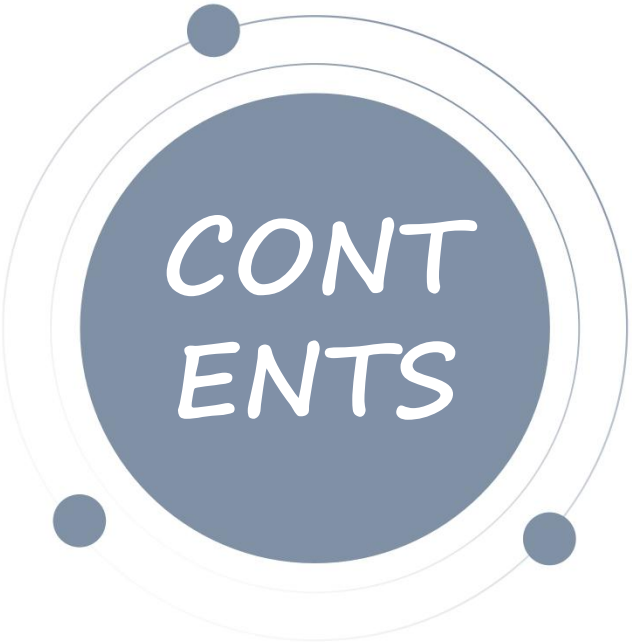
Mini-Rust-OS

一个x86-64架构的混合内核

黑袍纠察队-国防科技大学

2024-08-14



01

项目简介与完成情况

项目简介

1. 选题：proj278，用Rust语言开发一个完整的小巧的操作系统内核，提供基本的中断/异常处理、内存管理、进程管理等功能。
2. 目前已经有很多使用Rust开发的宏内核了，我们的侧重点在于对内核架构的探索。
3. 尝试权衡系统性能与安全性。
4. 设计实现了一种结合宏内核与微内核优点的混合架构模式。

