Rust xv6 on RV的设计与实现

陈恒杰
指导教师：周广禄
哈尔滨工业大学(威海)

自我介绍

- 计算机科学与技术-大四本科生

- 大三上学期学习操作系统课程，下学期开始做这个项目

- Q：契机?
  A：1. 大三上学习了xv6
  2. 想学习Rust语言（一门系统级编程语言）

前言

- xv6是？Rust在写OS上与C有什么不同？有哪些好处？
- 我实现这个项目的过程，心得和体会
- 现在项目的完成情况以及作为赛题

xv6(xv6-riscv)

- MIT-PDOS实验室用C语言开发的一款教学操作系统，基于Unix V6 ， LoC：~1w
- 多核，64个进程，RR调度，简单文件系统（7层，支持日志）20+系统调用，2个同步原语，中断管理，设备驱动，x86/riscv
- 用于6.828/6.S081课程，有配套的实验和文档

Rust v.s. C on OS

- Rust相对于C在语言层面支持更多特性：OOP，FP，module system等
- Rust中的智能指针 v.s. C中的指针
- Rust的所有权系统和生命周期使得程序员需要考虑：
  - 单线程或多线程
  - 不可变或可变借用
  - 与内部可变性相关的智能指针
  - raw pointer（unsafe）

  并显式写在代码中，并被Rust编译器检查

Rust实现xv6的好处

- 不必过多考虑OS的整体设计
- 不必过多考虑一些功能的取舍
- 更多关注在细节/实现上，适合初学者

我的实现步骤

- 学习OS课程，学习Rust语言

- 从零开始

1. OS启动：汇编，x86/riscv
2. Rust主函数：硬件，内存，文件系统等初始化
3. 调度器：CPU，进程管理
4. 实现系统调用 --> 完善123步

- 时间投入：3个月左右，遇到的困难

生命周期

- Q：全局对象比如Cpu, Process, Disk, Fs怎么表达？
  A：static mut, 他们的生命周期是'static

- 其他对象的生命周期来源于全局对象

- 但是在OS/xv6中很多情况中又要使用裸指针，设计上要权衡，比如进入内核态后myproc() -> *mut Proc还是&mut Proc

智能指针的一个例子：同步原语

- Xv6中：spinlock

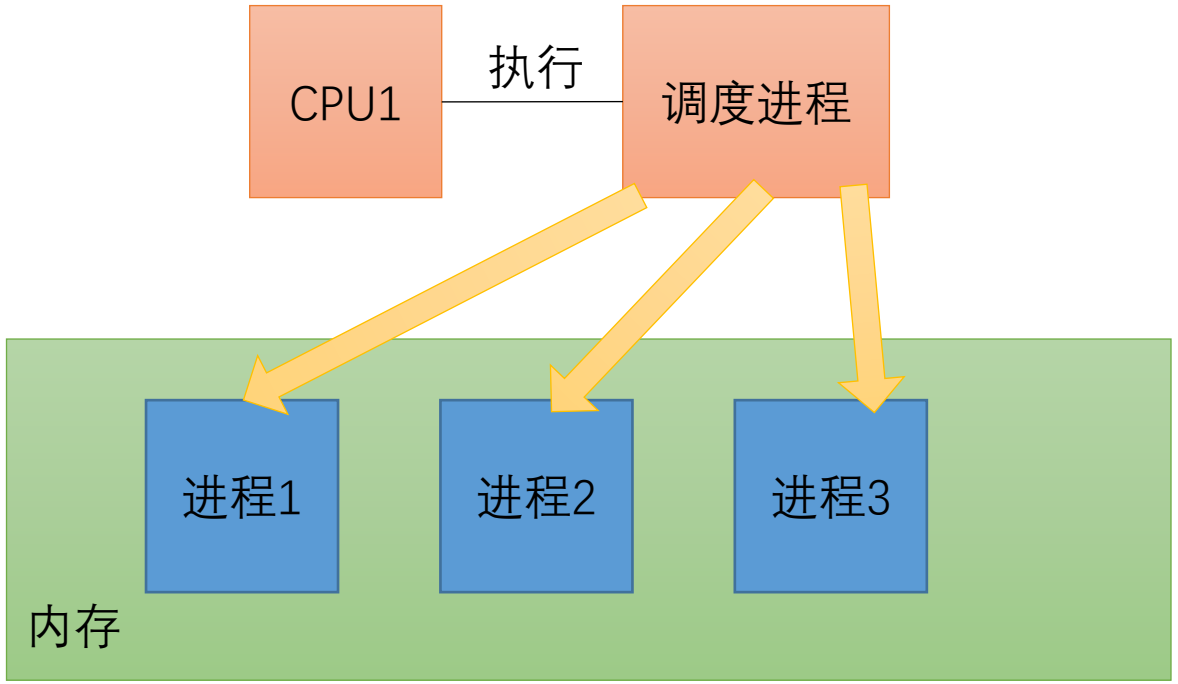
- API：acquire, release, holding
  - acquire：关闭中断，不断循环请求加锁
  - release：解锁，恢复上次中断打开状态

