第一次例会汇报

time： 3.26 week3

author：李思逸

第零部分：调研要求

内容要求:

1. 是什么(介绍给组里人听)
2. 优缺点
3. 目前仍待解决、改进的问题
4. 用到什么方面知识，可行性，创新性
5. 范围不局限于此，可自行拓展

其他要求:

1. 以调研报告的形式做调研笔记，在自己理解基础上，写出自己的行文通顺、逻辑通顺的内容，总之就是要能讲给大家听，一目了然
2. 可以着手先把git和github一些操作弄熟练
3. 无论做什么项目，每天大家都应该分配一些时间给大作业，小步快跑，一起交流进步。

第一部分：个人能力介绍

- 编程语言
  有一定算法实现能力
  c/c++：熟练。有过QT开发经历
  python：简单的基础
- 其他比较菜
  但是有责任心，乐于交流
  能肝，愿意学

第二部分：本周调研工作汇报

以下根据
https://github.com/OSH-2020/x-chital/blob/master/discuss/1/%E6%A1%86%E6%9E%B6.pdf
提供的框架改编，为可能实现的方向

重点调研了虚拟化技术，一些想法持续更新

虚拟化技术

容器技术

2020 x-chital

- 用rust 改写gVisor（用go实现）
  - gVisor 出于安全容器考虑 系统调用劫持 实现对容器技术的提升
  - 该小组设计了 rVisor 的kernel 和 整个系统的组成
  - 使用了清华的zCore作为与rVisor交互的内核， 修改linux 一小部分内核
  - 用rust修改上层软件接口
- 有关该项目容器技术 docker rVisor 等细节 见 https://github.com/OSH-2020/x-chital/blob/master/docs/conclusion/conclusion.md
- 有关rCore zCore fuchsia zircon的技术细节

  zCore是rCore的升级，对zircon用rust进行改写，只有1万行左右

  zircon是谷歌推出的开源项目fuchsia系统的内核

  https://zhuanlan.zhihu.com/p/137733625

对上述项目的一些看法：

轻量级虚拟化:

- 一种可能的方案：继续优化rVisor或者改进gVisor

- 另一种方案 kata+firecracker

Kata定位仍然是容器，但是它已经向虚拟机卖出了很重要的一步。就是Kata的容器提供的是容器的管理方式，是一种看起来像容器的虚拟机。也就是说Kata看到了容器的不足，但是同时也看到了Docker的优点，他综合起来，用虚拟机来弥补容器隔离上的不足，所以Kata其实是一个像容器的虚拟机方案。

而Kata不能解决的是虚拟机的问题，就是重量级。这个问题AWS最近给出了答案，Firecracker作为AWS开源的轻量级云内核完美的匹配了Kata的痛点，Kata+Firecracker相当于给出了对Docker的一个相对完美的升级方案。

轻量级的虚拟化本身就是一个解决虚拟化痛点的一个很大的方向。不但是AWS，谷歌给出了一个完全不同的方向。Firecracker的思路是既然虚拟机重量级，那么我就让虚拟机轻量，但是还是用虚拟机的思想来做到更好的隔离性，但是谷歌的gVisor不这么认为。gVisor认为既然隔离性不够，那我就从隔离性本身下手，让隔离性更好。所以谷歌也是在容器的后端下功夫，但是下功夫的方式是将所有的系统调用截断，在gVisor中用用户程序来实现系统调用的API，从而将内核的逻辑耦合变成了一个一个的独立的进程。相当于在用户空间实现了内核的功能，给Docker使用的时候，Docker看到的也是一个内核，但是实际上是一个gVisor自己实现的模拟内核。区别于真的把内核跑在用户空间的UML项目，gVisor极其轻量，隔离性却也达到了操作系统能带来的隔离程度。而且其实现强度极其让人发指，就连Linux的网络协议栈都在用户空间实现一遍。几乎实现了所有的Linux系统调用。

