
OSH 2022
MIDTERM REPORT
Runikraft
中国科学技术大学 2020 级 011 系
吴骏东 张子辰 蓝俊玮 郭耸霄 陈建绿
2022.4.25


目
录
01
What
什么是 Runikraft
02
Why
为什么要实现 Runikraft
03
How
如何实现 Runikraft
04
Q&A
问答


01
What
什么是 Runikraft

1.1 Runikraft 简介

Runikraft 是用 Rust 语言编写的能在 RISC-V 架构 + QEMU 平台上运行的 unikernel。它基于用 C 语言实现的 Unikraft，在继承 Unikraft 的高效性、可定制性、良兼容性、安全性的同时，加入了 RSIC-V 支持，并且用 Rust 语言提供了更强的内核安全保证。与目前的绝大部分的用 Rust 编写的操作系统不同，Runikraft 将完全使用 stable Rust。

关键词：*Unikernel，Unikraft，Rust，RISC-V*

Runikraft

Unikernel

Unikernel 是专一用途的、单地址空间的轻量操作系统。

Unikraft

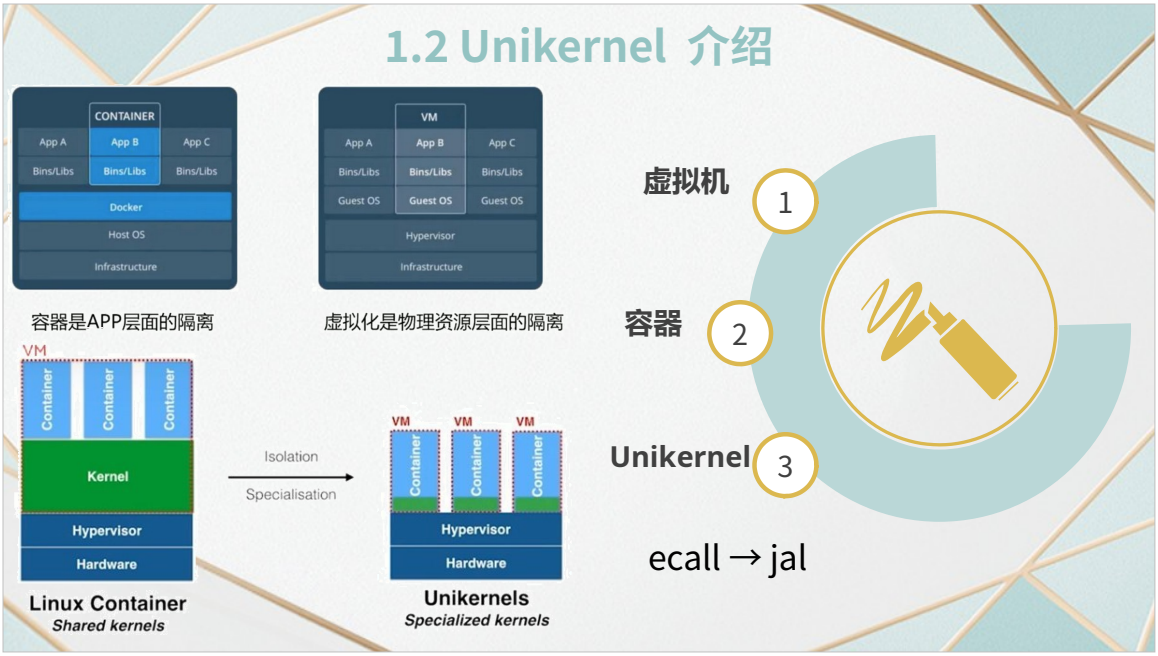
Unikraft 是一种快速、安全、开源并且高度模块化的 Unikernel 构建系统。

