

共享调度器技术报告

项目成员：赵方亮、廖东海

导师：向勇教授

清华大学

2023 年 3 月 26 日

目录

1 项目背景

2 研究工作

3 性能评估

4 未来展望

项目基础

用户态中断 [1]

- 外部中断
- 内核向用户进程发送中断

Rust 协程机制 [2]

- `async/await`
- Executor 运行时

项目整体概况

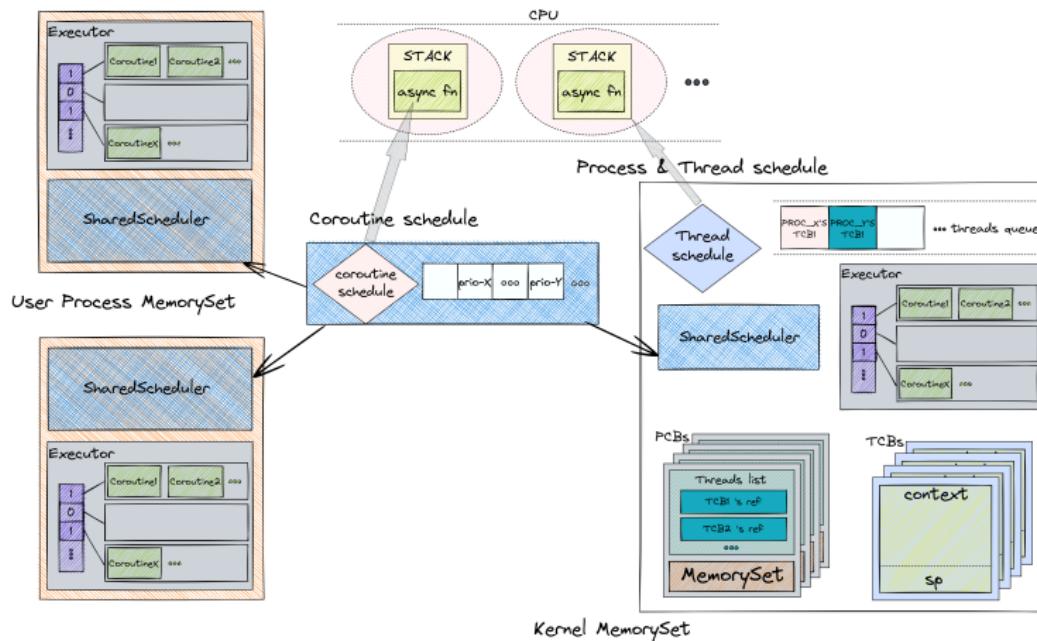


图 1: 项目整体概况

已完成的研究工作及成果

已完成的研究工作简介

- 任务调度
- 系统调用
- 内核改造
- vDSO 机制
- 协程调度框架

任务调度

进程、线程调度

- 由内核中的调度协程进行
- 优先级：依赖协程优先级
- 线程：平等

协程调度

- 用户态：共享调度器
- 优先级队列

状态转换模型

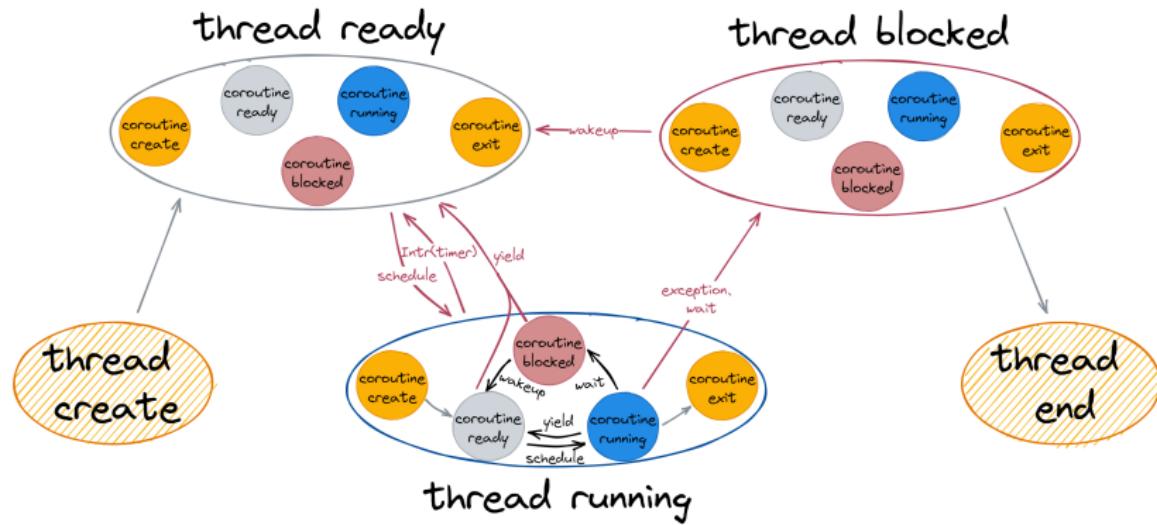


图 2: 状态转换模型

系统调用 (以 read 系统调用为例)

