

前面我们成功地把 KVM 跑起来了，有了些感性认识，这个对于初学者非常重要。不过还不够，我们多少得了解一些 KVM 的实现机制，这对以后的工作会有帮助。

**CPU 虚拟化**

KVM 的虚拟化是需要 CPU 硬件支持的。还记得我们在前面的章节讲过用命令来查看 CPU 是否支持KVM虚拟化吗？

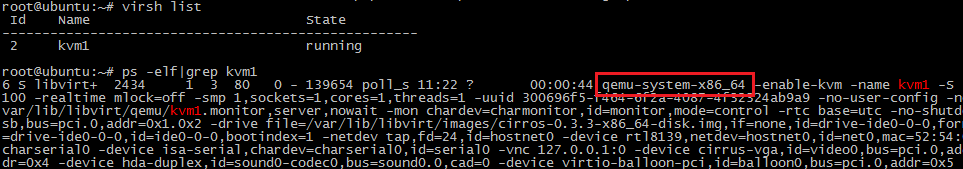
*root@ubuntu:~# egrep -o '(vmx|svm)' /proc/cpuinfo*

*vmx*

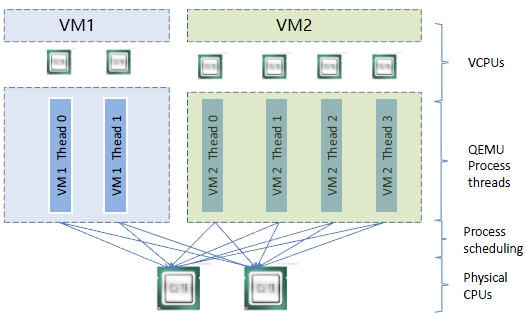
如果有输出 vmx 或者 svm，就说明当前的 CPU 支持 KVM。CPU 厂商 Intel 和 AMD 都支持虚拟化了，除非是非常老的 CPU。

一个 KVM 虚机在宿主机中其实是一个 qemu-kvm 进程，与其他 Linux 进程一样被调度。

比如在我的实验机上运行的虚机 kvm1 在宿主机中 ps 能看到相应的进程。



虚机中的每一个虚拟 vCPU 则对应 qemu-kvm 进程中的一个线程。看下图



在这个例子中，宿主机有两个物理 CPU，上面起了两个虚机 VM1 和 VM2。

VM1 有两个 vCPU，VM2 有 4 个 vCPU。可以看到 VM1 和 VM2 分别有两个和 4 个线程在两个物理 CPU 上调度。

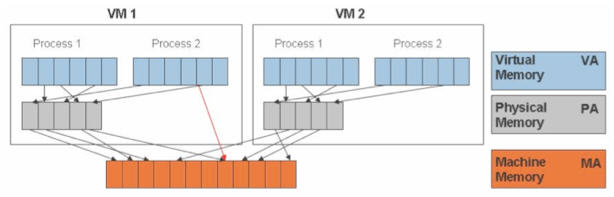
这里也演示了另一个知识点，即虚机的 vCPU 总数可以超过物理 CPU 数量，这个叫 CPU overcommit（超配）。

KVM 允许 overcommit，这个特性使得虚机能够充分利用宿主机的 CPU 资源，但前提是在同一时刻，不是所有的虚机都满负荷运行。

当然，如果每个虚机都很忙，反而会影响整体性能，所以在使用 overcommit 的时候，需要对虚机的负载情况有所了解，需要测试。

**内存虚拟化**

KVM 通过内存虚拟化共享物理系统内存，动态分配给虚拟机。看下图



为了在一台机器上运行多个虚拟机，KVM 需要实现 VA（虚拟内存） -> PA（物理内存） -> MA（机器内存）直接的地址转换。虚机 OS 控制虚拟地址到客户内存物理地址的映射 （VA -> PA），但是虚机 OS 不能直接访问实际机器内存，因此 KVM 需要负责映射客户物理内存到实际机器内存 （PA -> MA）。具体的实现就不做过多介绍了，大家有兴趣可以查查资料。

还有一点提醒大家，内存也是可以 overcommit 的，即所有虚机的内存之和可以超过宿主机的物理内存。但使用时也需要充分测试，否则性能会受影响。

下一节我们讨论 KVM 如何实现存储虚拟化。