完全虚拟化 和 半虚拟化

完全虚拟化 是捕捉指令 然后模拟

半虚拟化 是捕捉指令 一半模拟 一半是cpu直接运行

## 内存虚拟化

客户机 的内存虚拟的 物理地址空间 需要转换成实际物理地址

所以一个逻辑地址或线性地址需要经过两次转换。

DMA代表直接内存访问

有DMA控制器 会与系统总线通信，将数据从外部设备传输到内存或者从内存传输到外部设备，而不需要CPU的参与。一旦数据传输完成，DMA控制器可以发出中断通知CPU。

Intel VT-d技术通过在北桥MCH引入**DMA重映射硬件**

根据设备对应的IO页表，硬件可以对DMA中的地址进行转换，使设备只能访问到规定的内存。

## kvm原理

多个客户机就是宿主机中多个qemu进程

vcpu就是qemu进程中的多线程

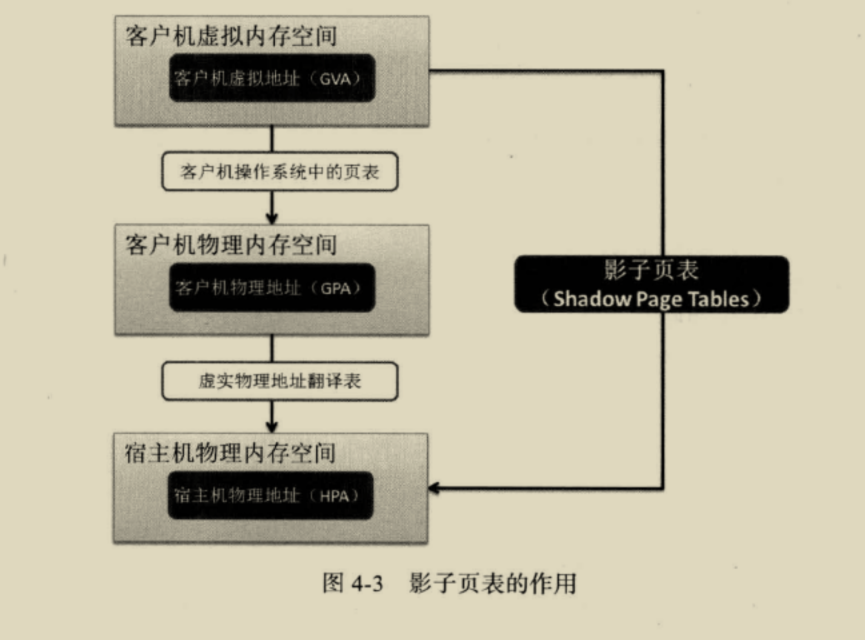
## smp是什么？

并发执行cpu核心执行任务需求 分块任务 分给多个cpu并发执行任务

## EPT是什么？

之前是软件实现的功能 后面CPU自带了这个功能 把影子页（也就是客户机的虚拟地址和真实地址的数据库）存在寄存器里面

客户虚拟内存空间 -> (客户机操作系统的页表) -> 客户物理内存空间 -> 叙事物理地址翻译表 -> 宿主机物理内存空间



自从CPU自带了ETP之后 就可以减少 VM-Exit的次数了

## VM-Exit 是什么？

VM-Exit 是虚拟化技术中的一个重要概念，用于描述虚拟机从运行在虚拟化环境中的客户操作系统切换到宿主操作系统（hypervisor）的过程。

## VM-Exit 什么时候执行

一 、

关键事件的发生：VM-Exit 可能在关键事件发生时执行，例如：

当虚拟机尝试执行特权指令（例如修改页表、更改控制寄存器等）时。

当虚拟机尝试执行无法虚拟化的指令或者特定于宿主硬件的指令时。

当虚拟机需要访问宿主系统资源（如网络、存储等）时。

当虚拟机需要响应来自宿主系统或其他虚拟机的事件时。

二、

I/O 访问：当虚拟机尝试执行需要 I/O 访问的指令（例如读写硬盘、网络通信等）时，可能会导致 VM-Exit。这是因为虚拟机通常无法直接访问物理硬件，而需要通过宿主系统进行中介。

