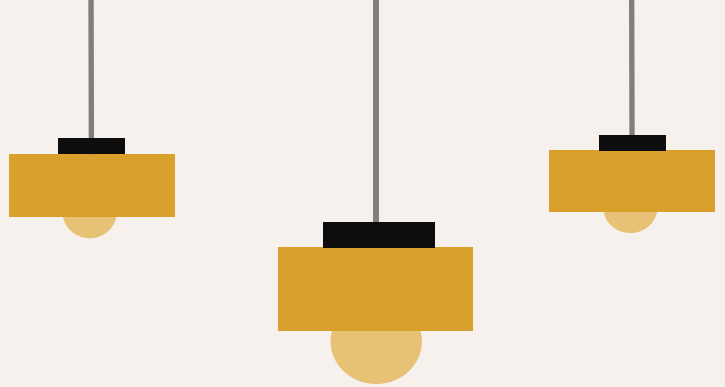


一种共享调度器的异步内核设计

洛佳

华中科技大学 网络空间安全学院

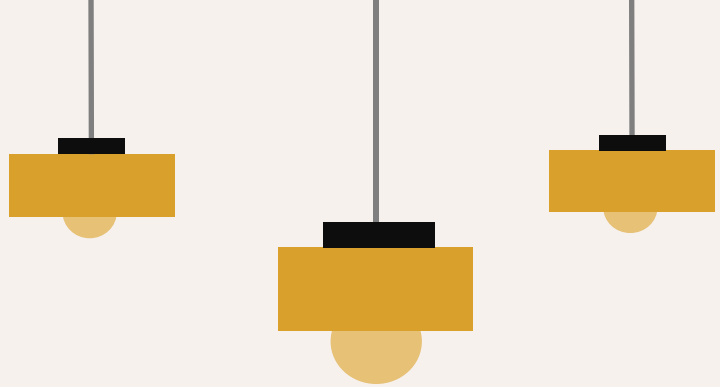
2021年4月

目 录

CONTENTS

介绍我自己

- 我叫蒋周奇，笔名是洛佳，华中科技大学网络空间安全学院，2018级本科生。
- Rust社区工作者。《Rust日报》编辑，翻译有《编写Rust语言的操作系统》。
- 热爱Rust语言与嵌入式、操作系统开发，热爱RISC-V架构，RustSBI项目作者。
- 3年Rust开发经验。曾开发“核能”和“科洛桑”游戏服务端引擎，与网易公司商业合作。



第一节

异步内核的设计思想

全新的内核设计方式

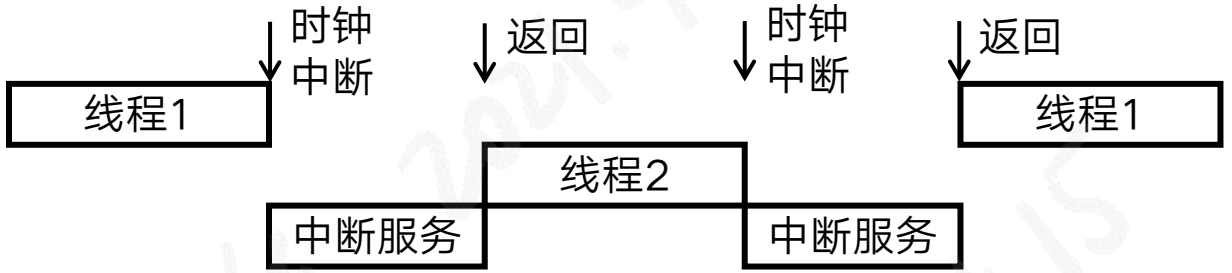
内核中的调度对象

- 异步内核是为处理任务而设计的
  - 传统内核的线程长期存在于系统内；新的编程技术提出了协程，是较短的不常驻的工作片段
  - 线程和协程可统称为任务，它们都是时间资源的组织形式
- 任务的优先级、当前状态和表示方式称作任务的元数据
  - 调度器根据元数据，决定下一阶段应当运行的任务
- 根据任务所占空间资源的不同，粗略划分任务到不同的地址空间
  - 地址空间中的任务共享同一组资源或映射方式，相互切换的开销较小
  - 任务所属的地址空间被标记再元数据中，以便调度器解释和计算