完成情况

从零实现了一个x86-64架构的混合内核，目前支持Qemu，单核。

支持并发原语（信号量、互斥锁、条件变量），Shell程序支持管道、重定向。

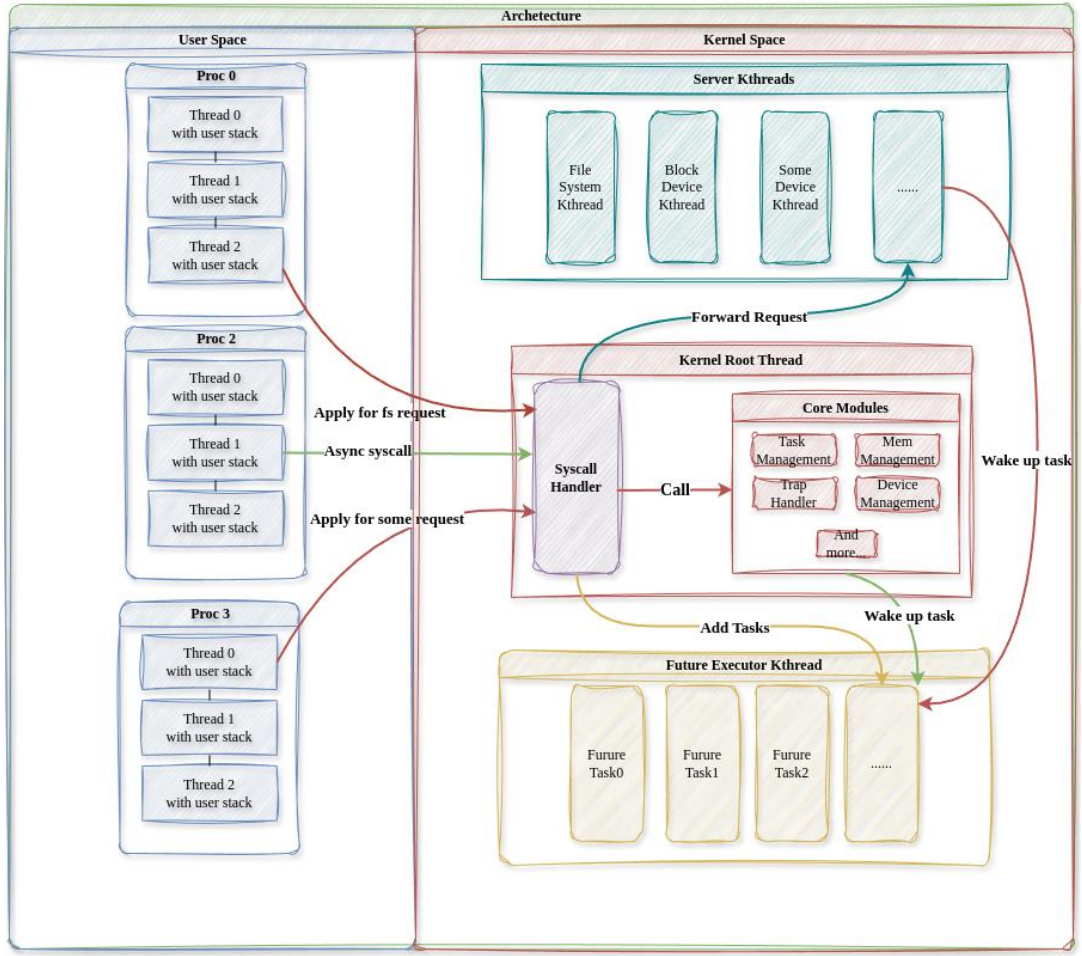
利用Rust无栈异步协程实现了多对多线程模型。

内核线程崩溃时，内核保持稳定，且可以重启内核线程。

具有内存管理、进/线程管理、文件系统、驱动管理、同步互斥等基础模块。

编写了一份在线文档NUDT-OS-BOOK，详细描述了各个模块的工作流程。

设计内核服务线程实现混合内核架构。



在线文档：[NUDT-OS-BOOK]

效果演示：[效果演示]

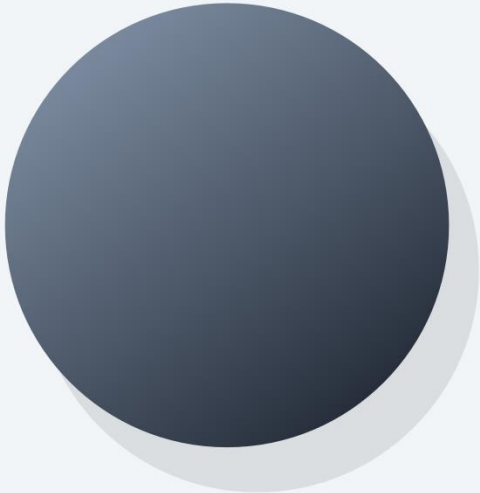
02

设计原则

一对一线程模型 VS 多对多线程模型

一对一线程模型要求每个用户线程有自己的内核栈，当发生阻塞系统调用时，OS将阻塞线程的内核态现场保存在内核栈上，并调度另一个用户线程运行，从而提高系统性能。

缺点是必须限制内核线程的数量或者减小内核线程栈的大小。且内核线程栈的上下文切换带来了一定的开销。



利用Rust无栈异步协程，所有用户线程共享内核栈，上下文保存在协程中。

节省了空间，一定程度上减少了上下文切换的开销。

内核线程服务模型

传统宏内核如Linux将所有的内核服务都集成在内核态中，内核中**任何一个模块崩溃就会导致内核本身崩溃**，无疑降低了系统的安全性，同时内核的**可维护性和可拓展性也较差**。

微内核将非核心的系统服务全部放置在用户态，以独立的进程运行，进程有相互隔离的地址空间，提高了系统的安全性，但**用户程序请求服务需要频繁的进程间通信，导致了性能较差**。

我们在宏内核与微内核之间采用一种折衷的方案，将**非核心的系统服务放置在内核线程中运行**，内核线程共享内核地址空间，但保持相对独立，**有自己的控制流和内核栈，内核服务线程崩溃时，内核本身不会崩溃**，且可以**尝试重启内核线程以恢复服务**。

