CPU 虚拟化

> 本文首发于我的公众号 **Linux云计算网络（id: cloud_dev）** ，专注于干货分享，号内有 **10T** 书籍和视频资源，后台回复取，欢迎大家关注，二维码文末可以扫。

CONTENTS

前面 虚拟化技术总览 中从虚拟平台 VMM 的角度，将虚拟化分为 Hypervisor 模型和宿主模型，如果根据虚拟的对象（资源拟化又可以分为计算虚拟化、存储虚拟化和网络虚拟化，再细一些，又有中断虚拟化，内存虚拟化，字符/块设备虚拟化，等。

我会将此作为一个系列来写，本文先看 CPU 虚拟化。在这之前，我们先来笼统看下虚拟化的本质是什么，它到底是如何做到将 Host 的硬件资源虚拟化给 Guest 用，我这里用两个词来定义， **intercept** 和 **virtualize** ，中文翻译成 **截获** 和 **模拟** 比较恰当一点，这两个词基本上是虚拟化的终极定义了，带着这两个词去看每一种虚拟化类型，会发现很容易理解和记忆。

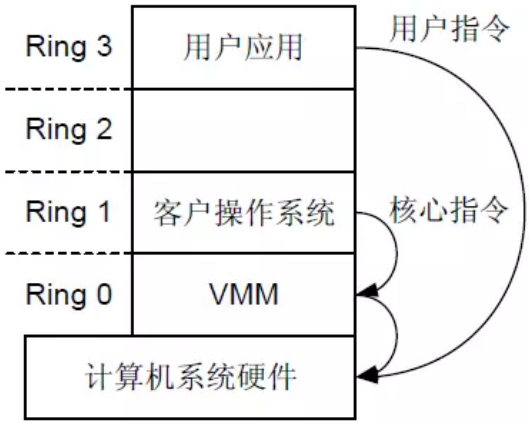
CPU 软件虚拟化

基于软件的 CPU 虚拟化，故名思议，就是通过软件的形式来模拟每一条指令。通过前面的文章我们知道常用的软件虚拟化技术有两种：优先级压缩和二进制代码翻译。这两种是通用技术，可以用在所有虚拟化类型中。我们就结合 intercept 和 virtualize 来看看 CPU 软件虚拟化是怎么做的。

首先，一些必须的硬件知识要知道，X86 体系架构为了让上层的软件（操作系统、应用程序）能够访问硬件，提供了四个 CPU 特权级别，Ring 0 是最高级别，Ring 1 次之，Ring 2 更次之，Ring 3 是最低级别。

一般，操作系统由于要直接访问硬件和内存，因此它的代码需要运行在最高级别 Ring 0 上，而应用程序的代码运行在最低级别 Ring 3 上，如果要访问硬件和内存，比如设备访问，写文件等，就要执行相关的系统调用，CPU 的运行级别发生从 Ring 3 到 Ring 0 的切换，当完成之后，再切换回去，我们熟悉的用户态和内核态切换的本质就来自这里。

虚拟化的实现也是基于这个思想，VMM 本质上是个 Host OS，运行在 Ring 0 上，Guest OS 运行在 Ring 1 上，再往上是相应层次的应用程序运行在 Ring 2 和 Ring 3 上。

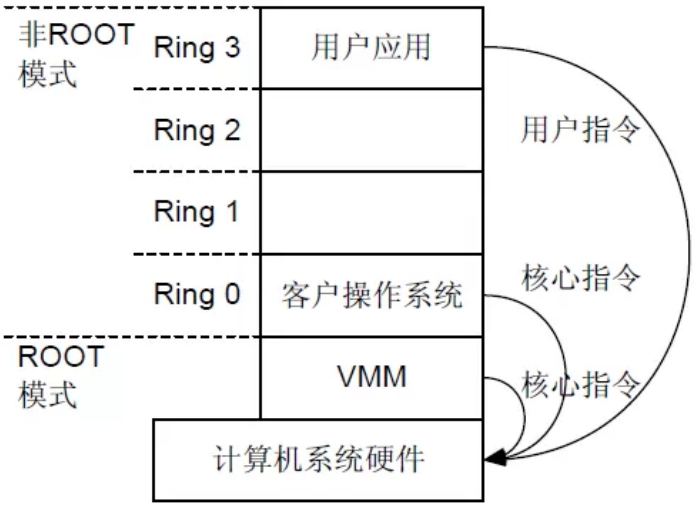


当 Guest OS 或上层应用在执行相关的特权指令时，就会发生越权访问，触发异常，这个时候 VMM 就截获（intercept）这个指令，然后模拟（virtualize）这个指令，返回给 Guest OS，让其以为自己的特权指令可以正常工作，继续运行。整个过程其实就是优先级压缩和二进制代码翻译的体现。

CPU 硬件虚拟化

上面的这种截获再模拟的纯软件的虚拟化方式，势必是性能非常低的。那怎么样提高性能呢，有一种改进的方式是修改 Guest OS 中关于特权指令的相关操作，将其改为一种函数调用的方式，让 VMM 直接执行，而不是截获和模拟，这样就能在一定程度上提高性能。

但这种方式并不通用，要去改 Guest OS 的代码，只能看作是一种定制。为了能够通用，又能够提高性能，就只能从硬件上去做文章了。所以，后来，以 Intel 的 VT-x 和 AMD 的 AMD-V 为主的硬件辅助的 CPU 虚拟化就被提出来（Intel VT 包括 VT-x （支持 CPU 虚拟化）、EPT（支持内存虚拟化）和 VT-d（支持 I/O 虚拟化））。



