# 异常的虚拟化

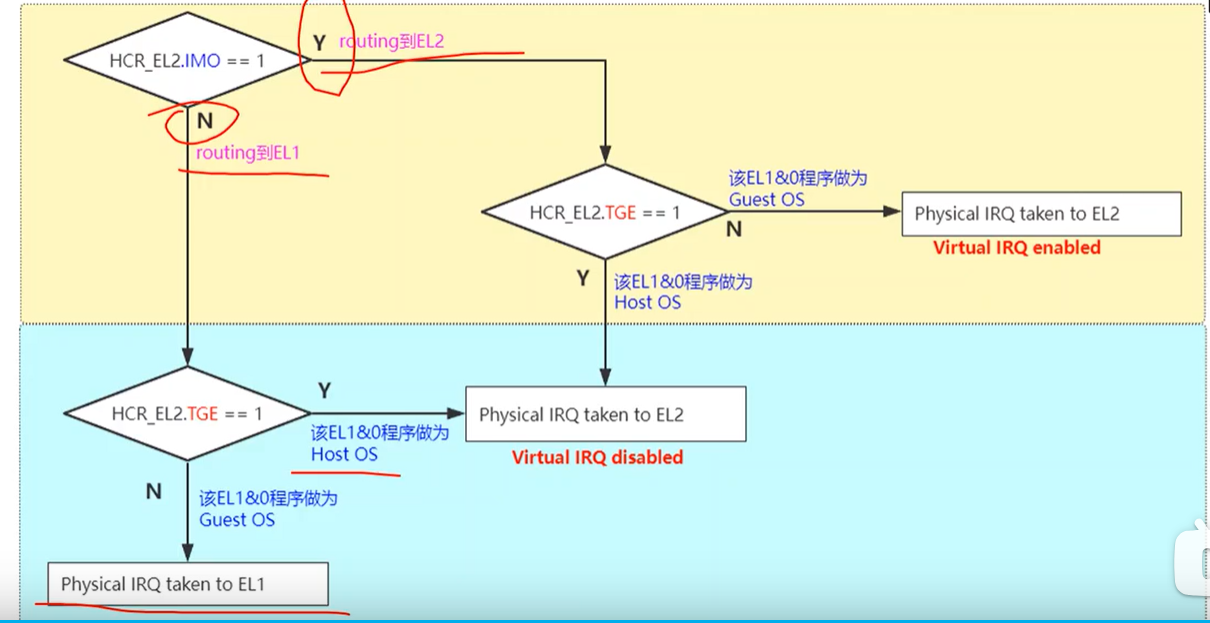
中断是硬件通知软件的机制，在一个使用虚拟化的系统中，中断处理会变得更为复杂。有些中断会由Hypervisor直接处理，有些中断被分配给了VM，需要由VM中的处理程序处理，并且还有可能在接收到这个中断时，对应的VM并没有被调度运行。这意味着我们不仅需要支持在EL2中直接处理中断，还需要一种机制能将收到的中断转发给相应VM的vCPU。Armv8提供了vIRQs, vFIQs, 和vSErrors来支持虚拟中断。这些中断的行为和物理中断（IRQs, FIQs, 和 SErrors）类似，只不过只有当系统运行在EL0/1是才会收到，运行在EL2/3是收不到虚拟中断的。

使用虚拟化的系统，一些中断可能被hypervisor自己来处理，一些可能由VM的软件来处理。为了避免出现VM此时没有被调度但针对该VM的中断发生的情况，ARM采用了两个机制：

1. 一些中断支持在EL2级被hypervisor解决
2. 其他中断能交给对应的VM或vCPU

为了实现这2种机制，ARM包含了对虚拟中断的支持：vIRQ和vFIQ。这两种中断只能在当执行El0和El1发生。

VM通常只会接受虚拟中断，而不是真实中断。



当运行在EL0时，HCR\_EL2.TGE控制使用EL1还是EL2空间，当应用运行在Guest OS (TGE==0)为前者，运行在Host OS（TGE==1）为后者。