接口

同步与异步的接口进行统一，通过参数进行区别

- `read!(fd, buffer);`
- `read!(fd, buffer, key, cid);`

功能扩展

定义 AsyncCall 数据结构，实现 Future 特性，让系统调用阻塞在用户态

内核改造

内核调度协程

- 主动让权

系统调用处理

以 read 系统调用为例

- 创建内核协程：立即返回
- 唤醒用户协程：内核协程完成复制操作，发起用户态中断唤醒用户态协程

vDSO 机制

共享调度器由内核进行维护，以 vDSO 的形式暴露给用户进程

加载与动态链接

- 创建进程时映射共享调度器
- 查找进程符号表，进行链接

堆、Executor

- 全局堆分配器
- 全局 Executor

协程调度框架

协程调度框架依托于 Executor 数据结构，为其实现不同的成员函数，从而实现不同的调度算法

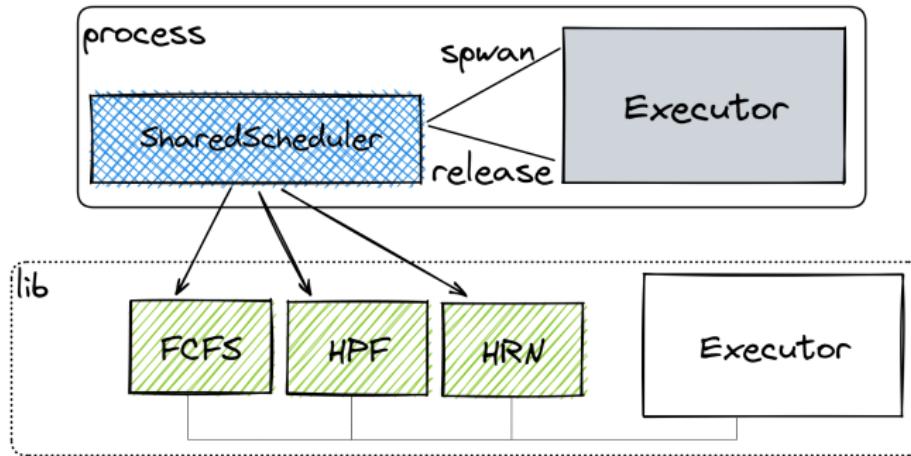


图 3: 协程调度框架

线程、协程对比

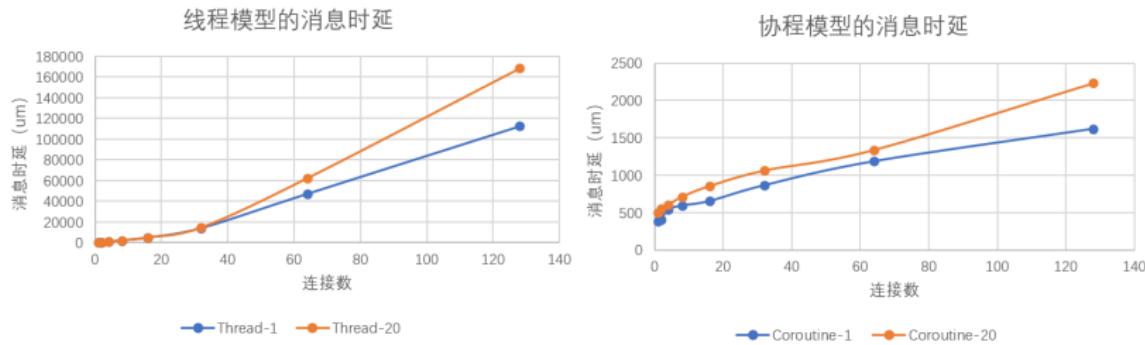


图 4: 线程、协程时延对比

- 连接数较少时，线程模型与协程模型持平，随着连接数增加，线程模型时延迅速上升，远高于协程
- 数据规模增大，线程模型时延增加幅度远大于协程模型

线程、协程对比

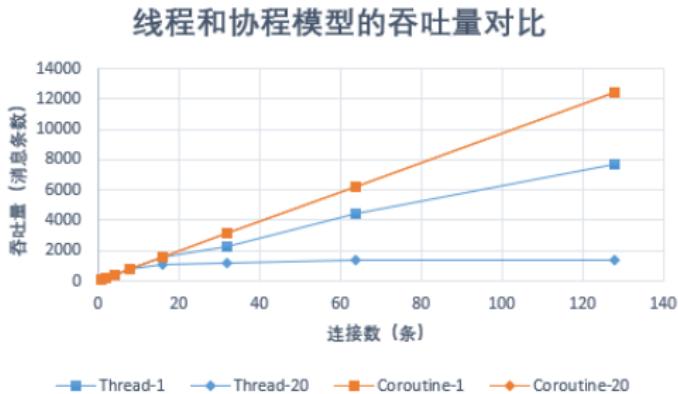