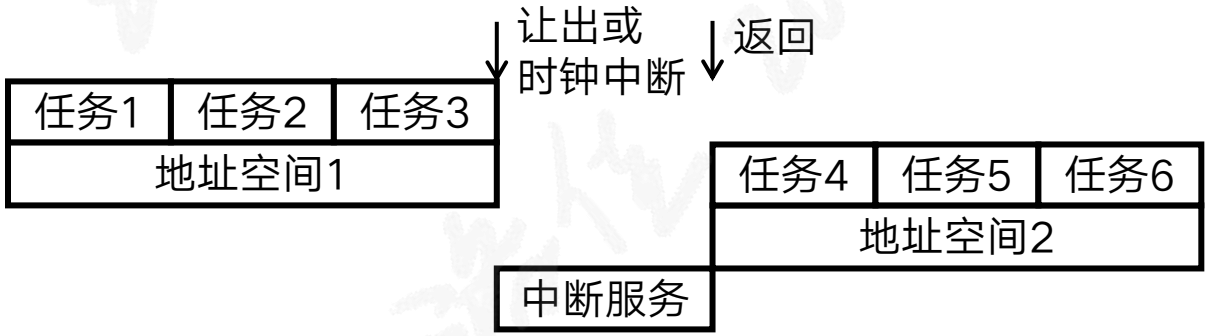
如何运行异步任务

- 传统内核中线程独占计算资源
  - 介入方式：时钟中断
- 引入“让出操作”
  - 任务已完成或等待资源时使用
  - 一定情况下无需保存、恢复现场
- “保底机制”是时钟中断
  - 阻止过长的任务占用处理核时间
- 调度器需要特殊设计
  - 尽量同地址空间，避免频繁让出

传统内核运行任务



异步内核运行任务



路佳 2021.4.15

共享调度器

- 传统的操作系统内核中，调度器只在内核中运行
  - 内核中执行线程，线程的内容完全由用户给定，不涉及调度运行的逻辑
- 异步内核中，用户层也要完成任务的切换过程，就要求用户层也运行一个调度器
- 方式一：共享参数的分离调度器，如rCore操作系统团队的aCore内核
  - 类似于许多共享内存的输入-输出接口，以共享内存的信息为媒介与内核交流
  - 无论共享信息为何，用户不能干扰内核的运行，安全性较高
- 方式二：合并的共享调度器，本次设计将会使用
  - 用户、内核共享代码和数据，软件工程学上拥有稳定的数据接口

调度器与执行器

- 共享调度器输出执行、让出或结束
- 执行器包含执行任务的逻辑
  - 负责解释任务，解释后才能执行
  - 每个地址空间都有一个执行器
- 安全性：若遇到不合规则的执行器
  - 错误让出：用户将损失时间
  - 错误执行：“保底机制”
- 执行器与用户程序共同编译，构成编程语言标准库的一部分

弹出任务（共享调度器s，切换判断函数f）→ 任务结果r：

尝试从s中弹出新任务t。

如果存在任务t：

如果f(t)为“应切换地址空间”：

返回“应执行让出操作”

否则：

返回“应执行任务t”

如果不存在任务：

返回“没有任务了，应结束执行”