而服务端，两条路到底会走哪一条？看起来势均力敌。AWS的这条路船上的人多，最后的形态会是K8S+Kata+Firecracker，一个完整的轻量级容器VM方案，一套方案既可以做容器，又可以做VPS是这个方案在产品上的优势。另外一个是干净利落的解决Docker的痛点问题的gVisor方案，又是谷歌的杰作。在无服务化计算的这条路，目前双方话语权相当，胜负难定。

详见：https://zhuanlan.zhihu.com/p/55603422

以下选自：https://aws.amazon.com/cn/blogs/china/firecracker-open-source-secure-fast-microvm-serverless/

2017 年秋，我们（**AWS**）决定以 Rust 语言来编写 Firecracker，这是一种非常先进的编程语言，可保证线程和内存安全，防止缓存溢出以及可能导致安全性漏洞的许多其他类型的内存安全问题。请访问 Firecracker 设计以了解有关 Firecracker VMM 功能和架构的更多详细信息。

关于虚拟化、KVM介绍的文章：

https://developer.aliyun.com/article/724394

节选：

KVM从诞生开始就定位于基于硬件虚拟化支持的全虚拟化实现。它以内核模块的形式加载之后，就将Linux内核变成了一个Hypervisor，但硬件管理等还是通过Linux kernel来完成的，所以它是一个典型的Type 2 Hypervisor，如图1-7所示。