## VIPD是什么？

在传统的虚拟化中，每次虚拟机切换到宿主操作系统或者另一个虚拟机时，都需要刷新处理器的 TLB（转换后备缓冲区）和其他缓存。这种刷新操作会带来性能开销。

引入 VPID 技术后，处理器可以为每个虚拟机分配一个唯一的 VPID。这样，当虚拟机切换时，处理器可以根据当前的 VPID 来区分不同的虚拟机，而不需要刷新整个 TLB。

## TLB是什么？

它存储了虚拟地址到物理地址的映射关系，加速了内存地址的访问速度。

## 查看是否开启EPT和VPID

cat /sys/module/kvm\_intel/parameters/ept

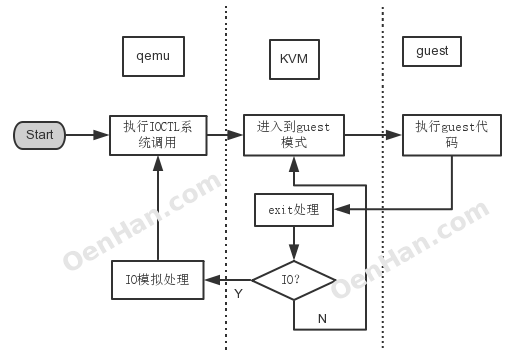
cat /sys/module/kvm\_intel/parameters/vpid

## Huge Page是什么？

传统的内存管理中，操作系统会将物理内存划分成固定大小的页面（通常为4KB），用于管理进程的内存分配和访问。然而，对于一些特定的应用场景，如大型数据库、虚拟化平台、科学计算等，使用传统的页面会导致一些性能瓶颈，主要是由于内存管理的开销和内存碎片化等问题。

Huge Page技术通过增加页面的大小来解决这些问题，允许操作系统将内存分配为更大的页面（通常为2MB或更大）。这些大页面可以减少内存管理的开销，并且提高内存访问的效率，因为更多的数据可以被放置在一个页面中，减少了页表项的数量和内存访问的开销。

## KVM运行原理



上图是一个执行过程图，

* 首先启动一个虚拟化管理软件qemu，开始启动一个虚拟机
* 通过ioctl等系统调用向内核中申请指定的资源，搭建好虚拟环境，启动虚拟机内的OS，执行 VMLAUCH 指令，即进入了guest代码执行过程。
* 如果 Guest OS 发生外部中断或者影子页表缺页之类的事件，暂停 Guest OS 的执行，退出QEMU即guest VM-exit，进行一些必要的处理，然后重新进入客户模式，执行guest代码；这个时候如果是io请求，则提交给用户态下的qemu处理，qemu模拟处理后再次通过IOCTL反馈给KVM驱动。

## qemu-kvm命令 注意

有逗号，就加转译 ，，两个逗号代表一个逗号

例如用于"-cpu qemu64,+vmx"

改成 -hda my,,file

## 磁盘镜像文件格式

对磁盘镜像文件进行一致性检查，查找镜像文件中的错误，目前仅支持对“qcow2’

qed”、“vdi”格式文件的检查。其中，qcow2 是 QEMU 0.8.3 版本引人的镜像文件格式，也

是目前使用最广泛的格式。qed (QEMU enhanced disk) 是从 QEMU 0.14 版开始加人的增强

磁盘文件格式，避免了 qcow2 格式的一些缺点，也提高了性能，不过目前还不够成熟。而

vdi ( Virtual Disk Image)是 Oracle 的 VirtualBox 虚拟机中的存储格式。参数 -f fmt 是指定文

件的格式，如果不指定格式，qemu-img 会自动检测。ilename 是磁盘镜像文件的名称(包括

路径 )。

# 网络

## stp协议

STP 的主要目标是避免在以太网网络中出现环路，因为环路会导致数据包在网络中不断循环，造成网络拥塞和数据丢失。STP 通过选择一条根网桥（Root Bridge）以及在网络中选择一条最佳路径，来防止网络中出现环路。

