多形态的操作系统——ArceOS 的微内核改造

操作系统结课报告

蒋昊迪

2023 年 6 月 10 日

* 完整报告：https://github.com/jhdjames37/arceos/blob/main/log/report.md

概述

微内核

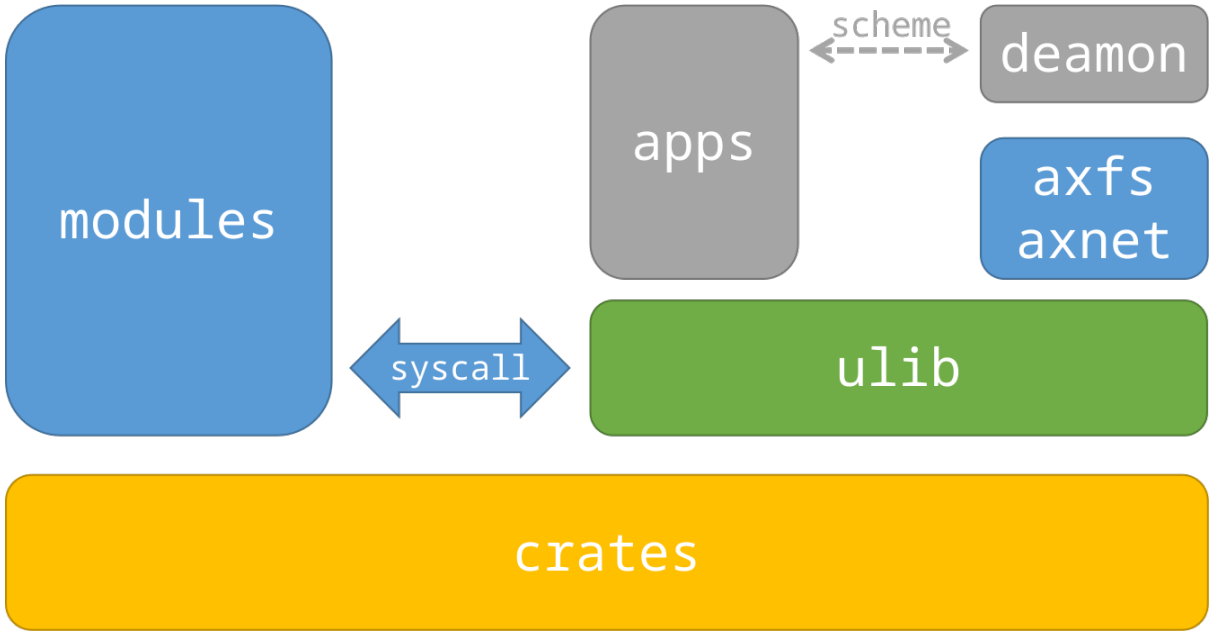
- 将非核心模块（各类驱动，FS，网络栈等）移至用户态
- 模块化：更容易定制化，易于维护
- 安全：减少内核态的代码量
- 稳定：外围驱动崩溃不影响内核运行
- 性能：带来更多上下文切换
- 典型的微内核
  - seL 4
  - Redox
  - Ziricon