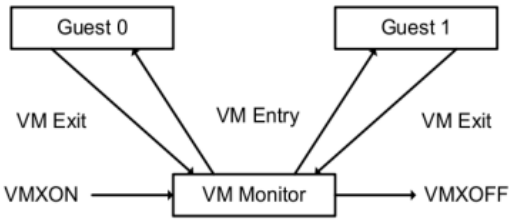
CPU 硬件辅助虚拟化在 Ring 模式的基础上引入了一种新的模式，叫 VMX 模式。它包括根操作模式（VMX Root Operation）和非根操作模式（VMX Non-Root Operation）。

这两种模式都有 Ring 0 - Ring 3 的特权级。所以，在描述某个应用程序时，除了描述其属于哪个特权级，还要指明其处于根模式还是非根模式。

引入这种模式的好处就在于，Guest OS 运行在 Ring 0 上，就意味着它的核心指令可以直接下达到硬件层去执行，而特权指令等敏感指令的执行则是由硬件辅助，直接切换到 VMM 执行，这是自动执行的，应用程序是感知不到的，性能自然就提高了。

这种切换 VT-x 定义了一套机制，称为 VM-entry 和 VM-exit。从非根模式切换到根模式，也就是从 Guest 切换到 Host VMM，称为 VM-exit，反之称为 VM-entry。

- VM-exit： 如果 Guest OS 运行过程中遇到需要 VMM 处理的事件，比如中断或缺页异常，或者主动调用 VMCALL 指令调用 VMM 服务的时候（类似于系统调用），硬件自动挂起 Guest OS，切换到根模式，VMM 开始执行。
- VM-entry： VMM 通过显示调用 VMLAUNCH 或 VMRESUME 指令切换到非根模式，硬件自动加载 Guest OS 的上下文，Guest OS 开始执行。



KVM CPU 虚拟化

KVM 是一种硬件辅助的虚拟化技术，支持 Intel VT-x 和 AMD-v 技术，怎么知道 CPU 是否支持 KVM 虚拟化呢？可以通过如下命令查看：

```
# grep -E '(vmx|svm)' /proc/cpuinfo
```

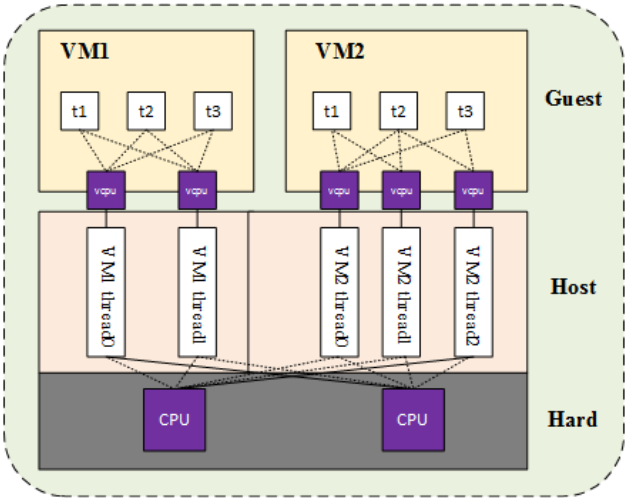
Copy

如果输出是 vmx 或 svm，则表明当前 CPU 支持 KVM，Intel 是 vmx，AMD 是svm。

从本质上看，一个 KVM 虚拟机对应 Host 上的一个 qemu-kvm 进程，它和其他 Linux 进程一样被调度，而 qemu-kvm 进程中的一个线程就对应虚拟机的虚拟 CPU （vCPU），虚拟机中的任务线程就被 vCPU 所调度。

比如下面这个例子，Host 机有两个物理 CPU，上面起了两个虚拟机 VM1 和 VM2，VM1 有两个 vCPU，VM2 有 3 个 vCPU，VM1 和 VM2 分别有 2 个 和 3 个线程在 2 个物理 CPU 上调度。VM1 和 VM2 中又分别有 3 个任务线程在被 vCPU 调度。

所以，这里有两级的 CPU 调度，Guest OS 中的 vCPU 负责一级调度，Host VMM 负责另一级调度，即 vCPU 在物理 CPU 上的调度。



我们也可以看到，vCPU 的个数，可以超过物理 CPU 的个数，这个叫 CPU 「超配」，这正是 CPU 虚拟化的优势所在，这表明了虚拟机能够充分利用 Host 的 CPU 资源，进行相应的业务处理，运维人员也可以据此控制 CPU 资源使用，达到灵活调度。

OK，CPU 虚拟化就到这里，下篇文章将讲述内存虚拟化。觉得写得凑合可以给个赞，谢谢大家的支持。

我的公众号 **「Linux云计算网络」(id: cloud_dev)** ，号内有 **10T** 书籍和视频资源，后台回复 **「1024」** 即可领取，分享的内容包括但不限于 Linux、网络、云计算虚拟化、容器Docker、OpenStack、Kubernetes、工具、SDN、OVS、DPDK、Go、Python、C/C++编程技术等内容，欢迎大家关注。

作者：公众号「Linux云计算网络」，专注于Linux、云计算、网络领域技术干货分享

出处：https://www.cnblogs.com/bakari/p/7966671.html

本站使用「署名 4.0 国际」创作共享协议，转载请在文章明显位置注明作者及出处。

分类：云计算，虚拟化
标签：虚拟化，云计算

推荐 3　　赞赏　　收藏　　反对 0

« 上一篇： Qemu 简述
» 下一篇： 内存虚拟化

posted @ 2017-12-03 18:03 CloudDeveloper 阅读(5472) 评论(0) 收藏

注册用户登录后才能发表评论，请 登录 或 注册，访问 网站首页。

Copyright © 2020 CloudDeveloper
Powered by .NET Core on Kubernetes

Powered By Cnblogs | Theme Silence v2.0.0