RSIC-V

一个基于精简指令集原则（RISC）的开源指令集架构（ISA)。

Rust

Rust 是一门多范式的、通用的编程语言，旨在提高性能和安全性，尤其是安全并发性。



大背景：云计算
Unikernel 兼顾传统虚拟机的优秀隔离和传统容器的高效、低延迟
Unikernel 本质上是运行在虚拟机上的操作系统，但是放弃了系统内的隔离，让用户程序和系统内核运行在同一个地址空间下，这样就可以用高效的函数调用（ jal ）取代低效的环境调用（ ecall ）

1.2 Unikernel 介绍

unikernel

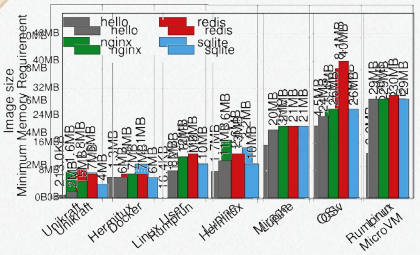


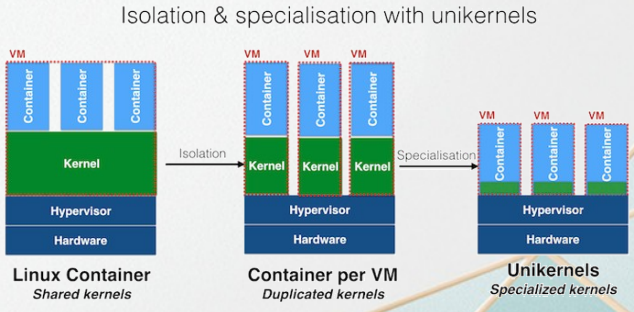
Figure 9.1: Minimum memory needed to run different applications using different OSes, including Unikraft.

