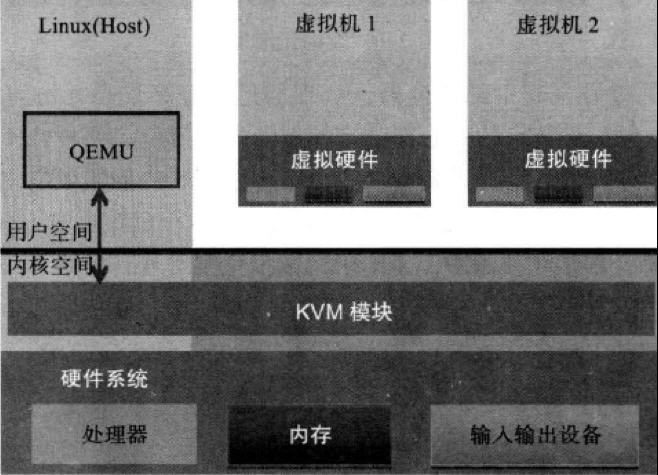
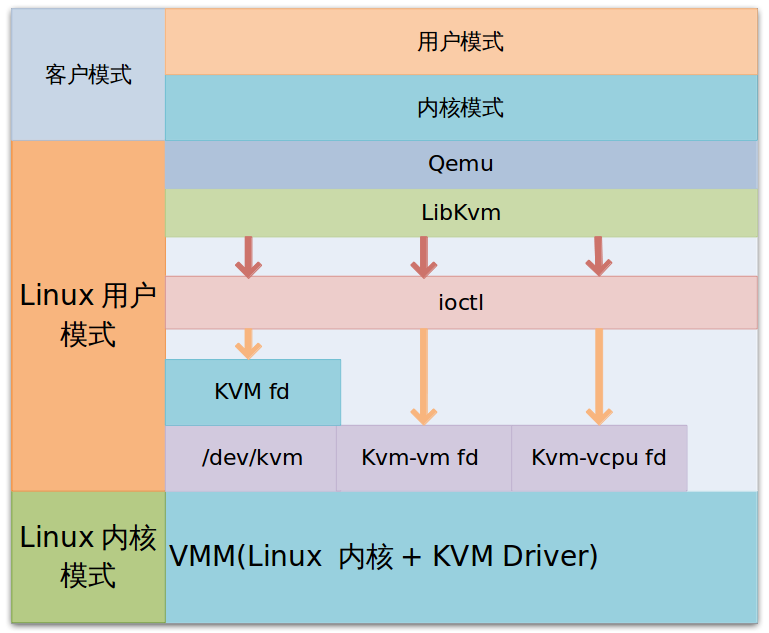
KVM全称是**基于内核的虚拟机（Kernel-based Vitrual Machine）**，它是Linux的一个内核模块，该内核模块使得Linux变成了一个Hypervisor。

**KVM架构**

KVM是基于虚拟化扩展（Intel VT或者AMD-V）的x86硬件的开源Linux原生的全虚拟化解决方案。KVM中虚拟机被实现为常规的Linux进程，由标准Linux调度程序进行调度，虚拟机的每个虚拟CPU被实现为一个常规的Linux进程，这使得KVM能够使用Linux内核的已有功能。

但是KVM本身不执行任何硬件模拟，需要客户空间程序通过/dev/kvm接口设置一个客户机虚拟服务器的地址空间，向它提供模拟的IO。





**Guest：**

客户机系统，包括CPU（vCPU）、内存、驱动，被KVM置于一种受限制的CPU模式下运行；

**KVM：**

运行再内核空间，提供CPU和内存的虚拟化，以及客户机的I/O拦截，Guest的I/O被KVM拦截后，交给QEMU处理，它属于标准Linux内核的一部分，是一个专门提供虚拟化功能的模块，主要负责CPU和内存的虚拟化，包括：客户机的创建、虚拟内存的分配、CPU执行模式的切换、vCPU寄存器的访问、vCPU的执行；