### 选择根网桥：

STP 协议中的所有交换机都会参与选举根网桥的过程。根网桥是网络中具有最低 Bridge ID（桥标识符）的交换机。Bridge ID 由优先级和 MAC 地址组成，优先级越低、MAC 地址越小的交换机被选举为根网桥。

一旦根网桥选举完成，每个交换机就会计算到根网桥的最佳路径，并将该路径作为主路径（Root Port）。这样，每个交换机都只会维护到根网桥的一条最佳路径。

## NAT

多个主机公用一个ip

bridge iptables dnsmasq（轻量级的DHCP和DNS服务器软件）

跟桥接模式的区别 就是把虚拟机的ip段改成跟宿主机不一样的ip段

## qemu内部的用户模式网络

qemu自己实现的功能 不用安装其他东西

qemu内部实现所有网络协议栈 所以性能差

不支持部分网络功能（如icmp） 所以客户机不能ping

不能从宿主机和外部网络访问客户机 跟nat一样

## icmp是什么？

ICMP是Internet Control Message Protocol（互联网控制消息协议）的缩写。它是在TCP/IP协议族中的一个子协议，用于在IP网络上发送控制消息和错误报告。

## virtio是什么？

Virtio是一种I/O虚拟化解决方案，它将前端（在虚拟机内部）和后端（在虚拟化宿主机或物理设备上）之间的通信协议化，从而实现高效的虚拟设备访问。前后端驱动是Virtio架构中的两个重要组成部分，它们协同工作以提供虚拟设备的功能。

以下是Virtio前后端驱动的基本原理：

前端驱动（Frontend Driver）： 前端驱动运行在虚拟机内部，作为虚拟机中的设备驱动程序。它负责与虚拟设备进行通信，并将来自虚拟机的I/O请求传递给后端驱动。

后端驱动（Backend Driver）： 后端驱动运行在虚拟化宿主机或物理设备上，负责实际的I/O设备管理和控制。它接收来自前端驱动的I/O请求，并将其转发到底层的物理设备或者虚拟化宿主机的相应模块。

通信协议： 前后端驱动之间的通信通常通过一种称为Virtqueue的环形缓冲区队列来实现。前端驱动将I/O请求放入Virtqueue中，后端驱动则从Virtqueue中获取请求并执行相应的操作。完成操作后，后端驱动将结果写回到Virtqueue中，前端驱动再将结果传递给虚拟机中的应用程序。

特定设备的实现： 根据具体的虚拟设备类型（例如网卡、磁盘等），前后端驱动会有不同的实现。例如，对于网络设备，前端驱动会处理网络数据包的发送和接收，而后端驱动则负责管理物理网络接口并处理数据包的转发。

virtio-ring 实现了 环形缓冲区（ring buffer）

一次性保存前端驱动多次 i/o请求 然后再发给后端驱动处理再返回结果

virtio半虚拟化驱动的方式 性能几乎可以达到和native差不多的I/O性能

linux 默认都自动安装了

windows需要额外安装

## Ballonning 动态控制内存 （不推荐用）

修改内存大小 需要重启

但是通过它就可以动态新增减少了

她是第三方的程序 客户机和宿主机从这里拿内存的控制权

## virtio\_net是什么？

内核级别的后端处理程序

## tap是什么

TAP（即网络设备）是一种虚拟网络设备，常用于虚拟化环境中。它可以让用户创建一个虚拟的网络接口，并且可以与物理网络接口进行通信。

## KVM\_CLOCK是什么

因为虚拟机会暂停，所以时间也会暂停 这样时间就不同步了

cpu有一个不变的constant TSC频率 那这个做基础 实现的时间精准的问题

开启参数 -cpu host

## VT-d是什么

英特尔叫VT-d AMD叫AMD-Vi(IOMMU)