注：切换判断函数f（任务t）→布尔值。整个弹出任务函数在用户层的执行器中运行，它的返回值将决定执行器下一步的动作。

向调度器添加任务

- 新建任务的步骤包含执行器、调度器两步
  - 创建新任务，并保存到执行器中。创建完毕后，得到任务的表示方式；
  - 以表示方式为索引，创建任务的元数据。将元数据添加到调度器中。
- 执行任务时，元数据在创建任务的地址空间，能被还原为元数据，进而提取出任务，开始运行
- “生成”原语——异步编程核心思想的直观体现
  - 分支操作：生成一个新的任务，分担本任务的工作
  - 回调操作：本任务结束后，接受返回数据，并做后续的处理工作
- 将新建过程封装为异步运行环境，提供给应用层用户，就可以在应用层使用异步任务了

进程与地址空间

- 地址空间是内存资源映射和分配的因素
- 内核中的空间资源仍然需要谨慎考虑，就可以引入传统内核中“进程”的概念
- 一个地址空间可以包含多个进程

内核本身的异步设计

异步文件系统

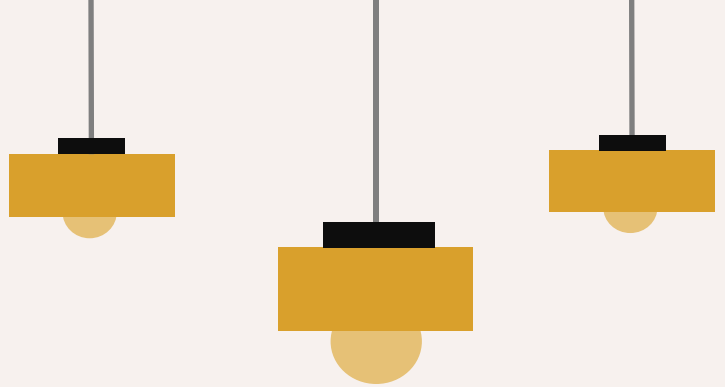
和硬件层异步结合更紧密、块设备交互效率更高

异步网络层

契合硬件缓冲区的异步操作、流畅的回调设计

更多内核模块

内核也将受益于本身提供的运行时，提高运行效率



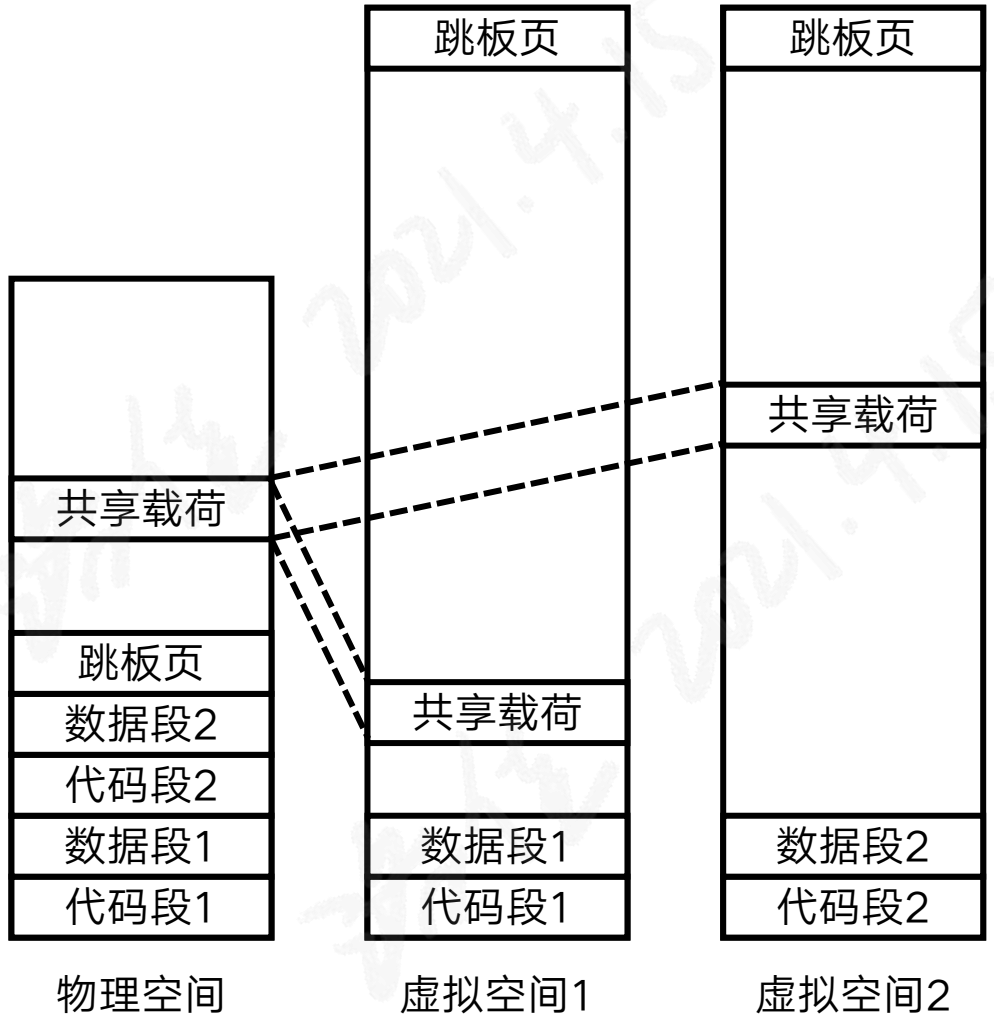
第二节

异步内核的实现与运用

如何实现异步内核到具体架构

实现共享调度器

- 有虚拟内存，基于页表的虚拟内存
  - 页表的多重映射
  - 共享数据段、共享代码段
- 调度算法打包为共享载荷
  - 同时映射到内核、用户地址空间
  - 可根据具体场景更换载荷
- 用于添加任务、取出下一个任务
- 没有虚拟空间也可共享调度器
  - 或者不隔离，或者简单隔离