## 开启虚拟中断

虚拟中断也是根据中断类型控制的。为了发送虚拟中断到EL0/1, Hypervisor需要设置 HCR\_EL2中相应的中断路由比特位。例如，开启vIRQ，你需要设置 HCR\_EL2.IMO， 这意味着物理IRQ中断将被发送到EL2，同时虚拟中断将被发送到EL1。理论上，Armv8可以配置成VM直接接收物理FIQs和虚拟IRQs。但在实际应用中，通常配置VM只接收虚拟中断。

## 产生虚拟中断

**有2种机制可以用来产生虚拟中断。**

a. 通过设定HCR\_EL2寄存器的3个特定位VI、VF、VSE来产生一个对vCPU的中断信号。这样由hypervisor来模拟产生中断，会带来额外运行开销。

b. **使用GIC产生虚拟中断。**GIC通过两套相同的接口来产生真实和虚拟中断。hypervisor可以将虚拟CPU接口映射到VM，实现VM和GIC的直接通信，从而不需要hypervisor来模拟中断

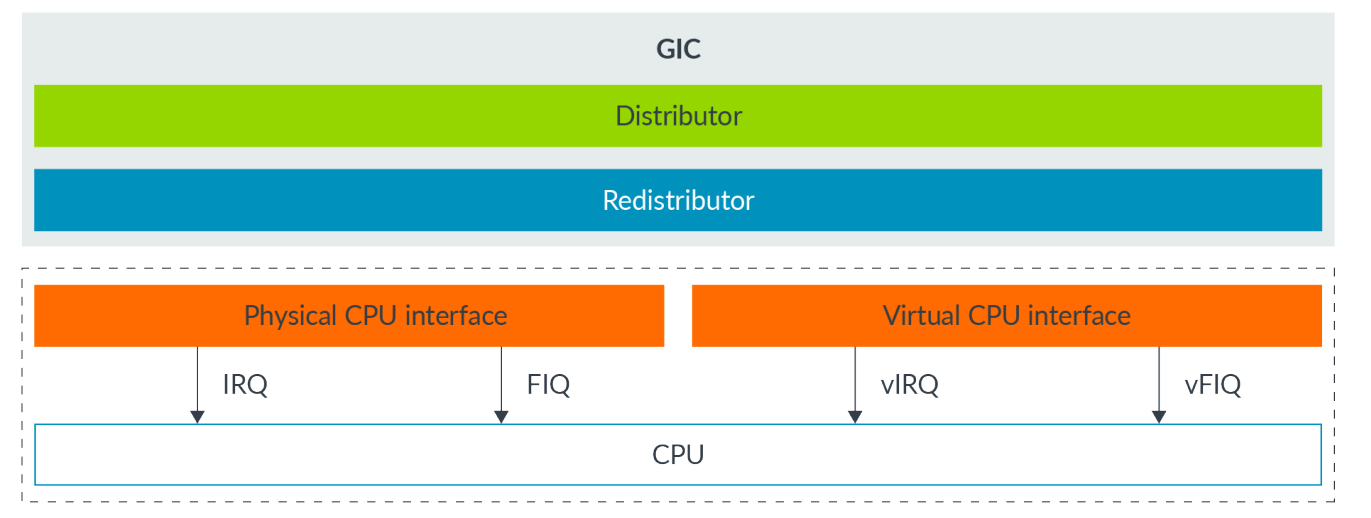
* 我们先来看第一种机制，HCR\_EL2中有如下的控制比特位

VI: 配置vIRQ

VF: 配置vFIQ

VSE: 配置vSError  
设置上述比特位等同于中断控制器向vCPU发送中断信号。和常规物理中断一样，虚拟中断受PSTATE控制。这种机制简单易用，但有个明显的缺点，需要由Hypervisor来模拟中断控制器的相关操作，一系列的 陷入 – 模拟将带来性能上的开销。