**QEMU：**

修改过的为KVM虚拟机使用的VMM，运行在用户空间，提供硬件I/O的虚拟化，通过IOCTL/dev/kvm设备和KVM交互。它是一个普通的Linux进程，为客户及提供设备模拟的功能，包括模拟BIOS、PCI/PCIE总线、磁盘、网卡、显卡、声卡、键盘、鼠标等，同时它通过ioctl系统调用与内核态的KVM模块进行交互。KVM是在硬件虚拟化支持下的玩全虚拟化技术，所以它支持在相应硬件上能运行的几乎所有的操作系统。

在KVM虚拟化架构下，每个客户机就是一个QEMU进程，在一个宿主机上有多少个虚拟机就会有多少个QEMU进程，客户机中的每个虚拟CPU对应QEMU进程中的一个执行线程，一个宿主机中只有一个KVM内核模块，所有客户机都与这个内核模块进行交互。

**CPU虚拟化：**

CPU是计算机系统最核心的模块，我们的程序执行到最后都是翻译为机器语言在CPU上执行的，在没有CPU硬件虚拟化技术之前，通常使用指令的二进制翻译来实现虚拟客户机中CPU指令的执行，很早起的VMware就是用这样的方案，器质性执行的翻译比较复杂，效率比较低。所以Intel最早发布的虚拟化技术就是CPU虚拟化方面的。称为VMX（vitual-machine extensions），有两种VMX操作模式：VMX根操作与VMX非根操作，作为虚拟机监视器中的KVM就是运行在根操作模式下，而虚拟机客户机的整个软件栈（包括操作系统和应用程序）则运行在非根操作模式下。VMX的非跟操作模式是一个相对受限的执行环境，为了适应虚拟化而专门作了一定的修改，在客户集中执行的一些特殊的敏感指令或者一些异常会触发VM Exit退到虚拟机监控器中，从而运行在VMX根模式，正是这样的限制，让虚拟机监控器保持了对处理器资源的控制。

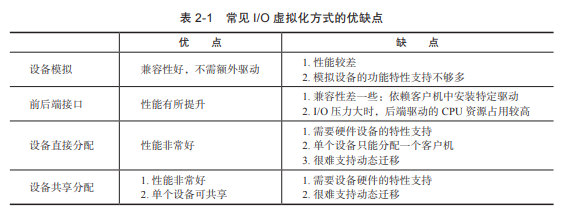
**内存虚拟化：**

内存虚拟化的目的是给虚拟客户机操作系统提供一个从0地址开始的连续物理内存空间，同时在多个客户机之间实现隔离和调度。

**I/O虚拟化：**

在虚拟化的架构下，虚拟机监控器必需支持来自客户机的I/O请求，通常情况下有以下4种虚拟化方式：

1. 设备模拟：在虚拟机监控器种模拟一个传统的/IO设备的特性，比如在QEMU种模拟一个Intel的千兆网卡或者一个IDE硬盘驱动器，在客户机种就爆露为对应的硬件设备，客户集中的I/O请求都由虚拟机监控器捕获并模拟执行后返还给客户机。
2. 前后端驱动接口：在虚拟机监控器与客户机之间定义一种全新的适合于虚拟化环境的交互接口，比如常见的virtio协议就是在客户集中暴漏为virtio-net、virtio-blk等网络和磁盘设备，在QEMU中实现相应的virtio后端驱动。
3. 设备直接分配：将一个物理设备，如一个网卡或硬盘驱动器直接分配给客户机使用，这种情况下I/O请求的链路中国很少需要或基本不需要虚拟机监控器的参与，所以性能很好。
4. 设备共享分配：其实是设备直接分配方式的一个扩展，在这种模式下，一个物理设备可以支持多个虚拟机功能接口，可以将虚拟功能接口独立分配给不同的客户机使用，如SR-IOV就是这种方式的一个标准协议。



**Intel虚拟化技术的发展：**

虚拟化技术从最初的纯软件的虚拟化技术，逐步发展到硬件虚拟化技术的支持，时至今日硬件虚拟化技术已比较成熟，Intel硬件虚拟化技术大致分为如下3个类别：

1. VT-x技术：

是指Intel处理器中进行的一些虚拟化技术支持，包括CPU中引用的最基础的VMX技术，使得KVM等硬件虚拟化基础的出现成为可能，同事也包括内存虚拟化的硬件支持EPT，VPID等技术。