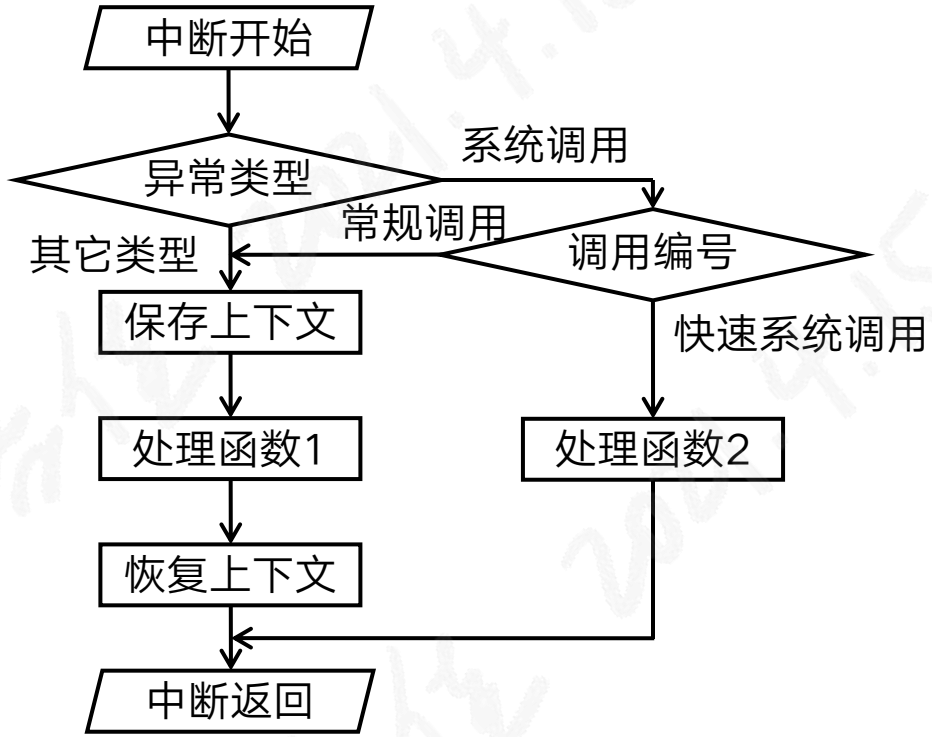


路佳 2021.4.15

快速中断处理

- 传统内核中，时钟中断可在任一时刻发生，但线程必须仍可运行，因此一般都应保存完整现场
- 引入“超时”：协程内核中，任务超出时间片，必须强制介入而切换任务的情况
  - “超时率”：所有切换任务中属于超时的比率。越低越好，是内核实时性的一种数值表现
  - 若一组任务的超时率为1，说明它属于完全抢占调度；若为0，说明属于完全的协作调度
- 协程内核的陷入只有任务超时时需要保存现场
  - 此时任务被强制介入，和传统内核的线程相似，它的现场被保存在一处
  - 恢复超时的任务，也必须恢复它对应的现场
- 若未超时，任务的主动让出说明它的现场已经结束，可以丢弃此现场，也无需恢复现场

例子：快速系统调用



特殊的中断处理通路

```
156 #[naked]
157 #[link_section = ".text"]
158 pub unsafe extern "C" fn trap_entry() -> ! {
159     asm!(
160         ".p2align 2",
161         "csrrw  sp, sscratch, sp",
162         "bnez   a7, 1f", // 不用保存上下文的条件：a7=0 且 scause=8
163         "csrr   a7, scause", // 此时恰好a7的值已经使用过了
164         "addi   a7, a7, -8
165         bnez    a7, 1f",
166         "call   {fast_syscall}", // a0-a6未被更改，恰为该函数的参数
167         "unimp", // 非法指令。fast_syscall函数不可能返回，应当使用其它
168         "1:", // 需要保存上下文
169         "addi   sp, sp, -33*8",
170         "sd     x1, 0*8(sp)
171         sd      x3, 2*8(sp)