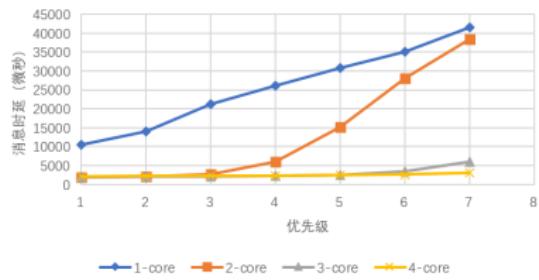


图 5: 线程、协程吞吐量对比

- 连接数增加，线程模型吞吐量增长缓慢

优先级

不同优先级的消息时延



不同优先级的时延抖动

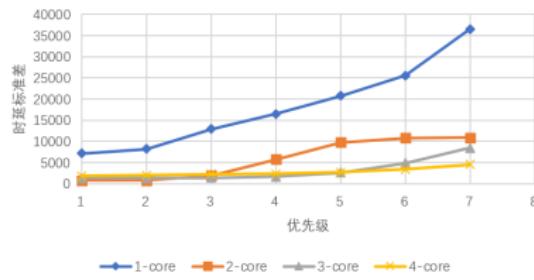


图 6: 不同优先级下的时延与抖动

- 随着 CPU 资源数量的增加，由于调度器同步互斥带来的开销，高优先级的连接出现小幅度的性能下降

优先级

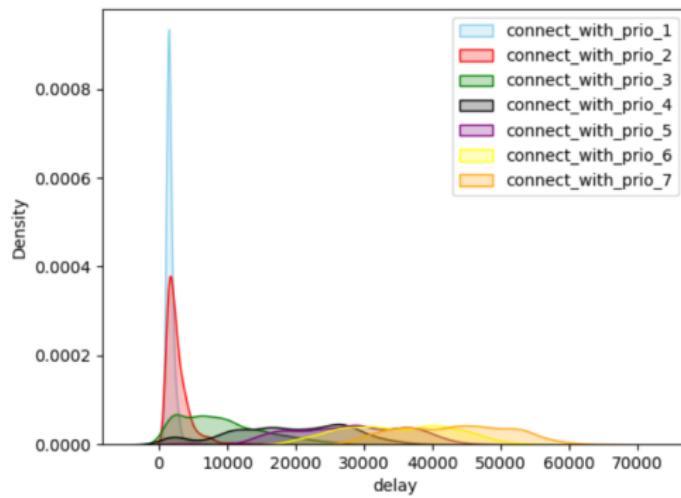


图 7: 不同优先级下的时延分布

- 资源有限时，保证了优先级高的任务

未来展望

下一步工作

- 文件系统扩展
- 进程、线程、协程调度

参考文献 |

- [1] gallium70. Risc-V Extension N Implementation[EB/OL]. .
https://gallium70.github.io/rv-n-ext-impl/ch2_1_n_ext_spec.html.
- [2] stevenbai. Futures Explained in 200 Lines of Rust[EB/OL]. .
<https://cfsamson.github.io/books-futures-explained/introduction.html>.

Thanks for your attention!

Q & A