1. VT-d技术：

是指Intel的芯片组的虚拟化技术支持，通过Intel IOMMU可以实现对设备直接分配的支持。

1. VT-c技术：

是指Intel的I/O设备相关的虚拟化技术支持，主要包含两个技术，一个是借助虚拟机设备队列（VMDq）最大限度提高I/O吞吐率，VMDq由Intel网卡中的专用硬件来完成，另一个是借助虚拟机直接互联(VMDc)大幅提升虚拟化性能，VMDc主要就是基于SR-IOV标准将单个Intel网卡产生多个VF设备，用来直接分配给客户机。

**KVM内核模块**

KVM内核模块是标准Linux内核的一部分，由于KVM的存在让Linux本身就变成了一个Hypervisor，可以原生的支持虚拟化功能，目前KVM支持多种处理器平台，它支持嘴常见的以Lnux和AMD为代表的x86平台，也支持PowerPC、ARM等平台。

KVM模块是KVM虚拟化的核心模块，它在内核中由两部分组成：一个是处理器架构无关的部分，用lsmod命令中可以看到，叫做kvm模块，另一个是处理器架构相关的部分，在Intel平台上就是kvm\_intel这个内核模块，KVM的主要功能是初始化PCU硬件，打开虚拟化模式，然后将虚拟客户机运行在虚拟机模式下，并对虚拟客户机的运行提供一定的支持。

KVM仅支持硬件辅助的虚拟化，所以打开并初始化系统硬件以支持虚拟机的运行，是KVM模块的职责所在，以KVM在Intel公司的CPU上运行为例，在被内核加载的时候，KVM模块会初始化内部的数据结构，做好准备后KVM模块会检测当前系统的CPU，然后代开CPU控制寄存器CR4中的虚拟化模式开关，并通过执行VMXON执行将宿主操作系统（包括KVM模块本身）置于CPU执行模式的虚拟化模式中的根模式，最后，KVM模块创建特殊设备文件/dev/kvm并等待来自用户空间的命令，接下来，虚拟机的创建和运行将是一个用户空间的应用程序（QEMU）和KVM模块项目配合的过程。

/dev/kvm这个设备可以被当作一个标准的字符设备，KVM模块与用户空间QEMU的通信接口主要是一系列针对这个特殊设备文件的Ioctl调用，当然，每个虚拟客户机针对/dev/kvm文件的最重要的Ioctl调用就是“创建虚拟机”。KVM还会返回一个文件句柄来代表所创建的虚拟客户机，针对该文件句柄的Ioctl调用可以对虚拟机做相应的管理，比如创建用户空间虚拟地址和客户机屋里地址及真实内存物理量地址的映射关系，再比如创建多个可供运行的虚拟处理器(vCPU)。

针对虚拟处理器的最重要的Ioctl调用就是“执行虚拟处理器”，通过它，用户空间准备好的虚拟机再KVM模块的支持下，被置于虚拟化模式中的非根模式下，开始执行二进制指令，在非根模式下，所有敏感的二进制指令都会被处理器捕捉到，处理器在保存现场后自动切换到根模式，由KVM决定如何进一步处理。