172         sd      x4, 3*8(sp)
173         sd      x5, 4*8(sp)
174         sd      x6, 5*8(sp)
175         sd      x7, 6*8(sp)
```

代码实现例子

让出操作与生成原语

- 协程内核的让出操作实现为系统调用
  - 陷入内核，内核应当运行共享调度器，得到下一个任务和它所在的地址空间
  - 内核切换到下一个任务的地址空间，唤醒该空间用户的执行器，进而执行下一个任务
  - 在完整的设计中，应用类似于“快速系统调用”的方式实现，能显著提高效率
- 生成原语由用户层的执行器实现
  - 包装任务为结构体，在用户的地址空间下，得到任务的表示方式，通常是一个地址
  - 其它地址空间下，不能解释这个地址的意义，恰好切换到本地址空间时，就可解释意义
  - 将这个地址包装为任务的元数据，添加到共享调度器中，即完成生成过程

实现执行器

- 执行器的功能：运行已创建的任务
  - 不同的编程模型不同，但最终能归纳到异步任务上
  - 适当时候执行让出调用
- 操作“运行任务”的简单实现
  - 钉在栈上，创建上下文，运行
- 复杂实现：丢弃现场的让出操作
  - 切分切换用户栈
  - 这种方法很有挑战性
- 执行器通常由用户层的编程语言实现

用户层启动函数（）→ 进程的返回值：

包装入口函数为入口函数任务t_0。

得到共享调度器s；将任务t_0加入s中。

循环：

尝试从s中弹出任务结果r：

如果r为“应执行任务t”：

运行任务t

如果r为“应执行让出操作”：

执行“让出”系统调用

如果r为“没有任务了，应当退出”：

退出循环

返回t_0的返回值为进程返回值

异步标准库

- 调度器、执行器被包装好，以相应的编程模型暴露到应用开发者
  - 编译型语言提供异步入口函数，或者有协程的运行时语言提供协程运行环境
  - 将整个操作系统作为运行时
- 异步的系统调用封装
  - 文件、网络等等全都作为异步函数提供
- 使用内核提供的“任务”
  - 高效的写法中，我们将任务包装为协程，提供一个生成函数，生成新的协程
  - 如果为了兼容性，可以视任务为线程，退化到传统的线程内核，仍然可以运行