可以让虚拟机独占pci设备 缺点不能多个虚拟机一起用

网卡 硬盘 usb显卡 等等pci设备都可以分配

## SR-IOV是什么

physical funciton = pf

virtual function = vf

SR-IOV（Single Root I/O Virtualization，单根 I/O 虚拟化）是一种用于提高虚拟化环境中网络和存储设备性能的技术。硬件支持该功能才可以用

就是上面VT-D的多虚拟主机使用的功能

是一种基于硬件的虚拟化技术。它允许在虚拟机之间高效地共享PCIe（Peripheral Component Interconnect Express，快速外设组件互连）设备

SR-IOV技术将一个PCIe的网络控制器虚拟化成多个PCIe设备，并且每个设备可以直接分配给一个虚拟机。这样做的好处是，虚拟机可以直接控制硬件，从而避免了虚拟设备的性能会随着宿主机的性能而改变，另外宿主机由于需要进行数据处理，时延等也产生了

VF分配的数量由设备的硬件设计机器驱动程序共同决定

igb驱动的82576每个PF最多支持7个VF

ixgbe驱动的82599最多支持63个VF

## 迁移

动态迁移

kvm自身支持

注意内存修改速度不能快过迁移速度 要不然会迁移失败

也就是业务不能频繁在跑

如果有vt-d/sr-iov的存在 就需要先取消挂载 迁移之后在新的客户机在挂上去

要不然不能迁移

用libvirt可以实现该功能 动态迁移

## KSM是什么

让相同的虚拟主机 系统相同或则应用程序相同 内核同页合并 如果有进程需要修改那就会复制出一个新的内存页让他修改 这样就可以实现超开

## 内存页是什么？

用于存储和管理程序执行时的数据和指令。计算机内存被划分成许多固定大小的内存页，通常是4KB或者更大的大小。

## 大页

可以用hugetlbfs文件系统来实现1G大页 但是需要显式调用libhugetlb库的函数 也就是手动分配内存调用等等

缺点 用不完就是浪费 不能交换到交换分区swap

### 为什么需要大页

内存管理效率提高： 使用大页可以减少内存管理的开销。传统的小页（通常为4KB）需要更多的页表项来管理相同数量的内存，而大页（通常为2MB或更大）可以减少页表的数量，从而降低了内存管理的开销。

TLB（Translation Lookaside Buffer）效率提高： TLB 是用于加速虚拟地址到物理地址转换的高速缓存。使用大页可以减少 TLB 缓存的条目数，从而提高 TLB 的命中率，减少 TLB 的失效率，加速内存访问速度。

减少内存碎片： 使用大页可以减少内存碎片的产生。大页的分配和释放单位更大，因此更不容易产生碎片，可以更有效地利用物理内存

提高性能： 大页可以提高系统性能。因为大页可以减少内存管理和地址转换的开销，减少了内存访问的延迟，提高了系统的整体性能。

### 透明大页

配置开启一下就可以 自动使用 大页了 不需要什么操作 只要硬件支持

## AVX/XSAVE是什么

可以将AVX指令集比喻为一种工厂里的自动化生产线。传统的生产线只能处理一个产品，而AVX指令集就像是将生产线升级成了多功能机器，可以同时处理多个相似的产品。

## AES-NI是什么

AES-NI指令集提供了一组硬件加速的指令，可以在处理器级别上执行AES算法中的加密和解密操作。通过使用AES-NI指令集，处理器可以更快地执行AES算法，从而加速加密和解密过程。

## SMEP是什么

SMEP通过限制内核模式下的代码执行，防止了用户态的代码被直接执行，从而增强了操作系统的安全性。一旦启用了SMEP，操作系统的内核将无法执行来自用户空间的代码，从而有效地阻止了一些攻击类型，如代码注入和执行攻击。