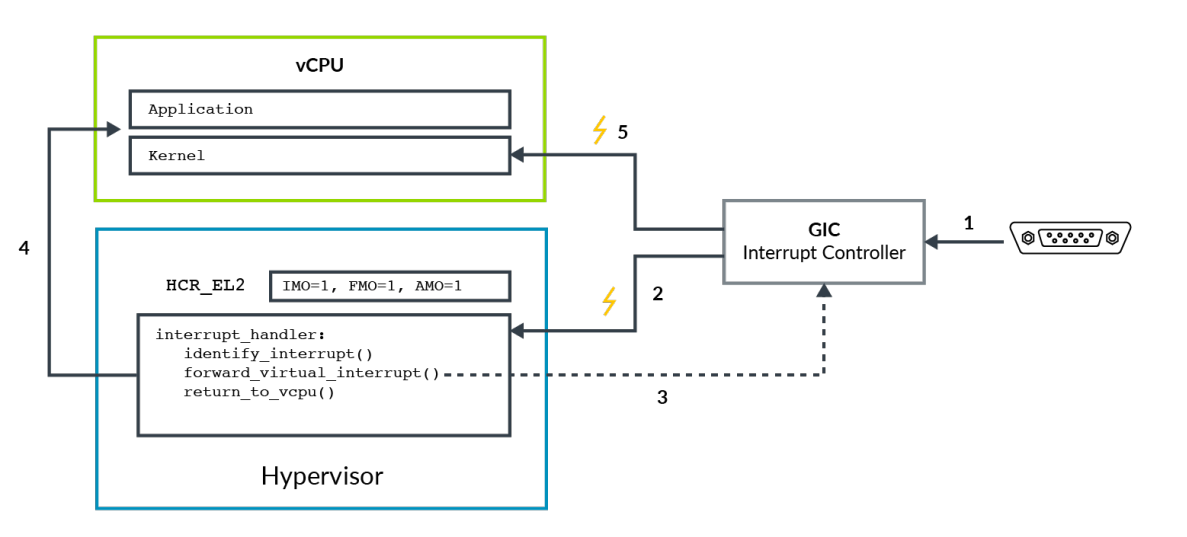
**第二种方式是使用**Arm的通用中断控制器(Generic Interrupt Controller, GIC)来产生虚拟中断。从GICv2版本开始，GIC可以通过物理CPU interface 和 虚拟CPU interface发送物理中断和虚拟中断。见下图：

[](https://mdpics4lgw.oss-cn-beijing.aliyuncs.com/aliyun/202307231533552.png)

这两个CPU interface是等同的，区别是一个发送物理中断信号，另一个发送虚拟中断信号。Hypervisor可以将虚拟CPU interface映射给VM，以便VM可以直接和GIC通信。这种方式的好处是Hypervisor只需建立映射，不需要做任何模拟，从而提升了性能。**（PS：虚拟化性能提升的关键就在优化陷入，减少次数，优化流程）**

* 一个例子

假设使用GIC来产生虚拟中断。一台物理外设连接GIC，当物理外设发出中断信号时，交由GIC处理，之后的步骤如下图所示：

[](https://mdpics4lgw.oss-cn-beijing.aliyuncs.com/aliyun/202307231554730.png)

具体步骤如下：

1. 物理外围设备发送中断信号给GIC。
2. GIC产生物理中断异常，可能是IRQ或FIQ。由于配置了HCR\_EL2.IMO/FMO，这些异常会被路由到EL2。Hyperviosr发现该设备已被分配给了某个VM，于是检查需要将该中断信号转发给哪个vCPU。
3. Hypervisor配置了GIC将该物理中断以虚拟中断的形式转给某个vCPU。GIC于是发送vIRQ/vFIQ信号，如果此时还运行在EL2，这些信号会被忽略。
4. Hypervisor将控制权返还给vCPU。
5. 处理器运行在EL0或EL1，来自GIC的虚拟中断被接收（受PSTATE控制）。

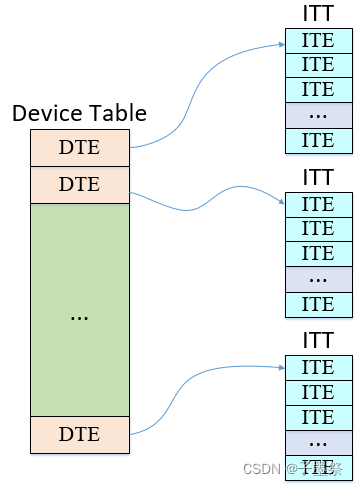
上面的例子展示了如何将一个物理中断以虚拟中断的形式转发给VM。如果是一个没有物理中断对应的纯虚拟中断，Hypervisor可以直接注入虚拟中断。

kvm\_set\_irq通过ioctl KVM\_IRQ\_LINE向VM注入中断。KVM\_SIGNAL\_MSI可以注入msi中断。KVM\_SET\_GSI\_ROUTING可以设置gsi 路由