异步内核的应用场景

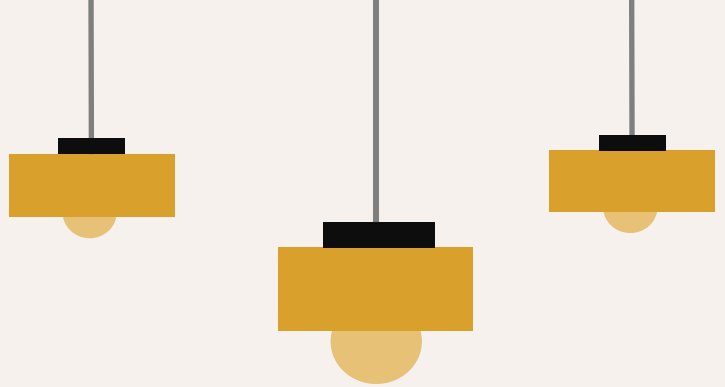
在哪里使用异步内核？

桌面和服务器平台

使用共享调度器、快速让出调用，相比传统节省一层运行时，提高现代高并发应用的执行效率，增加平台用户的生产力。

嵌入式和实时性系统平台

高实时性的开发方法，适应性价比设备的内核设计，多种载荷可更换，不同场景中自由选择各个模块，提高应用的实时性。



第三节

软件生态中的异步内核

从软件工程学改进内核设计

共享调度器的兼容性设计

- 操作系统内核会更新，调度器也是
  - 共享参数的分离调度器：版本不宜多，开发者应共同协商，得到稳定设计，才便于推广使用
  - 合并的共享调度器：代码、数据由共享载荷定义，共享数据由共享代码解释
- 用户层执行器与共享调度器分离
  - 共同构造最简单的异步编程概念
  - 便于用户应用的执行器更换和更新，用户可自由选择需要的编程技术
- 兼容性设计有利于在保证协程内核灵活性的同时，打造内核统一的生态环境，进一步降低应用编写成本，提高开发效率