03

异步协程与协程执行器

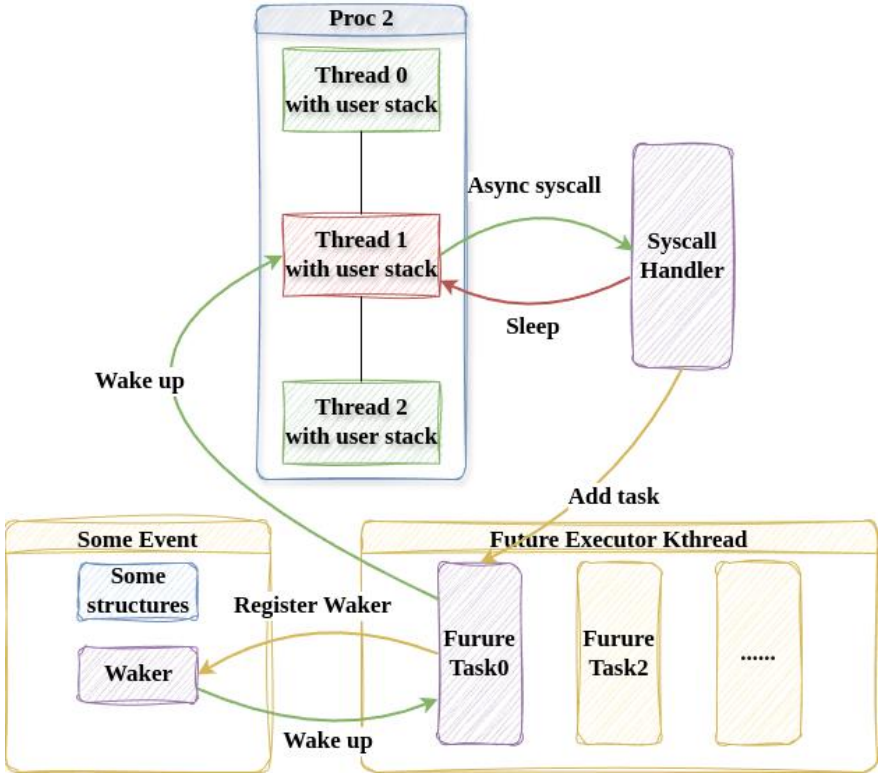
异步协程与协程执行器

- 当用户线程阻塞系统调用时，OS将阻塞线程挂起，切换到另一个线程运行，直到阻塞线程可用时才将其唤醒，并继续执行。

- 使用异步协程是为了不为每个线程分配单独的内核栈，线程上下文保存在协程中而不保存在栈上。

- 最终达到节省空间和节省上下文开销的目的。

使用异步协程

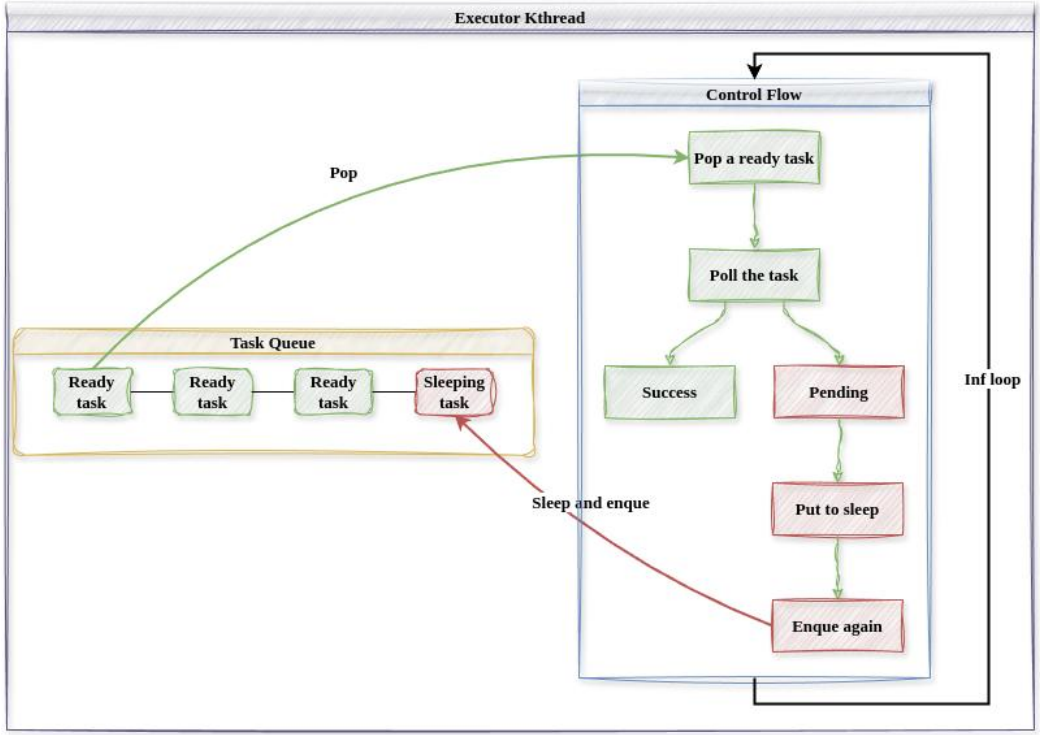


线程阻塞时**创建一个等待协程**，并进入异步等待。

全局协程执行器轮询内核中的所有协程，**协程准备就绪时，唤醒等待线程**。

协程在第一次被轮询时，若尚未准备就绪，则**将自己的唤醒器注册到被等待事件中去**，被等待事件完成时，**使用唤醒器唤醒协程**。（**每个等待协程最多被轮询两次**）