## cgroups是什么

限制资源用的

下面是一些使用cgroups控制进程的命令示例，用于限制PHP-FPM进程的资源使用：创建一个新的cgroup：

bashCopy code

sudo mkdir/sys/fs/cgroup/cpu/php-fpm

将PHP-FPM进程添加到新创建的cgroup中：

首先，找到PHP-FPM进程的PID（进程ID），可以使用以下命令找到：

Copy code

pgrep php-fpm

然后将PID添加到cgroup中：

rubyCopy code

sudo echo <PID> > /sys/fs/cgroup/cpu/php-fpm/tasks

设置CPU时间配额：

rubyCopy code

sudo echo 50000 > /sys/fs/cgroup/cpu/php-fpm/cpu.cfs\_quota\_us

这个命令将设置PHP-FPM进程的CPU时间配额为50000微秒（50毫秒），这意味着PHP-FPM进程每50毫秒只能使用一定量的CPU时间。

设置内存限制：

rubyCopy code

sudo echo 100M > /sys/fs/cgroup/memory/php-fpm/memory.limit\_in\_bytes

这个命令将设置PHP-FPM进程的内存限制为100MB，即PHP-FPM进程只能使用最多100MB的内存。

## 各个参数

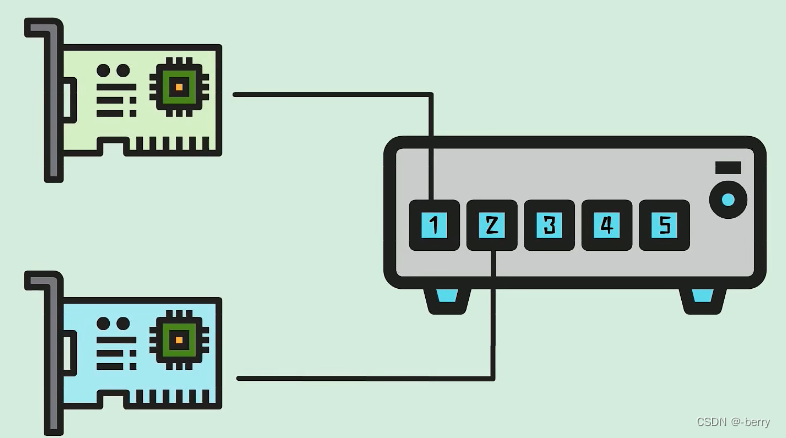
250页

# 配置虚拟机

## 交换机是什么？

多个设备需要通讯 通过他来转发信息 根据目标mac地址

**我们可以把交换机比作邮递员，根据数据包中目的mac地址，找到它对应的物理端口，一台交换机有很多端口，他们都有自己的编号。交换机子关心数据包中的mac地址，不关心IP地址，MAC地址在TCP/IP协议中，处于计算机网络中的第二层数据链路层，即二层设备。**

****

## 网关是什么？

**网关：负责多个不同网段之间的数据通信，如果在同一个网段通信，就不需要网关了(TCP/IPx协议规定，不同子网之间是不可以直接通信的，如果需要通信需要网关来转发)**

## 光猫是什么？

**一般是把光信号转换为电信号**

## 路由是什么？

**指的是根据目标IP判断是否如何发送（将数据包从源地址传输到目标地址的过程）**

****

路由在第三层 （网络层）

网关在第二层 数据链路层

网关是通过mac

路由是通过ip

NAT模式

如果外面想要访问里面的虚拟机 就需要做端口转发

桥接模式

里面的虚拟机 可以直接被外面的网络访问

要执行 才能让流量通过

iptables -I FORWARD -m state -d 172.23.19.0/24 --state NEW,RELATED,ESTABLISHED -j ACCEPT

两个命令 用后面的

firewall-cmd --permanent --zone=public --add-rich-rule='rule family="ipv4" destination address="172.23.19.0/24" accept'

* **-m state:** 启用状态匹配模块（state matching module）
* **–-state:** 状态匹配模块的参数。当SSH客户端第一个数据包到达服务器时，状态字段为NEW；建立连接后数据包的状态字段都是ESTABLISHED
* **–sport 22:** sshd监听22端口，同时也通过该端口和客户端建立连接、传送数据。因此对于SSH服务器而言，源端口就是22
* **–dport 22:** ssh客户端程序可以从本机的随机端口与SSH服务器的22端口建立连接。因此对于SSH客户端而言，目的端口就是22