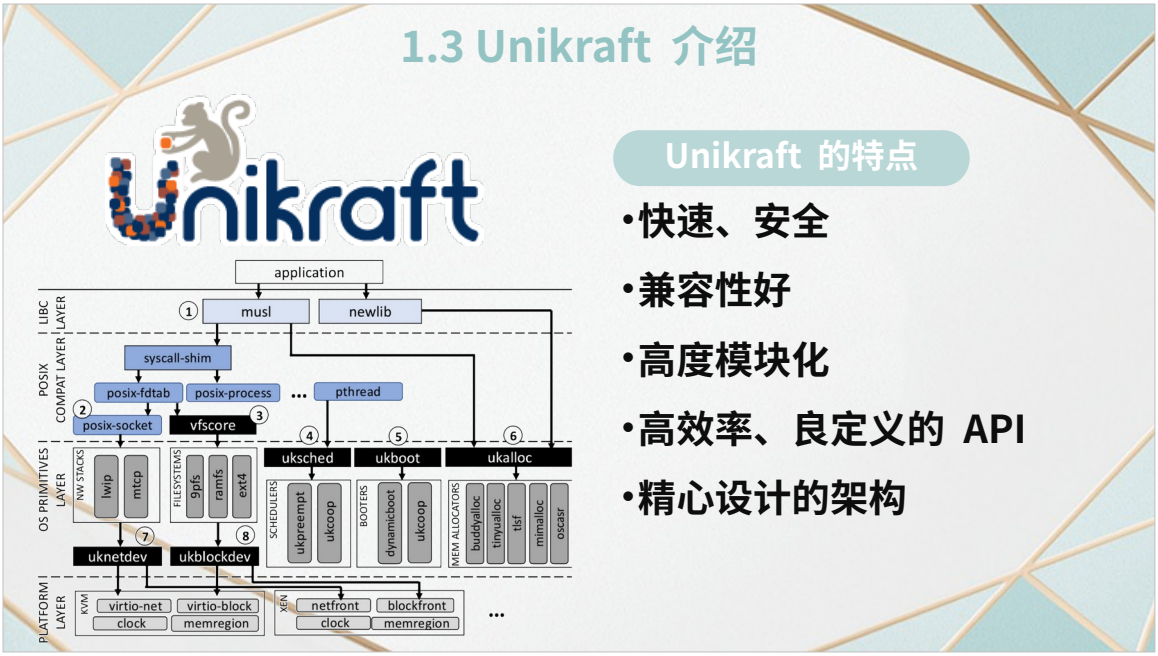
Unikernel 的优点

1. 内存占用量小
与传统的操作系统部署相比， Unikernel 镜像数量级通常更小。

2. 高度优化
Unikernel 编译模型支持跨设备驱动程序和应用程序逻辑进行整个系统优化。

3. 启动速度快
Unikernel 可以极其迅速地启动，启动时间以毫秒为单位。





- **兼容性好**

Unikraft 在保持 unikernel 的极简化、高效的同时，兼容了完整的 POSIX 层，使开发者可以轻松地将现有的为 Linux 编写的代码移植到 unikernel 上。

- **高度模块化**

Unikraft 由若干低耦合的模块组成，内存分配器、调度器、网络栈、引导代码都是独立的微型库。

- **高效率、良定义的 API**

Unikraft 的 API 即为微型库本身，这意味着可以在构建时轻松地添加或移除 APIs，而且提供更多的微型库即可拓展它们功能。

- **精心设计的架构**

在架构方面， Unikraft 融合了宏内核的单地址空间带来的高效性和微内核的模块化带来的可拓展性。 OS 的功能被分割成若干细粒子度的组件，而各个组件之间通过良定义的 APIs 通信。

