# KVM

喵喵喵
在真实的世界真实地活着



## KVM

最近实习时，老板让我研究一下安全容器：kata-containers。这个项目本质上就是把虚拟机和容器生态相整合。所以需要大致了解一下虚拟化是怎么跑起来的，我就以KVM为例，大致学习了一下。

## 虚拟化基础知识

啥几种类型的VMM之类的东西就不用说了，OS课本上多的是老掉牙的图。主要要说的是以下几个东西。

### 半虚拟化与全虚拟化

para-virtualization：半虚拟化就是guest OS要进行修改，才能跑上去。例如Xen（详见论文）就是把guest OS给改改，改guest OS，让它不做特权操作，只通过hyperCall来和host OS通信。这样子可以跑在不支持虚拟化扩展的老x86硬件上，同时也没有复杂的设备模拟，性能也高。 full-virtualization：全虚拟化就是guest OS不经过修改，就可以直接跑上去。典型的就是KVM还有VMware。全虚拟化要想性能好，一般需要处理器的虚拟化支持。

Xen论文：Xen and the Art of Virtualization

KVM论文：kvm: the Linux Virtual Machine Monitor

### 虚拟化扩展

x86的虚拟化扩展主要包括了如下三方面：

- A new guest operating mode
- Hardware state switch
- Exit reason reporting

我们可以看看这三方面各自解决了什么问题：

第一个是引入了一个guest OS模式。传统的x86不太好虚拟化，主要就是因为它的指令行为设计得不好，会有silent fail以及side effect之类的（不知道具体是不是这个说法，反正x86指令集就是虚拟化不太友好），不太好拿trap and emulate的方式来做虚拟化。现在引入了这个mode后，在这个模式下，指令的行为可以被trap and emulate。值得注意的是guest mode与x86传统的ring权限结构是正交的。guest OS运行时是在guest mode，ring 0，guest OS的app是运行在ring 3。只要它们做了特定的非法动作或者机器上来了某些中断，就会触发VMExit。host OS、host OS上的程序，VMM运行在root mode，host OS是ring 0，kvm.ko也是ring 0，qumu是ring 3。在root mode，处理器的行为就和传统的处理器一样。在root mode下，通过VMEntry之类的指令可以再进入guest mode。

下一个就是hardware state switch，guest OS的hardware context可以被保存，修改再恢复。这个可以对比OS与process的关系，OS要能保存、修改再恢复process的hardware context。VMM就需要有能力保存、修改、再回复OS的hardware context。VMM的这个hardware context在Inte上面叫vmstruct。

最后就是exit reason reporting。这个就类似process trap时的error code。有了这个，VMM就可以做针对性的处理。

## KVM

KVM是Linux是给linux kernel增加了一个子系统，添加了这个子系统后，linux kernel就可以变成一个hypervisor了。同时，kvm还用到了qemu来进行设备模拟，把它放在用户态。

由于虚拟化本质上还是靠trap and emulate来实现。不同的是，如果硬件上支持更多的虚拟化扩展，可以把guest OS的更多代码直接在硬件上运行，可以尽量少地触发trap，提升性能。

### KVM的运行流程

大致流程是：KVM guest OS运行在VMX non root mode下，在这个下面，它可以自由地干自己想干的事情，例如进程调度，内存分配，巴拉巴拉。然后等到实在干不了的时候，例如IO，fault之类的，就会VMExit，VMExit后，可以由KVM的内核部分来处理CPU，memory相关的，也可以有host kernel再把任务丢到user space，由qemu进行设备模拟。