**QEMU用户态设备模拟**

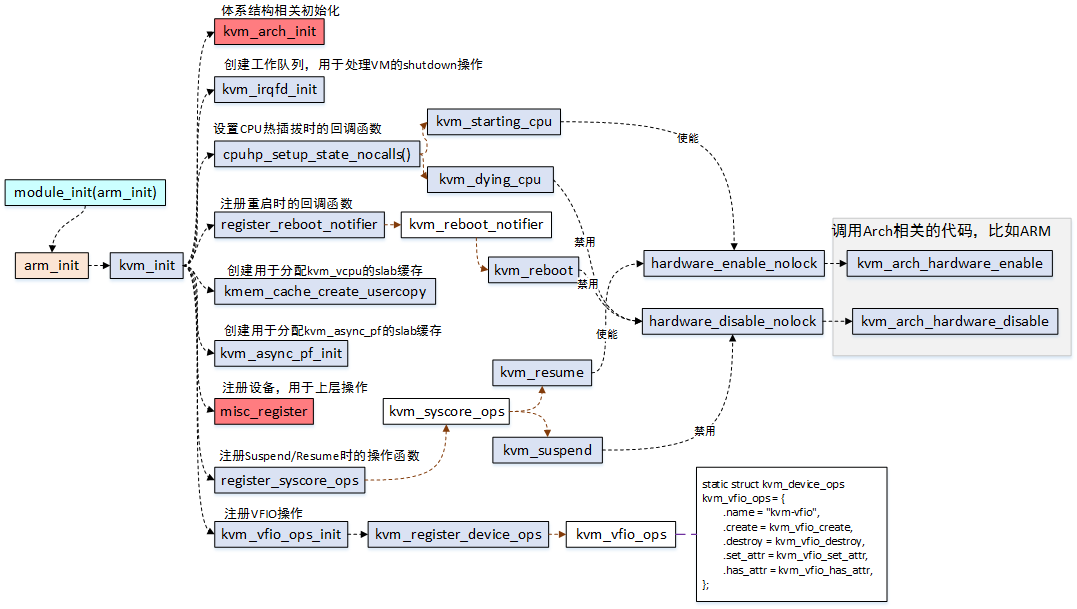
QEMU原本是一个开源的软件项目，实现的虚拟机是一个纯软件的实现，通过二进制翻译来实现虚拟化客户机中的CPU指令模拟，所以性能比较低，但是其优点是跨平台，甚至可以支持客户机与宿主机并不是同一个架构。每一个虚拟客户机再宿主机中就体现为一个QEMU进程，而客户机的每一个虚拟CPU就是要给QEMU线程，虚拟机运行期间，QEMU会通过KVM模块提供的系统调用进入内核，由KVM模块负责将虚拟机置于处理器的特殊模式下运行，遇到虚拟机进行I/O操作时，KVM模块会从上册的系统调用出口处返回QEMU，由QEMU来负责解析和模拟这些设备。

QEMU除了提供完全模拟的设备，还支持virtio协议中的设备模拟，virtio是一个沟通客户机前端设备与宿主机上设备后端模拟的比较高性能的协议，在前端客户机中需要安装相应的virtio-blk、virtio-scsi、vitrio-net等驱动，而QEMU就实现了virtio的虚拟化后端，QEMU还提供了叫做virtio-blk-data-plane的一种高性能的块设备I/O方式，使得块设备I/O效率进一步提高。

**QEMU结合KVM执行Guest OS指令的步骤简要描述：**

打开/dev/kvm设备文件、创建虚拟机、创建vCPU、向KVM模块发送KVM\_RUN和ioctl调用，接下来KVM模块利用CPU上的硬件虚拟化扩展功能来直接执行guest的运行，当guest在执行过程中遇到读取硬件设备寄存器等I/O操作、暂停guest CPU或其他特殊指令等KVM无法执行的操作时，会退出当前KVM并交由QEMU处理，QEMU模拟器判断退出原因并执行相应的操作。

**KVM初始化**



1. 内核的功能模块

完成模块初始化；

相应各类请求。

1. KVM的初始化，在kvm\_init中完成，即包含了体系结构相关的初始化设置，也包含了各类回调函数的设置，资源分配，以及设备注册等，只有当初始化完成后，才能响应各类请求，比如创建虚拟机等。
2. 回调函数设置，这几个模块的回调函数，最终都会去调用体系结构相关的函数去打开或变比Hypervisor。
3. 资源分配，kmem\_cache\_create\_usercopy和kvm\_async\_pf\_init都是创建slab缓存，用于内核对象的分配。
4. kvm\_vfio\_ops\_init，VFIO是一个可以安全将设备I/O、中断、DMA导出到用户空间的框架，后续将IO虚拟化时再深入分析。

kvm\_arch\_init的流程：