异步内核与编程语言技术

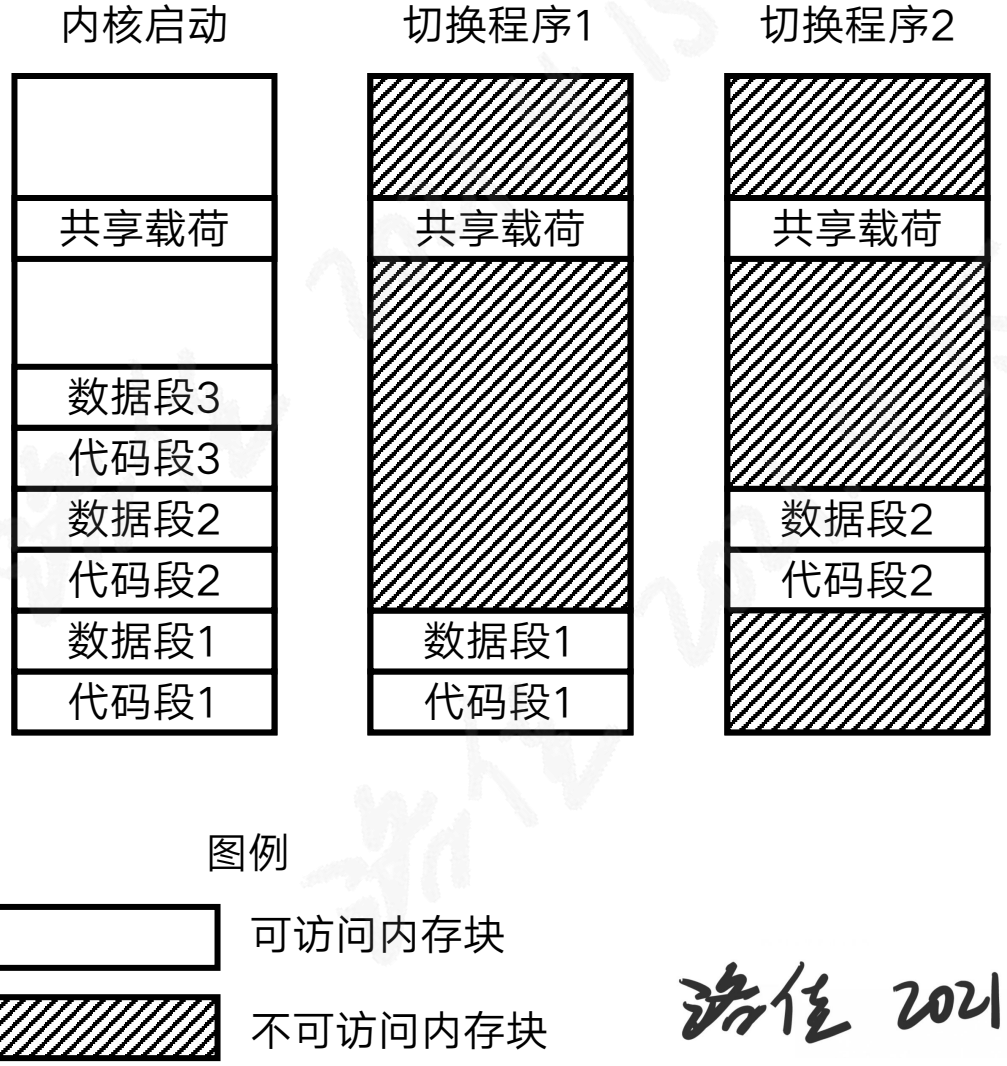
- Rust语言提供async/await无栈协程模型
  - 异步内核直接提供用户层的async fn main()异步入口函数
  - 通常情况下，一个任务会复用上一个已结束任务的栈。只要发生超时，内核就会新建一个栈。只要超时的任务恢复了，它的新建栈就会被销毁。从长时间来看，符合无栈协程的思想
  - 提供生成函数，类似于std::thread里的spawn函数
- Go语言提供协程运行时
  - 基于异步内核直接实现协程，绕过传统内核的线程概念
- 更多与异步开发有关的编程技术，可以受益于异步内核带来的性能提升

嵌入式平台的异步内核

- 没有虚拟空间，也可以共享调度器，需要一些机制实现权限隔离
  - 方法一：不隔离。适用于原型设计开发、硬件机制安全性强的场景
  - 方法二：使用简单寄存器隔离。如果指令集架构提供此类寄存器，则可划分物理内存的权限
- 拍扁“地址空间”，只存在一个物理地址空间，直接在上面运行所有进程
- 操作硬件，使用裸机的内核模块
  - 裸机的文件系统和网络协议栈，得益于异步编程，可节省处理核等待的时间
- 配置稍高的嵌入式平台，需要安装应用程序
  - 重定向安装，映射共享调度器。更换共享载荷，获得更高的安全性或极致的性能

例子：嵌入式平台的共享调度器

- 可使用简单寄存器隔离内存
  - 每个应用拥有同一组寄存器设置
  - 不可访问其它应用的资源
  - 都能访问共享载荷
- 如果选择不隔离内存……
  - 较为简便，适合原型开发
  - 硬件烧录的安全机制
- 使用共享载荷，运行协程等异步应用
- 根据应用的实际需求，选择合适的共享载荷，和异步内核搭配使用



