成的实验设计

華東師範大學
EAST CHINA NORMAL UNIVERSITY

- 具体实验指导与参考实现详见初赛文档

華東師範大學
EAST CHINA NORMAL UNIVERSITY

第八部分

总结

8 总结

- 本项目构建了一个基于 Rust 的教学操作系统实验平台，参考 xv6 的系统结构与接口，结合 xv6-riscv-rust实现的内核，设计了完整的文档注释体系、用户程序与测试用例、自动化测试脚本、实验指导手册与参考实现。
- 内核采用 Rust 编写并配套标准化注释文档，支持通过 cargo doc 自动生成阅读材料；用户程序与测试机制兼容 xv6，逐步向 Rust 迁移；测试脚本支持自动编译、运行与评分，保障评测的准确性和教学效率；实验设计结合 MIT 6.828 内容，提供分层提示与参考代码，适应不同教学需求，形成一套可持续发展的操作系统教学支撑体系。

華東師範大學
EAST CHINA NORMAL UNIVERSITY

谢谢！
Thanks for listening!

汇报队伍：SEIOS 队员：郑子攸，叶晨皓，周恒如 指导老师：郭建