VMM本质上就是许多个事件处理循环，这里本质上就是在讲和VMM，host OS，guest OS相关的一些事件处理循环。

KVM在linux传统的kernel mode和app mode之外增加
看下图。 这张图主要讲的是VMM，host OS，guest O

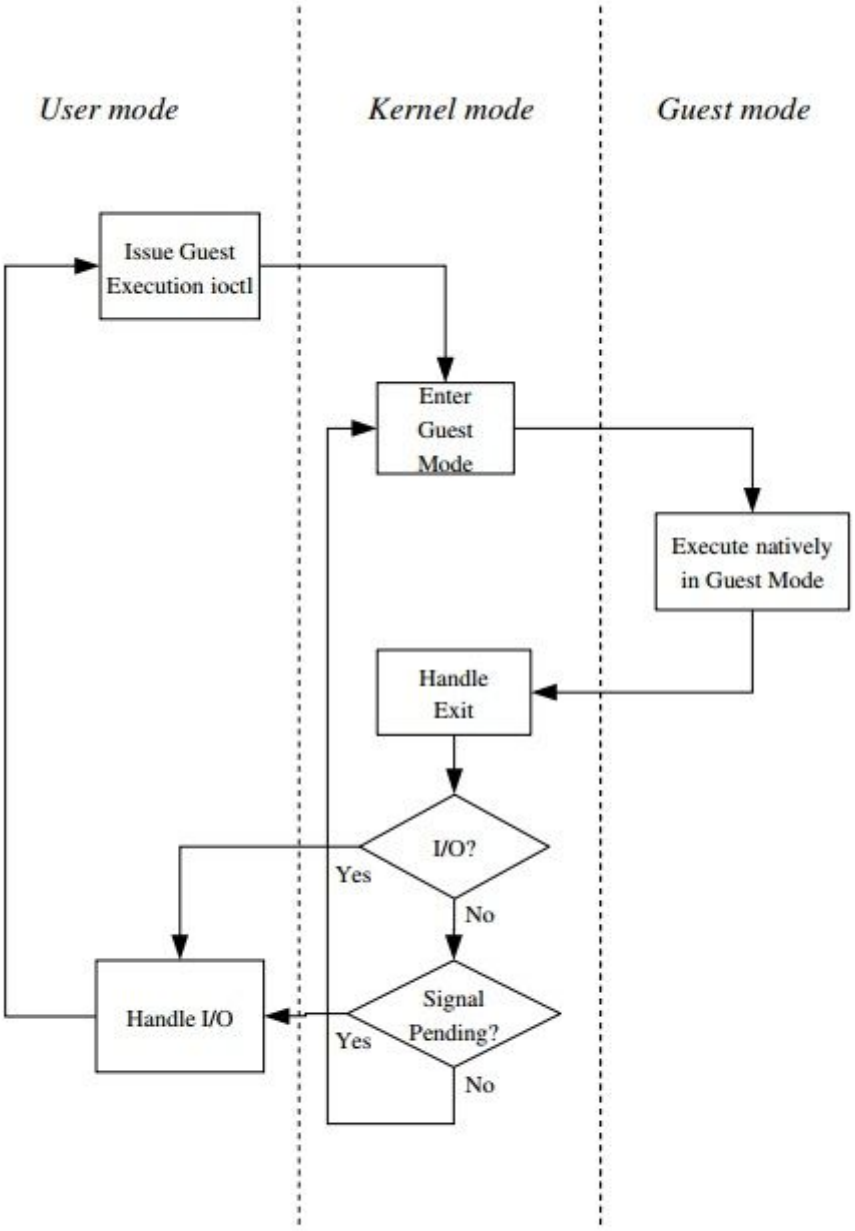


Figure 2: Guest Execution Loop



1. 首先kvm.user通过ioctl系统调用，让host kernel调用VMEntry，进入guest mode。
2. guest OS就开始执行，它在里面可以运行自己的guest app，进行自己的系统调用，就和普通的操作系统一样。
3. 假如遇到了一些问题（例如自己trap或者是外面有信号或者中断），就会触发VMExit，退出guest OS，host OS开始接管。
4. 然后假如是因为guest OS Io相关的中断或者trap的话，由于IO是由qemu在用户态进行设备模拟的，所以host kernel将控制权转交qemu
5. qemu处理好了后，就可以再进入guest OS运行了。
6. 上面判断的IO的另一条路径，假如不是IO的话，就可

然后这篇文章的剩下部分基本都是在介绍CPU，MMU以及IO这几大子系统是如何实现虚拟化的。总的来说，KVM的核心设计思路就是把不得不由内核干的事情，通过内核的kvm子系统来完成，然后其他部分，包括虚拟化CPU，IO，MMU全部由user process来干。

## KVM的实现

那么KVM具体是如何实现的呢？一个guest OS在host OS看来到底是什么？进程调度，切换，内存分配，IO都是怎么搞的呢？ 要知道这些，就要知道KVM的具体实现了。

## KVM CPU

一个KVM虚拟机就是一个运行qemu-kvm的进程，一个vcpu就是进程里面的一个线程。这个虚拟机进程里面除了有N个VCPU线程，还会有其他的IO、管理线程等等。

KVM guest OS根本不用管它的那个vcpu在host os里面是不是被抢占了，是不是跑得快，跑得慢了，它只负责把它的任务调度到VCPU上面运行，然后这些VCPU线程就像普通的Linux线程一样，被host调度。所以这个里面其实有两级的调度。

host OS根本不管你这个线程上面运行的是guest OS kernel还是guest OS的app还是VMM的部分代码，host OS统统不关心，它只管调度运行它就是了。

问题1：host os怎么知道guest OS的vcpu上面有任务呢？ host OS不用知道guest OS的vcpu上面是不是有任务，只要guest OS没有halt它的vcpu，那么这个vcpu的thread就是runnable的，那么host os就会调度运行这个线程。

问题2：guest OS里面的idle进程始终在运行，怎么避免它们消耗过多的host CPU呢？ idle在实现时一般是通过特殊指令halt CPU，假如guest OS进入了idle，halt了VCPU，那么对应的，它的VCPU线程就不是runnable了，那么host OS就不会调度这个VCPU线程，那么就不会消耗host cpu了。

## KVM memory

guest OS的内存其实是来自qemu-kvm的虚拟内存空间。所以这里又是一个两级的映射，KVM最一开始是用shadow page的方式来搞，性能不太好。后来硬件支持nested paging后，就好搞多了。

KVM CPU以及memory管理一般是由内核KVM模块完成的。

## KVM IO

态做好IO模拟后，再切回到guest OS里面去，让它接着运行。

使用qemu进行设备的完整模拟，性能非常差，现在在KVM中，一般是使用virtio这类半虚拟化设备。

发布于 2020-02-04

KVM（Kernel-based Virtual Machine）　虚拟化

**推荐阅读**

**深入浅出KVM（一） | 简介&安装**

Avate...　发表于阳与月的轮...

**virtio 虚拟化系列之一：virtio 论文开始**

smart...　发表于

**1 条评论**　切换为时间排序

写下你的评论...

知乎用户　1 个月前

看不太懂，想要入门深入学习下，需要看些什么资料？有没有专门的书籍介绍这些内容？

赞