成本
简洁性
性能
架构和具体实现的分离
提升空间
程序大小
易于编程 / 编译 / 链接

高性能

Rust 速度惊人且内存利用率极高。由于没有运行时和垃圾回收，它能够胜任对性能要求特别高的服务，可以在嵌入式设备上运行，还能轻松和其他语言集成。

可靠性

Rust 丰富的类型系统和所有权模型保证了内存安全和线程安全，让您在编译期就能够消除各种各样的错误。

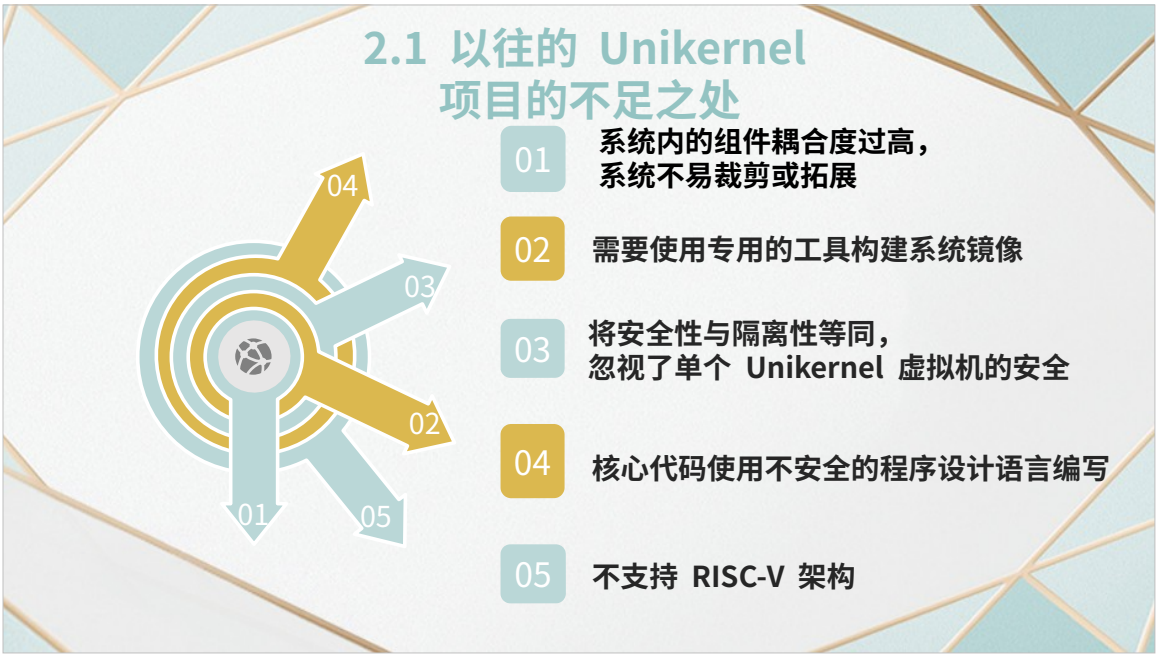
生产力

Rust 拥有出色的文档、友好的编译器和清晰的错误提示信息， 还集成了一流的工具——包管理器和构建工具， 智能地自动补全和类型检验的多编辑器支持， 以及自动格式化代码等等。

1.6 总结

概括来说，我们的项目就是仿照 Unikraft 的架构，用 Rust 语言编写能在 RISC-V 架构 + QEMU 平台上运行的 Unikernel —— Runikraft 。


02
Why
为什么要实现 Runikraft



一个 unikernel 的不足往往是另外一些 unikernels 的优势，所以我们的工作就是借鉴一些 unikernel 的优点，避免另一些 unikernel 的不足。

系统内的组件耦合度过高，系统不易裁剪或拓展

在拓展方面做得比较好的 unikernels 有 MirageOS、Rumprun 和 Unikraft。

需要使用专用的工具构建系统镜像

对于只有几个源文件的小型项目，使用专用的工具并不是什么大问题，但是，对于由成千上万个源文件组成的大型项目，更改构建环境本身就是一项浩大的工程。

将安全性与隔离性等同，忽视了单个 Unikernel 虚拟机的安全

目前，在文档中明确提到 安全措施的 unikernels 有 RustyHermit、Nanos 和 Unikraft。

核心代码使用不安全的程序设计语言编写

使用安全的编程语言写的 unikernels 只有 MirageOS 和 RustyHermit。用不安全的程序设计语言难以避免实现时引入的安全漏洞。

不支持 RISC-V 架构

目前只有 Nanos 支持 RISC-V 架构。

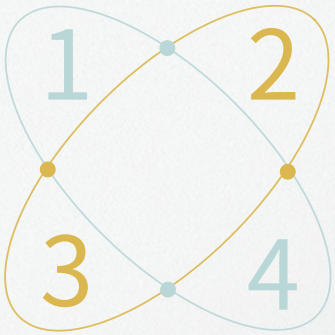
主要是 4、5。1 通过模仿 unikraft 自然实现。

如果只运行用户使用 Rust，那么 3 可以基本解决，但在支持不安全的语言编写的用户程序时需要考虑 3。

3 具体要考虑：内存分配器的实现需要考虑堆加固，平台层需要提供密码学安全的随机数发生器

2.2 Runikraft 的亮点

1. 用安全的 Rust 语言编写
2. 支持正在迅速发展的 RISC-V 指令集架构
3. 显式实现的安全特性
4. 模块化设计，在保持 unikernel 的高效的同时降低维护难度



2.3 前瞻性 / 重要性分析

- **使用先进的工具构建**
- **模块化设计**

目前的大多数 unikernel 强调“uni-”。

在 Runikraft 中，只有极少数平台层的代码被放到了系统的核心组件中，而调度器、分配器等组件一律是 micro-libraries。

micro-libraries 遵循一套明确定义的 APIs，同一个系统模块可以有多种实现，用户可以轻松为自己的需求选择合适的系统组件的实现。

使用先进的工具构建

Rust 和 RISC-V 都是新兴的事物，它们都是在吸取旧事物的教训的基础上诞生的。而且，实践表明，两者都正在进行蓬勃的发展，并正在分别逐步取代旧事物。因此，用 Rust 在 RISC-V 上开发 unikernel 顺应了历史的趋势。