## 虚拟机可以对外的原理

当本地电脑请求一个ip地址的时候 会ARP请求广播 问一下mac地址是多少

当虚拟机设置ip的时候就告诉了路由器 路由器会有一个arp表记录下来 mac地址

然后从路由器那边得知是宿主机的虚拟机 所以第一次要从宿主机进去跟虚拟机通讯确认能不能通过 这个mac地址是否有效

如果已经知道了 就是本地arp缓存表读取mac地址 直接通讯

第二次的时候这时候不走宿主机请求拿到mac了

通讯是发送方设备会将数据包发送到网络中，并在数据包中指定目标设备的MAC地址。网络中的网络设备（如交换机）会根据目标MAC地址将数据包转发到目标设备上。

## TCP封包是什么？

就是三次握手过程

# 多线程

## 并发是什么？

CPU运行时间划分成若干个时间段,再将时间 段分配给各个线程执行，在一个时间段的线程代码运行时，其它线程处于挂起状。.这种方式我们称之为并发(Concurrent)。

## 并行是什么？

当系统有一个以上CPU时,则线程的操作有可能非并发。当一个CPU执行一个线程时，另一个CPU可以执行另一个线程，两个线程互不抢占CPU资源，可以同时进行，这种方式我们称之为并行(Parallel)。

并行指的是多个任务可以同时执行，异步是多个任务并行的前提条件

## 串行是什么？

串行是指多个任务时，各个任务按顺序执行

多线程：多线程是程序设计的逻辑层概念，它是进程中并发运行的一段代码。

多线程可以实现线程间的切换执行。

## IOMMU是什么？

iommu是GPU虚拟机直通的核心，负责把主板上的设备分组，分成一个一个group，每个group都可以单独的启用或者停用，在没启用iommu之前，宿主系统会占用全部的硬件。在启用iommu之后，宿主系统可以有选择的预留一些硬件资源，把这些硬件资源分配给虚拟机

是一种硬件设备，通常集成在主板的芯片组中，用于管理输入输出设备与主存储器之间的内存访问。它的主要作用是在物理内存地址与设备直接访问之间提供地址转换和访问控制，从而增强系统的安全性和性能。

IOMMU允许操作系统对设备发出的内存访问进行地址转换，将设备访问的物理地址映射到实际的系统内存地址。这样可以实现多个设备同时访问系统内存，而不会发生地址冲突。同时，IOMMU还可以提供内存隔离和访问控制功能，限制设备只能访问被授权的内存区域，从而增强系统的安全性。

在虚拟化环境中，IOMMU对于直接分配物理设备给虚拟机（如使用VFIO技术）非常重要。它可以确保虚拟机对分配的设备进行安全的、受控的访问，同时保护宿主系统的内存不受恶意设备的非法访问。

自带直通功能 可以控制访问的内存是安全的 内存是需要显卡和系统传输数据

他不是自动开的 需要直通功能才开 可以不开就有直通功能

实现方法：

禁用 IOMMU： 在主板 BIOS 中，您可以查找并禁用 IOMMU。这通常需要对BIOS进行进一步的设置和调整，因为不是所有的主板都会提供直接关闭 IOMMU 的选项。

配置虚拟机： 在创建虚拟机时，您需要将显卡设备分配给虚拟机。您可以通过编辑虚拟机的 XML 配置文件，将显卡添加为 PCI 设备。

加载虚拟机所需的驱动程序： 在虚拟机中，您可能需要加载适当的显卡驱动程序以确保显卡正常工作。

禁用虚拟机监控器（VMware）： 如果您正在使用 VMware 等虚拟化平台，您可能需要禁用虚拟机监控器，以确保虚拟机可以直接访问显卡而不受虚拟化层的限制。

请注意，这种方法可能会导致严重的安全风险，因为虚拟机可以直接访问主机物理内存。此外，由于缺乏 IOMMU 的地址转换和管理，可能会导致性能问题或系统不稳定。因此，在生产环境中，强烈建议使用 IOMMU 来实现 GPU Passthrough，以确保系统的安全性和稳定性。