设计

- 基于 ArceOS
- 微内核架构
- 借鉴 Redox 的 Scheme 模式实现 IPC
- 网络和文件系统在用户态运行

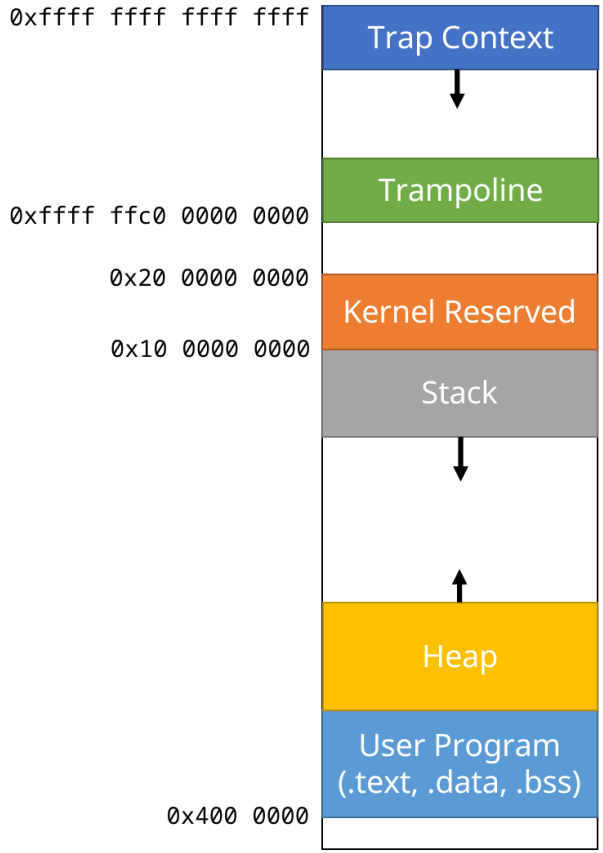
架构

- 分离特权级
- 移植内核模块



内核机制

- 特权级切换
  - 参考 rCore 的切换处理
  - S 态中断支持
  - 系统调用处理
- 地址空间
  - 双页表模式
  - 动态增删地址段
  - 支持用户堆
- 进程、线程管理
  - 进程的独立地址空间和文件表
  - 树状的进程关系
  - 实现了 fork, exec, exit, wait 等调用
- 同步互斥
  - 基于 futex 机制实现