- 在C语言中往往是其他struct的内部成员

Rust中怎么实现?

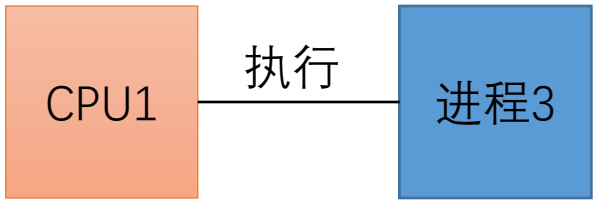
- Rust标准库中有诸如：Mutex，Condvar，RwLock等同步原语
- 在OS中需要自己实现
- 希望采用RAII的设计，好处：
  - 易用，锁和资源耦合
  - 安全，获取和释放是匹配的（Rust编译器检查）

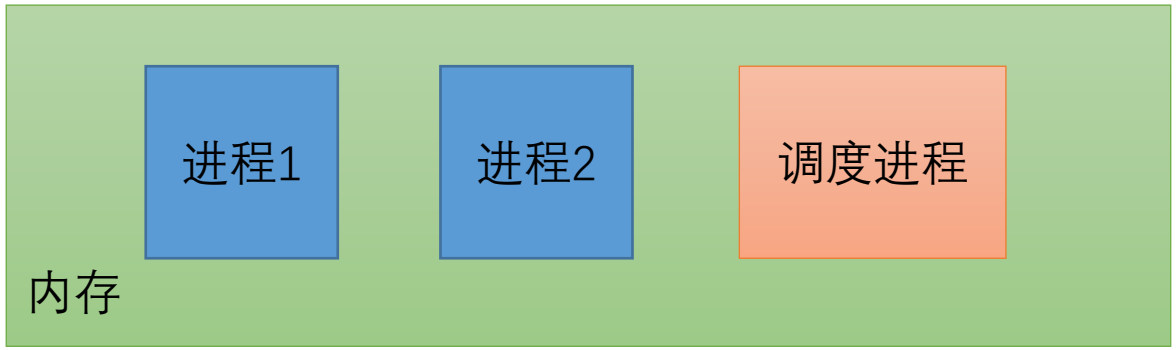
RAII带来的问题（I）



- 每个CPU各自一个调度进程
- 使用RR调度算法，p.lock保证p.state互斥访问
- API：
  - Acquire(p.lock)
  - Swtch: 切换上下文