建议启动IOMMU带的直通功能

GPU Passthrough

## VFIO 是什么？

IOMMU是底层功能 VFIO是api 可以调用IOMMU功能 让用户可以使用功能

VFIO代表“Virtual Function I/O”，它是一种技术，用于将物理设备（通常是图形处理器或其他加速器设备）直接分配给虚拟机以进行高性能的 I/O 操作。这种技术通常与虚拟化平台（如KVM）结合使用，允许虚拟机在不牺牲性能的情况下访问物理设备。VFIO通过在宿主系统和虚拟机之间建立严格的设备分离来实现这一点，这样虚拟机就可以像访问自己专用的设备一样访问被分配的物理设备。

## VT-x是什么？

VT-x（Virtualization Technology）：VT-x是英特尔的虚拟化技术，它提供了对处理器级别的虚拟化支持。通过VT-x，处理器可以在硬件级别上支持虚拟化指令集，允许在虚拟机中运行的操作系统直接访问处理器的硬件功能。这大大提高了虚拟化的性能和效率，使得虚拟机可以更快地执行指令和访问硬件资源。

## VT-d是什么？

VT-d（Virtualization Technology for Directed I/O）：VT-d是英特尔的I/O虚拟化技术扩展，它增强了对直通设备的支持。通过VT-d，处理器可以提供更强大的I/O虚拟化功能，包括对设备的直通（passthrough）和硬件加速的DMA（Direct Memory Access）隔离。这使得虚拟机可以直接访问物理设备，而不需要通过宿主系统进行中介，从而提高了设备访问的性能和安全性。

## DMA是什么？

硬件加速技术

DMA控制器通常是在主板上的北桥或南桥芯片中实现的，而不是在CPU内部。这些芯片是主板上的核心芯片组件，负责管理系统的I/O通信和数据传输。

硬件加速的DMA（Direct Memory Access）是一种技术，用于提高数据传输效率并减少CPU的负载。DMA允许外设（如网络适配器、磁盘控制器等）直接访问系统内存，而不需要CPU的干预。

传统的DMA是由CPU控制的，即CPU负责管理数据传输的起始、终止和中断处理等。而硬件加速的DMA则是通过硬件控制器来实现，这样可以减少CPU的干预，提高数据传输的效率。在硬件加速的DMA中，数据传输过程由专门的DMA控制器来管理，CPU只需设置传输参数并且等待传输完成的通知，而无需实际参与数据传输的过程。

对于虚拟化环境来说，硬件加速的DMA尤其重要。在虚拟机中，如果设备直通（passthrough）给了虚拟机，那么虚拟机需要能够以尽可能高效的方式与直通设备进行通信。使用硬件加速的DMA可以帮助虚拟机直接与直通设备进行数据传输，而无需通过宿主系统的干预，从而提高了数据传输的效率和性能。