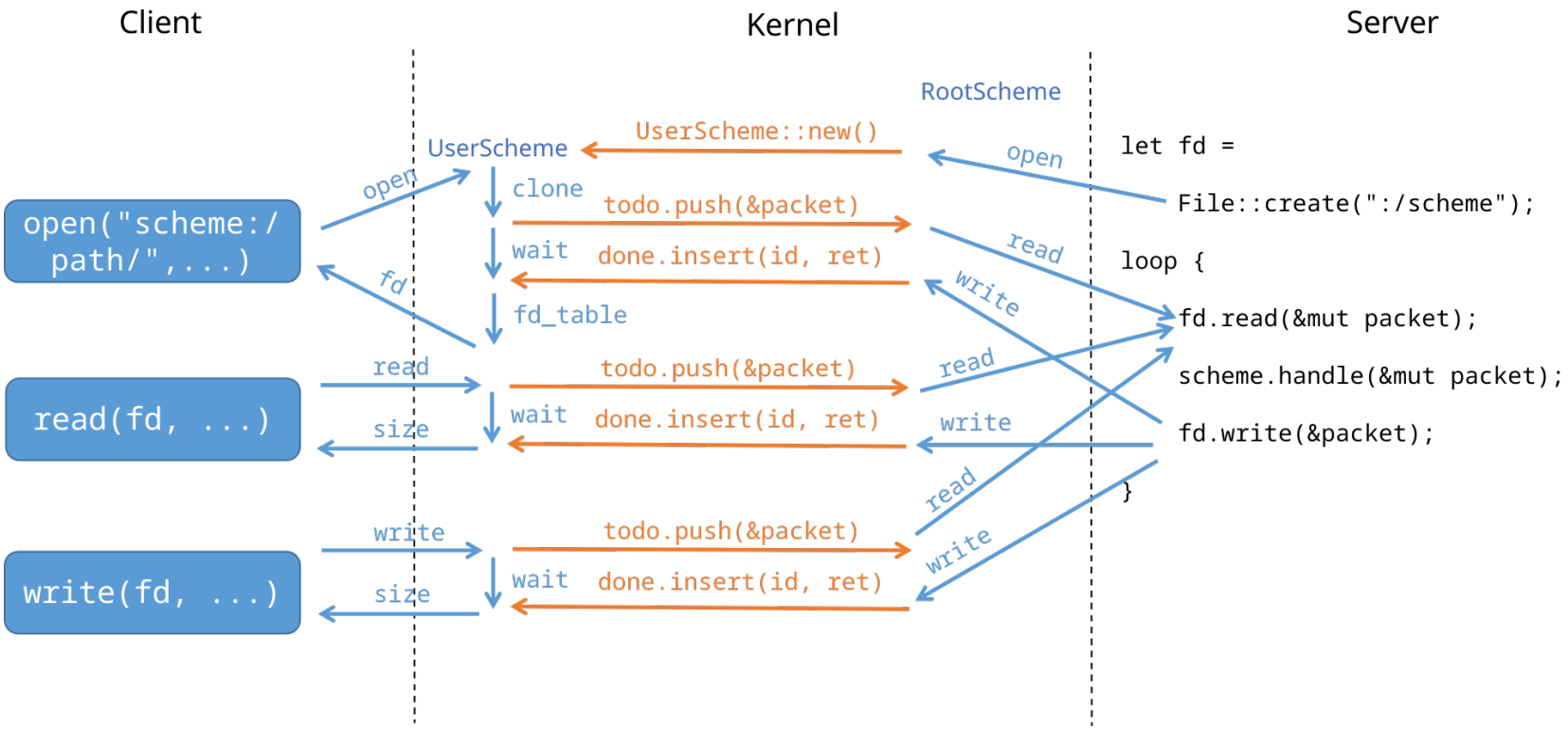


特色 IPC - Scheme - 简介

- Redox 的 IPC 机制
- 重新定义文件 I/O
  文件 I/O 的不同接口可以被赋予不同的含义
- 格式类似于 URL `scheme:/path`
  - scheme 对应了不同的系统服务
  - path 则对应了服务提供的不同接口（和参数）

特色 IPC - Scheme - 实现

- syscall 层根据文件表，将请求转发给 Scheme
- 简单的 Scheme：stdin, stdout
- 用户态 Scheme：
  - Root Scheme(`:/scheme`)：注册服务端
  - User Scheme：用户态 Scheme 的内核抽象
- 地址空间
  需要进行两次拷贝，并由内核提供临时空间。


Client
Kernel
Server
RootScheme
UserScheme
UserScheme::new()
open("scheme:/path/",...)
open
clone
todo.push(&packet)
wait
done.insert(id, ret)
fd
fd_table
read(fd, ...)
read
todo.push(&packet)
wait
done.insert(id, ret)
size
write(fd, ...)
write
todo.push(&packet)
wait
done.insert(id, ret)
size
open
read
write
read
write
read
write
let fd =
File::create(":/scheme");
loop {
fd.read(&mut packet);
scheme.handle(&mut packet);
fd.write(&packet);
}

用户态服务

- 核心在于修改接口
- 向下调用的接口：
  - axtask, axsync, etc: 使用系统调用
  - axdriver: 在内核创建 Scheme
- 向上提供的接口提供 Scheme 服务

应用

- 实现了基于 syscall 的用户库
- 移植了 httpserver, httpclient, shell 等应用
- 能够从硬盘加载应用。

性能

- 测试了 HTTP Server 的吞吐性能

	microkernel	unikernel
吞吐 (packets/s)	470.41	8573.04
延迟 (ms)	2.126	0.117
传输速率 (KB/s)	194.78	3549.77

展望

- 更加完善的用户态支持。
  - 信号、管道等机制
- 更具兼容性的用户库。
  - 与原有 libax 兼容
  - 符合 Rust std 接口
- 更多的微内核特性实现。
  - Redox 的 fork
  - axdriver
- 基于微内核的系统优化。
  - 优化现有实现
  - 用户态中断

总结

- 实现了一个简单的微内核架构系统
- 可以在两种架构之间自由切换
- 模块化架构便于迁移