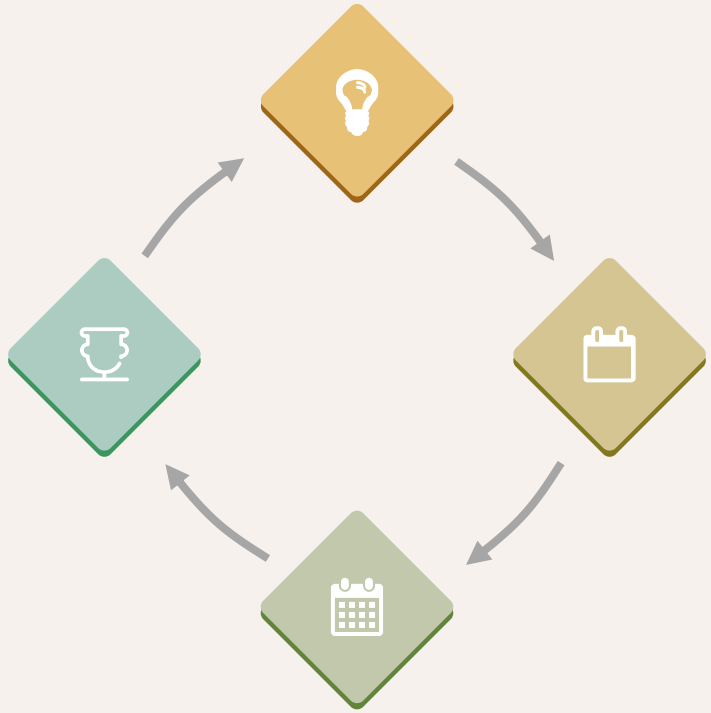
路佳 2021.4.15

异步内核的安全性

- 载荷是全地址空间可见的
  - 初步特权隔离：载荷代码段设置成只可运行，不可读、写
  - 数据段仍然是全空间可读写的，未来的指令集架构或许可以改善此类设计的安全性
- 错误编写执行器的时间安全性
  - 不断让出来堵占处理核时间
  - 使用统计或其它方法判断此种用户行为
- 向调度器添加了过多任务
  - 行为判断、优先级调整等等

飓风内核

一款实验性的异步内核



原生异步

内核共享调度器，提供初步的异步抽象

安全高效

使用Rust编写，支持最新的RISC-V架构

生态灵活

根据应用，自己选择搭配的调度器载荷

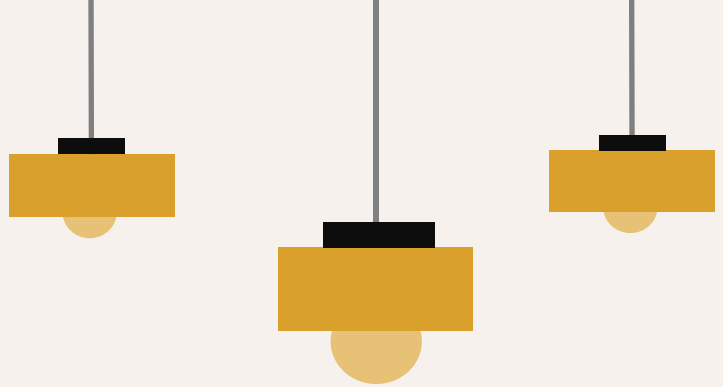
兼容性强

采用异步内核的各类兼容性设计

已开源，欢迎点星：https://github.com/HUST-OS/tornado-os

致谢

- 感谢我们“无相之风”战队的队友，包括车春池同学、王清雨同学等等，感谢我们的指导老师邵志远老师，有队友们的帮助，异步内核的想法才最终实现到项目中。
- 感谢参与本次比赛的其它队伍。队伍之间的交流也让我们能交换灵感。
- 感谢选题之前许多学长的谆谆教诲，我能改正一些想法，未来修好项目中的更多问题。
- 感谢在我学习路上无数帮助过我的老师们，还有比赛组委会提供的参赛机会。



感谢您的观看

一种共享调度器的异步内核设计

洛佳　2021年4月

华中科技大学 网络空间安全学院