协程执行器

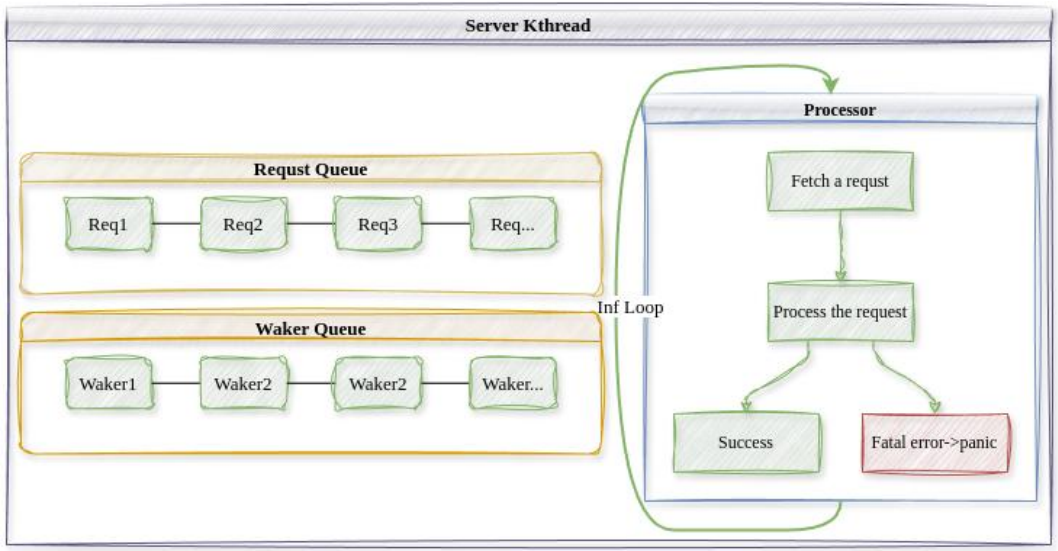


协程执行器在一个单独的内核线程中运行，无限循环，处理内核中的所有协程。

04

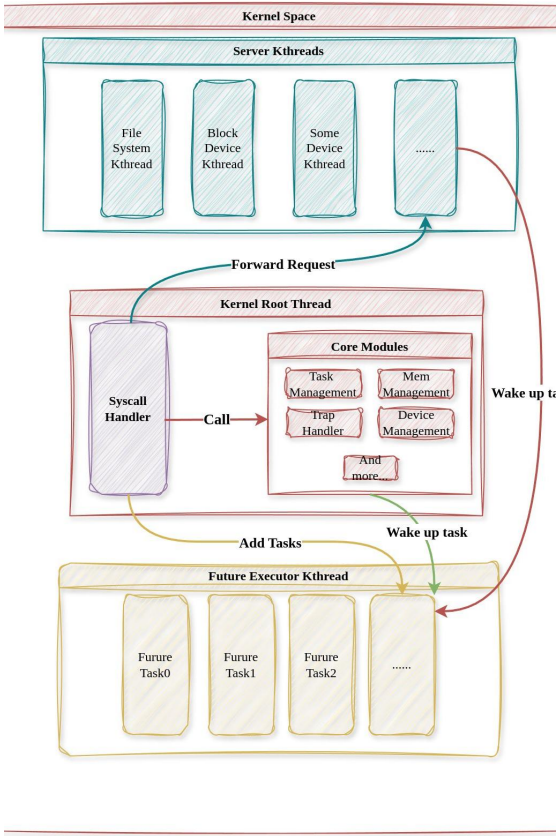
内核服务线程

内核服务线程



- 一个内核线程提供一种独立的内核服务，内核服务线程有独立的内核栈和控制流。
- 我们认为内核线程中运行的代码是不完全可靠的。
- 一个内核服务线程崩溃时不会影响其他线程，且内核能够尝试重启内核服务线程。

用户请求内核线程服务



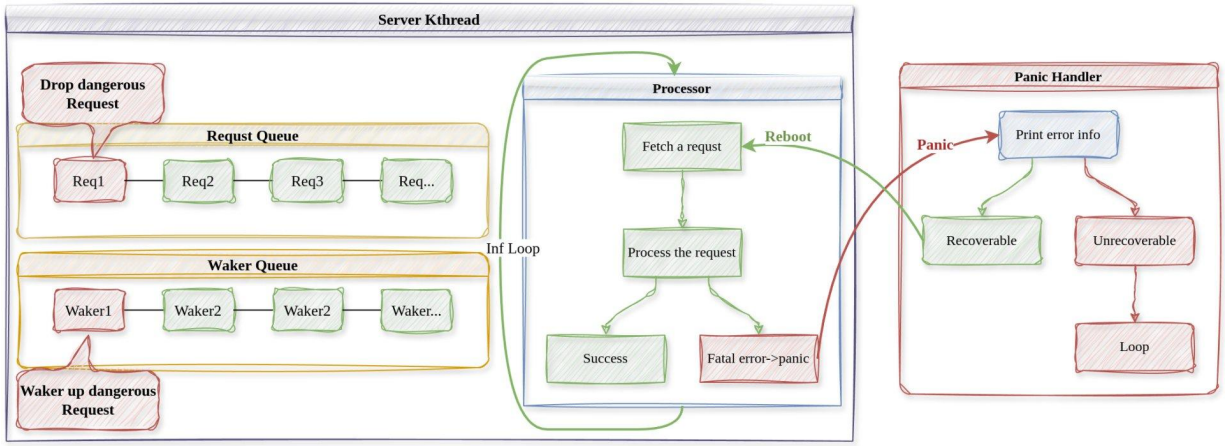
用户和内核线程约定好请求的格式，用户构造好请求后，将其转化为字节并发送到内核服务线程的请求队列。

内核服务线程从字节解析出具体的请求，分情况进行具体处理。

用户发送请求后创建等待协程，进入等待状态，内核服务线程服务完毕后唤醒等待协程，并唤醒等待线程。

内核线程故障恢复

内核线程内部的不可靠代码可能会出现故障，Rust语言触发panic，控制流进入panic handler中。



panic handler判断错误源是否是内核服务线程，若是，则尝试恢复内核服务线程，丢弃错误的请求，从下一个请求开始继续处理。

05

后续工作与改进

后续工作与改进

- 目前只支持单核，后续希望添加多核支持
- 支持真实硬件
- 添加更多内核服务线程（驱动程序）
- 对系统进行压力测试与性能测试
- 改善文件系统
- 添加C语言用户程序支持
- 移植编译器、标准库和图形程序库

THE END

THANKS