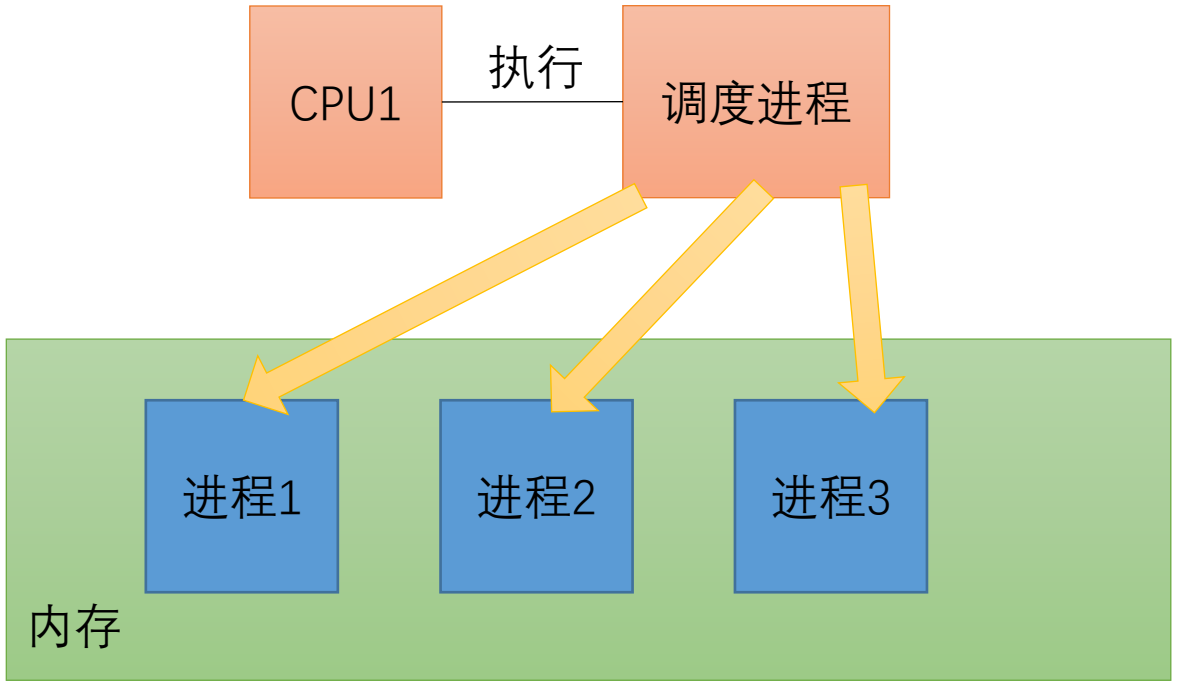
RAII带来的问题（II）





- 进程3空闲被CPU1调度
- 内核态中解锁p3.lock，然后继续执行
- API：
  - Release(p.lock)

RAII带来的问题（III）



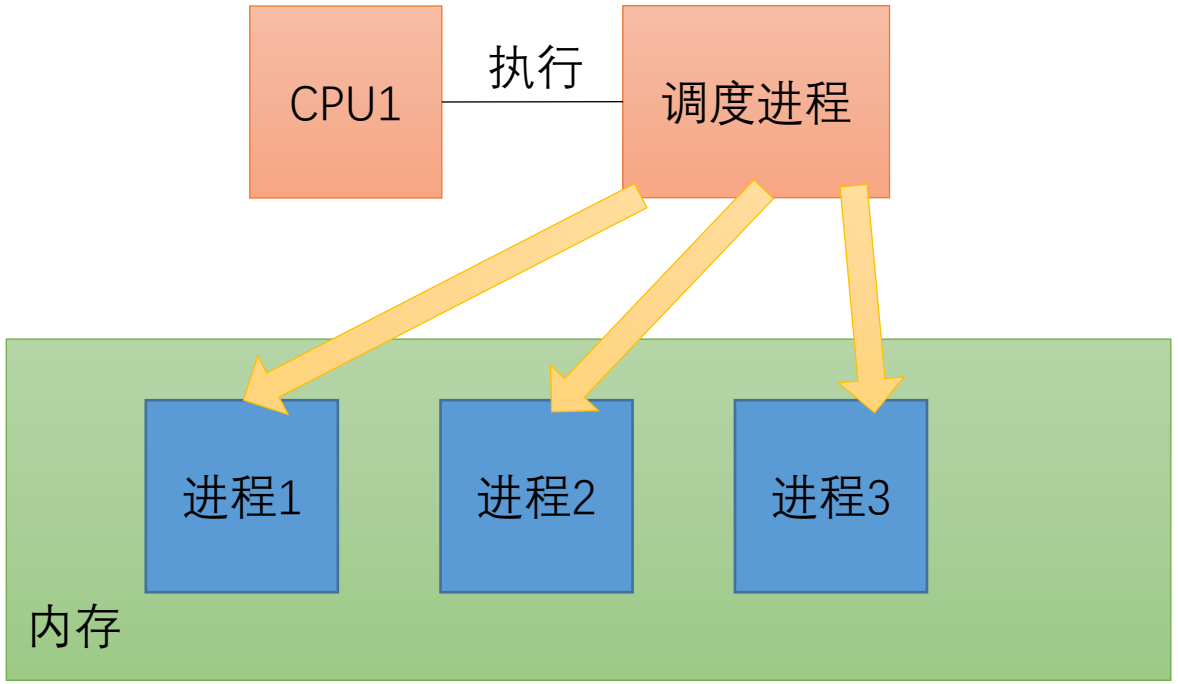
调度代码-C语言

```
Loop 进程1~n{
        acquire(p.lock)

        if p.state ok {
                // 切换上下文
                swtch()
        }

        release(p.lock)
}
```