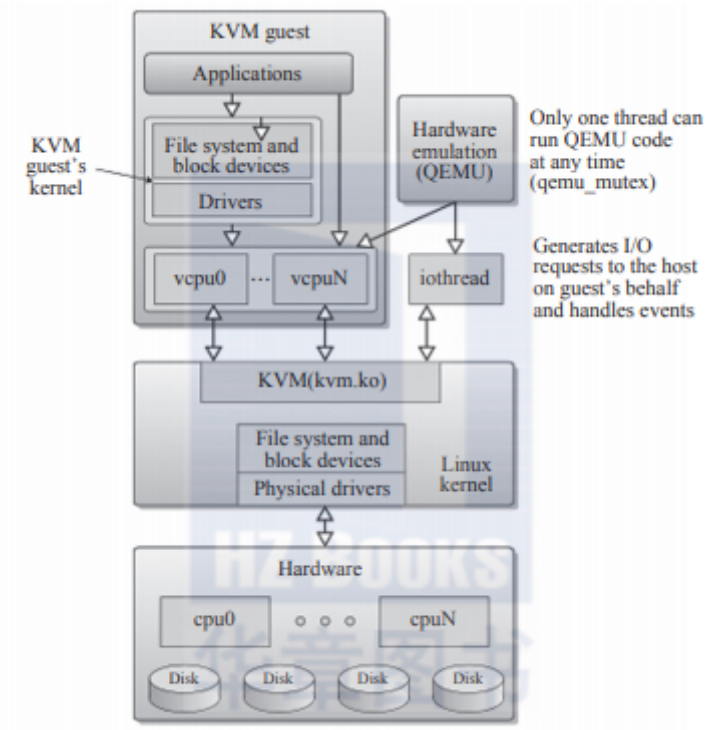


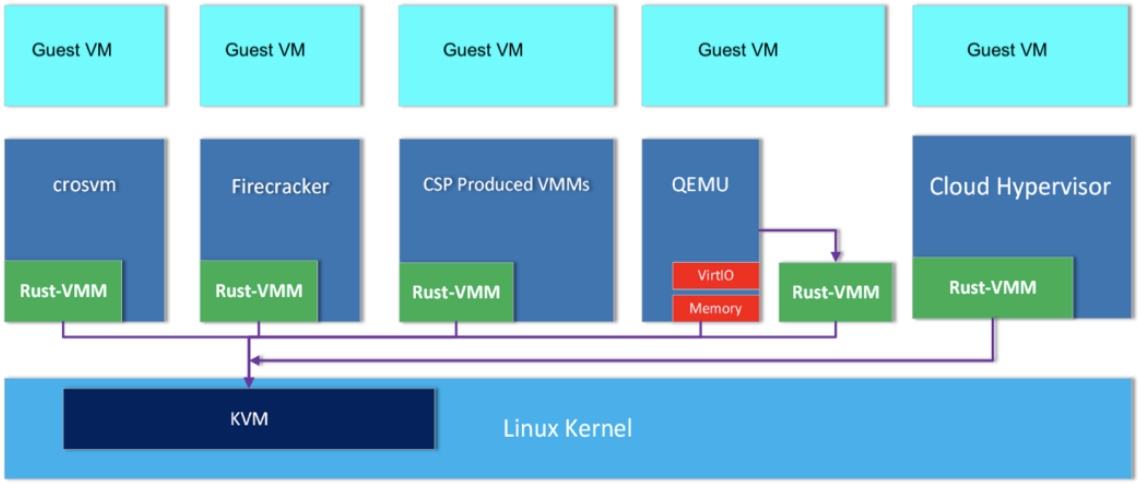
图 1-7　KVM 功能框架

以及有关 虚拟化 VMM KVM docker 的简介：

https://blog.csdn.net/baiyan83/article/details/110182411

- 新的观点 rust-vmm

简单介绍一下Rust-Vmm的一个历史，它是由谷歌首先实现的，谷歌首先实现一个Rust based的轻量级的VMM，它叫做crosVM，大家也可以从链接里面看到，它是一个为chrome浏览器做的一个微内核。然后AWS，亚马逊基于谷歌开源出来的crosVM，实现了自己的基于rust的VMM叫Firecracker。两个项目的开发人员会发现做这两个项目的时候，会有很多重复的重叠的通用的代码，很自然的把可以开源的、通用的部分结合到一块，就有了Rust-Vmm的项目。



我们是否可以基于rust-vmm 来做一些项目?

继续介绍firecracker：https://aws.amazon.com/cn/blogs/china/deep-analysis-aws-firecracker-principle-virtualization-container-runtime-technology/

从主要两个维度降低风险：

1. 抛弃 QEMU 使用的 C 语言，选择内存安全的 Rust 作为开发语言
2. 基于 crosvm 使用极简设备模型，模拟尽可能少的必要设备，减小暴露的攻击面

在 Rust 语言的定义中，凡是可能会导致程序内存使用出错的特性，都被认为是不安全的（Unsafe），反之是安全的（Safe）。除非通过”Unsafe “关键字显示使用， 如果您的代码有某种方式可能会导致内存不安全（例如缓冲区溢出），则会出现编译错误。生命周期和所有权也可以确保安全的并发性。 例如，编译器强制在任何时候，给定的内存区域只能由一个线程写，也可以由任意数量的线程读取。而与 Rust 并列的底层开发语言 C 和 C++，则是属于不安全的语言。

再次是高性能和低开销：得益于极简的设备模型， Firecracker 取消了 SeaBIOS （开源的 X86 BIOS），移除了 PCI 总线，取消了 VGA 显示等等硬件模拟，严格的说它甚至不是一台完整的虚拟计算机。而 Firecracker 运行的 GuestOS 使用的也是 AWS 定制过的精简 Linux 内核，同样裁剪掉了对应的设备驱动程序、子系统等等。因此叫它 MicroVM，其启动步骤和加载项要远远少于传统虚拟机。因此 Firecracker 目前已经能提供小于125ms 的 MircroVM 启动速度，每秒150台的启动能力，小于5MiB 的内存开销，并发运行4000台的极限承载容量（AWS i3.metal EC2 作为宿主机），以及热升级能力等。这些都是传统虚拟机所遥不可及，但现代化弹性工作负载又有强烈需求的性能指标。

为了解决容器安全隔离性问题，开源社区也涌现了大量同类项目，而 Firecracker 是跟它们相比有哪些不同，又如何脱颖而出的呢？在笔者看来这些其它方案大致分为两类：