模块化设计

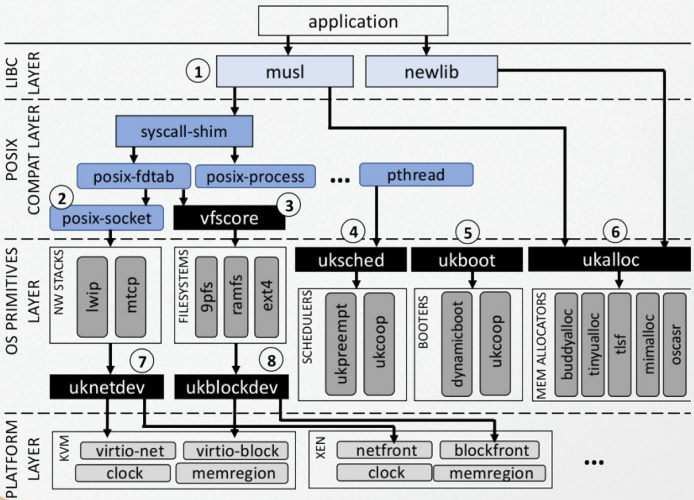
目前的大多数 unikernel 强调“uni-”，它们的设计者认为这样有利于提高效率，所以系统被设计成了一个整体，这个整体向用户提供能够调用的函数。在 Runikraft 中，只有极少数平台层的代码被放到了系统的核心组件中，而调度器、分配器等组件一律是 micro-libraries。这些 micro-libraries 遵循一套明确定义的 APIs，同一个系统模块可以有多种实现，用户可以轻松为自己的需求选择合适的系统组件的实现。从 Unikraft 给出的基准测试数据看，这种模块划分不会降低系统的效率。

03

How

如何实现 Runikraft

3.1 Unikraft 的架构



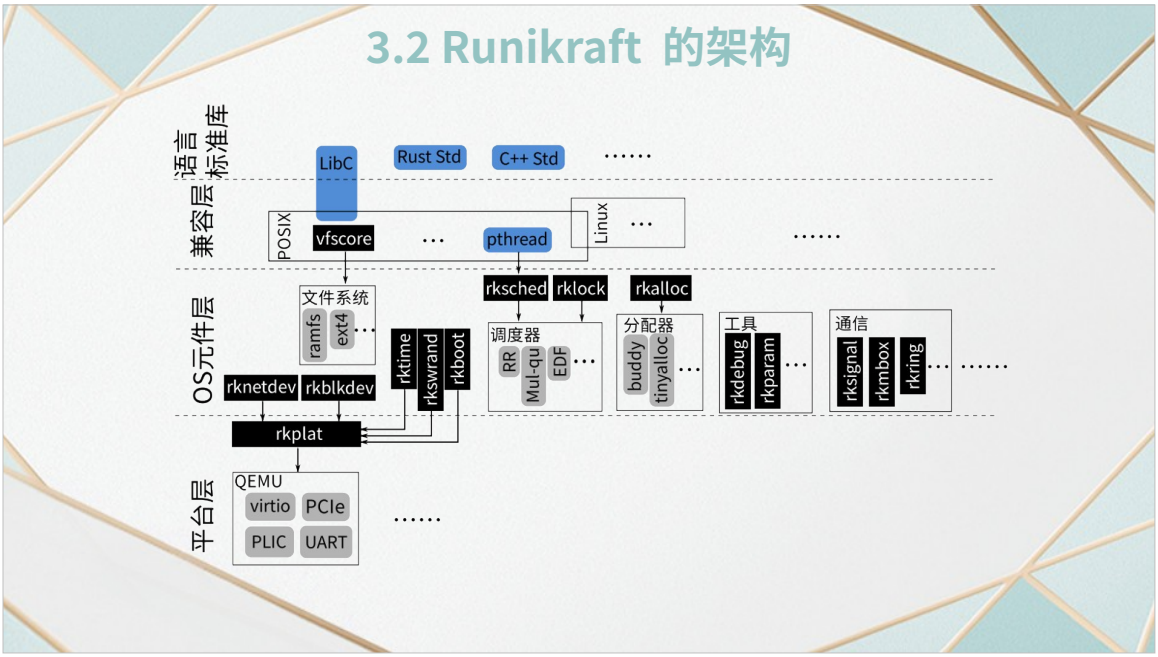
Unikraft 的代码组织方式充分体现了模块化设计，核心代码被放在了 arch/ 、 include/ 和 plat/ 三个目录下，而系统的几乎所有功能都由为了 lib/ 目录下的 micro-libraries 提供。