RAII带来的问题（IV）



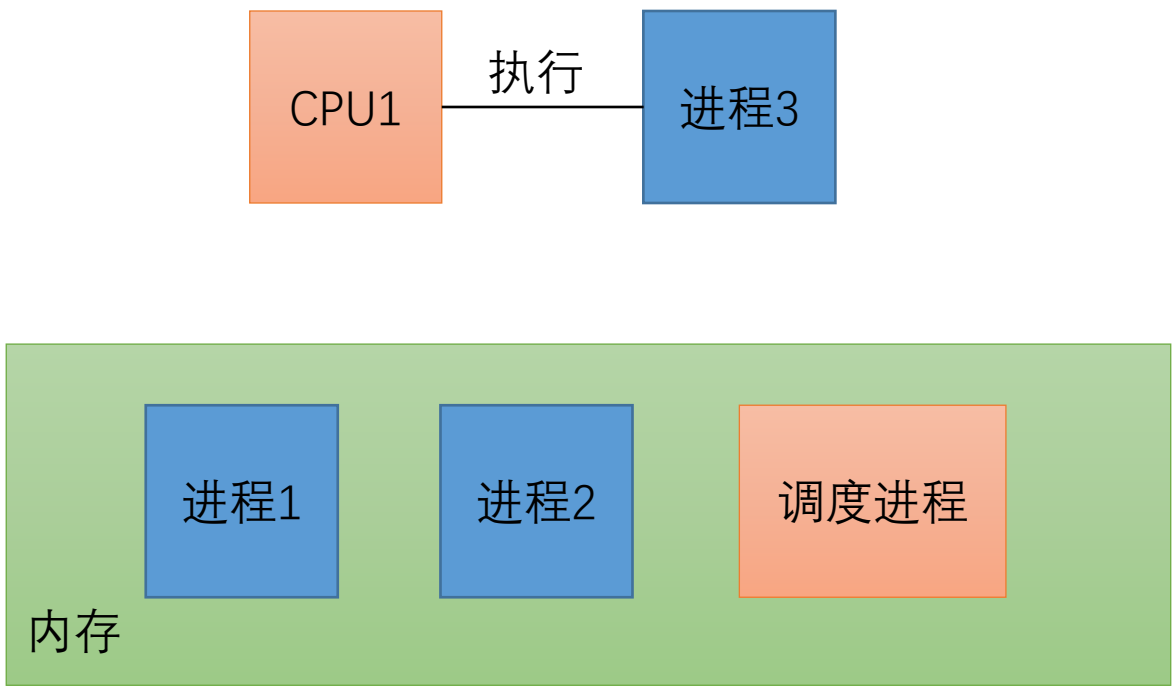
调度代码-Rust

```
Loop 进程1~n{
        guard = p.lock()

        if guard.state ok {
                // 切换上下文
                swtch()
        }

        drop(guard)
}
```

Fork_ret怎么解锁?



- 新创建的进程先执行fork_ret()
- 目前的方案：在SpinLock中额外给fork_ret开后门
- API：
  - p.unlock()

Xv6中的另一个同步原语

- Sleeplock
- API: acquiresleep, releasesleep
- 需要传递lock：Xv6中为了sleep和wakeup配合(sleep前不错过wakeup)，仍使用p.lock来保证，即sleep和wakeup前都需要上该锁，所以xv6先获取进程锁再释放sleeplock.
- 使用RAII的话，可以：
  - 传递LockGuard
  - 使用新的设计

Box

- Rust标准库中用于用户态程序堆分配的一种智能指针，同样是RAII
- 目前的实现：封装在linked list allocator之上
- 未来可以：分别实现两个智能指针，一个封装buddy system allocator，另一个封装slab

内存和并发安全

- 激发更多关于内存和并发的思考
- 内存：借用+Rc, RefCell, Arc, Mutex等智能指针
- 并发：存在并发相关的Send，Sync语义
- 更多在于预防，适合更大的项目

项目目前进展

- 文件系统尚不完整（WIP）
- 内核态第一次返回用户态进行初始化
- 目标：达到xv6的完整度，作为高校教学OS

赛题

- 与本项目一样：用Rust从零写xv6
- 添加智能指针，比如Arc和其他同步原语
- 实现伙伴分配系统管理内存
- 扩展文件系统和系统调用
- 移植xv6的十几个实验
- 支持Kendryte K210
- ...

谢谢聆听！