一类是基于在已有的成熟 VMM，进行编译选项裁剪和 GeustOS 精简，同样使用虚拟化技术解决问题。这类方案包括基于 QEMU-KVM 的 Hyper.sh RunV，Intel Clear Container，Pouch Container 等，基于 VMware ESXi 的 vSphere Integrated Containers，以及 Microsoft 的 Hyper-V Container。这类方案的主要不足源于这些 Hypervisor 过去用在传统虚拟机，并不是面向无服务器计算这类的现代化工作负载而设计开发，因此在安全、性能、开销方面可能会存在先天不足。

而另一类则完全不使用虚拟机，以 Google 的 gVisor 以及一众 Unikernel 技术为代表gVisor 引入了进程虚拟化的思想，去掉了设备模型，将虚拟化的边界移到了系统调用层面，从而减少传统设备虚拟化的开销。**而代价却是有大量的 Linux 系统调用需要沙箱处理，暴露了更多的攻击面（目前已实现了数百个系统调用）。**而 Unikernel 则源于90年代的操作系统微内核理念，随着近年来容器技术的走红，这项技术也越来越被人关注（例如 Nabla container）。其核心思想是将 Linux 的宏内核拆分成为多个库，只打包应用依赖的库到微内核镜像中。每个应用拥有独立的微内核，从而提升性能，降低开销，减少攻击

面。不过这些 Unikernel 项目大部分处于早期阶段，作为容器技术的竞争对手，其部署、管理以及与 OCI 的兼容性都会是一大考验。

- unikernel
- 有关unikernel的简介：

  https://www.jianshu.com/p/aa42225da211

  https://zhuanlan.zhihu.com/p/29053035
- 有关zsr推荐的项目includeOS在RISC-V架构上的实现，一些想法
-
  - unikernel本来是为了提供更轻量级的服务，如果在risc-v模拟器上实现，没有实际的意义。
  - risc-v的模拟器到底性能可以到达什么程度，需要的软硬件支持?
  - 网上有一些基于RISC-V芯片的开发板，如荔枝丹，荔枝糖之类，价格在200-300 之间，可以尝试
  - 或者把其他OS移植到树莓派的ARM架构上

文件系统

暂未仔细调研

WebAssembly 虚拟机

不是很懂这块的细节

其他语言->WA的运行机制

从看到的资料来说, 我对webassembly的了解主要是"它统一了JS, WEB的生态", "它可以(用LLVM等)把C等语言编译成统一的机器码". 总之这是一种工具, 但是现在对它的组件支持不完善(比如调试不是很方便), 所以我觉得选题可以从这些方面入手.

实现一个WebAssembly虚拟机

现阶段，WebAssembly 主要还是以Web应用为主，执行的容器大多基于主流的浏览器，并且通过javascript与外部通信，但是它的基于自定义内存和沙盒的特性，也使得WebAssembly 可以很好的适用于一些轻量级的场景，如作为执行区块链智能合约的虚拟机。

WebAssembly 是基于栈式的虚拟机，指令的执行都是在栈内完成的：

webAssembly 指令集参考：webAssembly bianry code

WebAssembly 只支持4种基本类型：

- int32
- int64
- float32
- float64

所以函数的参数和返回值也只能是这四种类型，并且每个函数只能有一个返回值。

如果想要使用复杂的类型，比如 string，就需要额外对内存进行操作。 这里是关于WASM更完整的介绍

原文链接：

https://github.com/OSH-2020/x-chital/blob/master/discuss/3/WA%E5%92%8C%E5%88%86%E5%B8%83%E5%BC%8F%E7%9A%84%E9%80%89%E9%A2%98%E6%83%B3%E6%B3%95(1).md

更完整的wasm vm的介绍：

https://github.com/ontio/ontology-wasm/blob/master/doc/wasmvm_introduction.md