分配器

分配器的 API 位于 ukalloc, 可选的实现有 ukallocbbuddy 、 ukallocpool 、 ukallocregion 。

调度器

ukschedcoop 是 uksched 的实现之一 , 它实现了最基本非抢占式的时间片轮转线程调度器。它使用双链尾队列 (tail queue) 维护所有线程 (thread_list) 和睡眠状态 (就绪或等待) 的线程 (sleeping_threads) 。每一轮调度 , ukschedcoop 从睡眠线程队列中找出一个处在就绪态的线程 , 并唤醒 (uk_thread_wake) 这个线程。如果没有任何能执行的线程 , 就挂起 CPU(ukplat_lcpu_halt_to) , 直到某个线程能执行。 ukschedcoop 并不需要为每一个事件维护等待队列 , 因为即使线程等待的事件发生了 , 由于当前线程的执行不能被抢占 , 等待的线程也不能被立即唤醒。



Runikraft 的架构与 Unikraft 几乎相同。平台层将不同的平台封装成通用的 rkplat API, 它提供与平台 / 架构密切相关的功能 , 比如外设驱动、外中断处理、内存分页、定时器、原子操作、内存屏障 , 我们只计划支持 RISC-V+QEMU virt 一种平台。如果时间不足 , 我们可以将现成的 OpenSBI 封装成 rkplat API 。

rknetdev 、 rkblkdev 、 rktime 、 rkswrand 、 rkboot 五个 APIs 的功能与平台密切相关 , 但是为了降低开发和维护难度 , 它们并没有直接实现 , 而是在 rkplat 提供的初级抽象的基础上实现。 rknetdev 和 rbblkdev 对应图 7 的⑦和⑧ , 它们分别提供网络设备和块设备的支持。 rktime 提供获取和修改系统时间的 API,rkswrand 提供密码学安全的随机数 ,rkboot 负责完成 OS 元件的初始化并将控制权转交给用户的代码。

rkalloc 是分配器 API, 它的后端可以是 buddy 、 tinyalloc 、 tlsf 、 mimalloc 等分配器。

rksched 和 rklock 是两个与调度器有关的 APIs, 前者负责创建、调度、撤销线程 , 后者负责线程间的同步和互斥。 Runikraft 支持多种调取器 , 比如 , 图 8 中的 RR 是时间片轮转调度器、 Mul-qu 是多重队列调取器、 EDF 是截止日期有限调度器。我们计划先实现 RR 调度器。

Runikraft 选择性地提供线程通信模块 , 如 UNIX 风格的信号 (rksignal) , 信箱 (rkmbox) , 无锁的环形缓冲队列 (rkring) 。

我们目前没有计划区分线程和进程 ,Runikraft 的线程同时具有传统的 OSes 的线程和进程的特性 : 线程之间没有隔离措施 , 但是线程之间又可以使用进程通信的方法更安全地同步。

Runikraft 还提供了一些工具模块 , 比如调试工具 rkdebug 、 命令行参数分析工具 rkparam 。

在 Unikraft 中 , 文件系统 API vfscore 是 POSIX 兼容层的一部分 , 我们沿用了它的设计 , 将 vfscore 放在了兼容层。 Runikraft 计划支持 RAM 上的临时文件系统和 Plan 9 OS 的 9fps 。

位于最顶层的是语言标准库 , 这一层可以帮助 Runikraft 支持多种语言。虽然语言标准库层被画在了兼容层之上 , 但它其实是直接用 OS 元件层的 APIs 实现的 , 这能避免分层系统的低效。我们只打算实现 Rust 标准库和 C 标准库的部分内容。