总的来说，硬件加速的DMA是一种提高数据传输效率并减少CPU干预的技术，特别适用于虚拟化环境中对设备直通的支持。

## docker和kvm区别

kvm是内核级虚拟化 需要转换地址调用

kvm 支持容器启动 容器启动 不是内核启动 虚拟机用容器模式启动 区别就是 容器轻量 共享内核 不像内核启动那样 需要各种模拟cpu等等

<os>

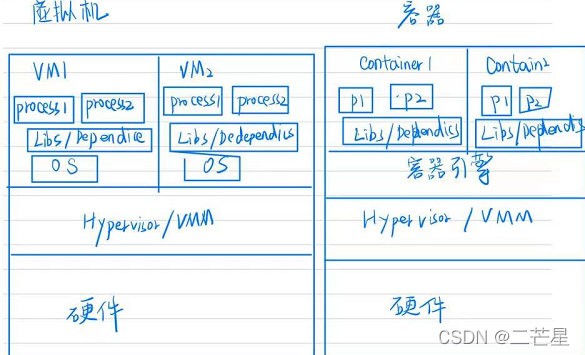
<type arch='x86\_64'>exe</type>

//exe 表示虚拟机是在模拟环境中运行的 hvm 表示虚拟机是通过硬件虚拟化技术（如Intel VT-x或AMD-V）来运行的完整虚拟机。

</os>

docker是操作系统虚拟化 直接调用硬件 用了容器引擎

此处要注意虚拟机和容器的区别：虚拟化技术是模拟出了一个虚拟机，而容器技术只是在与原宿主机的架构中加入一块新的空间，该空间对于硬件资源的使用还是得靠底层硬件，不过不是虚拟化的操作系统了，而是容器引擎技术。



全虚拟化是什么？

遇到不能虚拟化操作的时候

比如 遇到操作硬件的时候 需要

客户操作系统运行在 Ring 1，它在执行特权指令时，会触发异常（CPU的机制，没权限的指令会触发异常），然后 VMM 捕获这个异常，在异常里面做翻译，模拟，最后返回到客户操作系统内，客户操作系统认为自己的特权指令工作正常，继续运行。但是这个性能损耗，就非常的大，简单的一条指令，执行完，了事，现在却要通过复杂的异常处理过程。

#### **半虚拟化是什么？**

半虚拟化的思想就是， 修改操作系统内核，替换掉不能虚拟化的指令，通过超级调用（hypercall）直接和底层的虚拟化层hypervisor来通讯，hypervisor 同时也提供了超级调用接口来满足其他关键内核操作，比如内存管理、中断和时间保持。

这种做法省去了全虚拟化中的捕获和模拟，大大提高了效率。所以像XEN这种半虚拟化技术，客户机操作系统都是有一个专门的定制内核版本，和x86、mips、arm这些内核版本等价。这样以来，就不会有捕获异常、翻译、模拟的过程了，性能损耗非常低。这就是XEN这种半虚拟化架构的优势。这也是为什么XEN只支持虚拟化Linux，无法虚拟化windows原因，微软不改代码啊。

超级调用是什么？

就是hypervisor跟虚拟机通讯的接口方式

#### **硬件辅助的全虚拟化 是什么？**

就是qemu可以直接控制cpu 让cpu做操作 这时候 linux会暂停

1、在全虚拟化或硬件辅助虚拟化的情况下，QEMU 直接控制 CPU 并模拟虚拟机的行为，这时 QEMU 不会自己暂停自己。它将直接与虚拟化扩展（如 Intel VT-x 或 AMD-V）交互，以模拟和控制虚拟机的执行，而不需要宿主操作系统的介入。

2、在用户态模拟的情况下，QEMU 是通过用户空间程序运行的，但在这种模式下，它通常会通过系统调用与宿主操作系统交互。当虚拟机中的程序执行时，QEMU 将解释客户机指令并模拟其行为，但它不会暂停自己。相反，它将继续运行以模拟虚拟机的行为。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 利用二进制翻译的全虚拟化 | 硬件辅助虚拟化 | 操作系统协助/半虚拟化 |
| 实现技术 | BT 和直接执行 | 遇到特权指令转到root模式执行 | Hypercall |
| 客户操作系统修改/兼容性 | 无需修改客户操作系统，最佳兼容性 | 无需修改客户操作系统，最佳兼容性 | 客户操作系统需要修改来支持hypercall，因此它不能运行在物理硬件本身或其他的hypervisor上，兼容性差，不支持Windows |
| 性能 | 差 | 全虚拟化下，CPU需要在两种模式之间切换，带来性能开销；但是，其性能在逐渐逼近半虚拟化。 | 好。半虚拟化下CPU性能开销几乎为0，虚机的性能接近于物理机。 |
| 应用厂商 | VMware Workstation/QEMU/Virtual PC | VMware ESXi/Microsoft Hyper-V/Xen 3.0/KVM | Xen |