我们制定了详细的转换表，有兴趣的同学可以自行查阅我们的可行性报告。转换原则是指针尽可能转换成引用，遇到生命周期或所有权问题时才使用裸指针，尽可能使用 Rust core 中内置的类型， const char * 尽量转换成 str 。

所谓“面向对象式函数”就是第一个参数是与库同名的结构体的指针的函数。

3.4 开发路线

- 第 10~14 周实现 OS 元件层
- 第 15~16 周实现兼容层

<table>
<tr><td>第 10 周</td><td rowspan="2">rkplat</td><td>rkalloc</td><td rowspan="2">rkring</td><td rowspan="2">rkblkdev</td><td rowspan="2">rksched
无抢占</td></tr>
<tr><td>第 11 周</td><td rowspan="3">rknetdev</td></tr>
<tr><td>第 12 周</td><td>rktime</td><td rowspan="2">rkmbox</td><td>rklock</td></tr>
<tr><td>第 13 周</td><td>rkswrand</td><td rowspan="2">9pfs</td><td rowspan="2">rksched
可抢占</td></tr>
<tr><td>第 14 周</td><td rowspan="3">rkplat 不
依赖 SBI</td><td>rkparam</td><td>rksignal</td></tr>
<tr><td>第 15 周</td><td rowspan="2">RustStd</td><td rowspan="2">LibC</td><td rowspan="2">vfscore</td><td rowspan="2">pthread</td></tr>
<tr><td>第 16 周</td></tr>
</table>

SBI （supervisor binary interface）提供了一些最基本的硬件驱动，前期可以使用

内核的调度器、环形缓冲区等都需要使用链表， unikraft 的 compat_list.h 中定义了单链表、双链表、单向尾链表、双向尾链表 4 种链表，而且这个头文件还用宏实现了泛型；用 safe Rust 实现双向链表很难避免所有权问题，所以我们会用 unsafe Rust 实现链表。由于 unsafe 只出现在链表结构体的实现内部，我们的代码整体上是符合 unsafe 尽量少的原则的。

Unikraft 的开发者表示 uknetdev 是实现难度最大的部分，他们精心设计了 uknetdev 模块，并最终使 Unikraft 的网络吞吐量高于其他 unikernels 。所以把 uknetdev 改写成 rknetdev 后，必须保持它原本的高效。缺乏网络知识也加大了实现之一模块的跳转。

rCore-tutorial-book 讲了用 Rust 在 RISC-V 上实现操作系统，很多不明白的地方都可以查这本书

3.5 可能遇到的困难

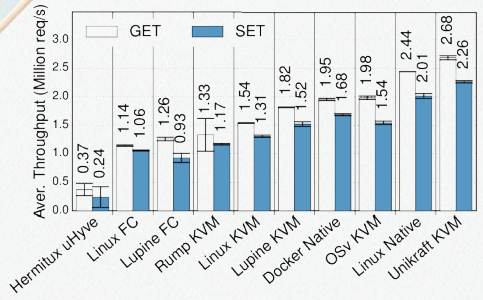


Figure 12. Redis perf (30 conns, 100k reqs, pipelining 16) with QEMU/KVM and Firecracker (FC).

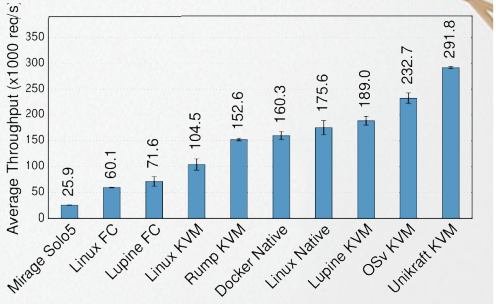


Figure 13. NGINX (and Mirage HTTP-reply) performance with `wrk` (1 minute, 14 threads, 30 conns, static 612B page).


04
Q&A
问答


Thanks!