# Gic3

## Device table

Device table提供了一组device table entry（DTE），其中每个DTE为指向特定deviceID相关的中断转换表（ITT）基地址。其转换关系如下：



## 中断转换表

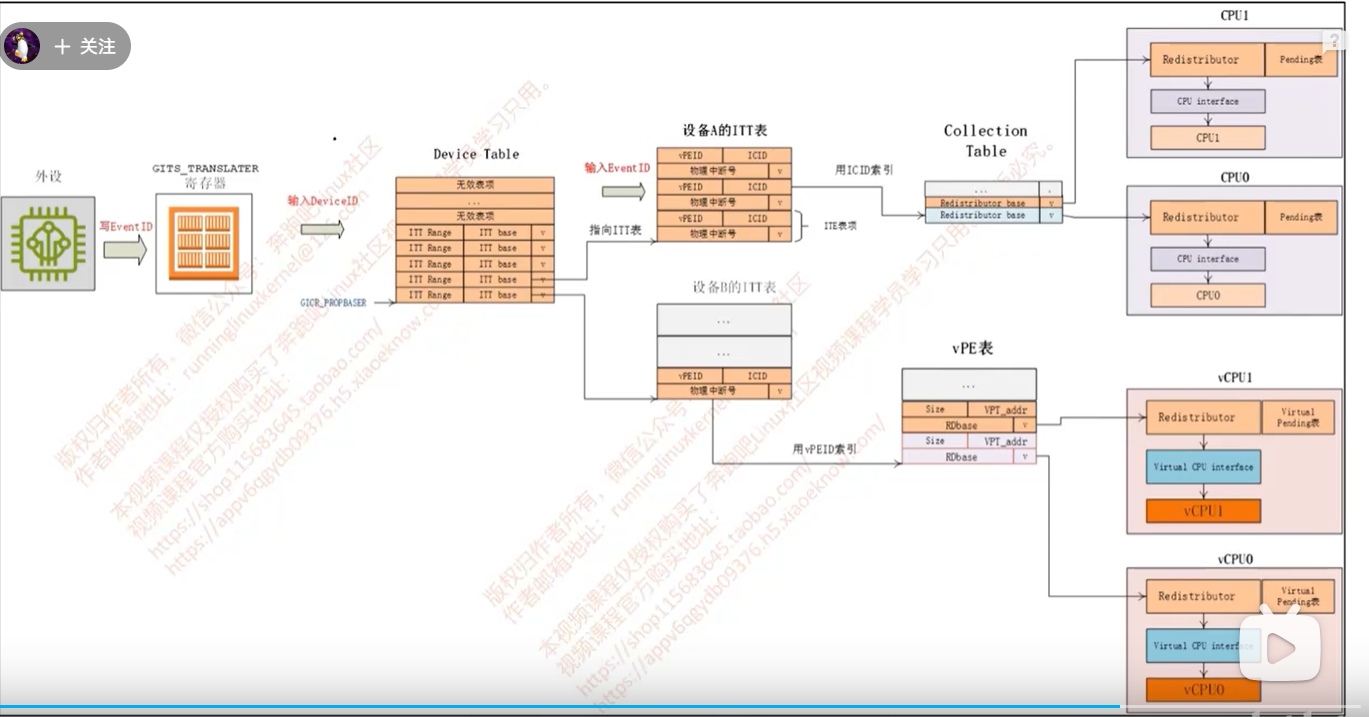
GICv3是基于armv8的SOC设计中应用较为广泛的一种中断控制器，GICv4与GICv3的功能基本相同，只是为了提高虚拟化的性能，增加了直接注入虚拟中断的能力。

每个连接到ITS上且含有多个event的设备，都拥有一张ITT表。对于物理中断和虚拟中断ITE分别提供了两种类型的映射，对于物理中断其映射关系如下：  
（1）eventID到中断ID的映射

（2）eventID到ICID的映射，其中ICID又指向collection表

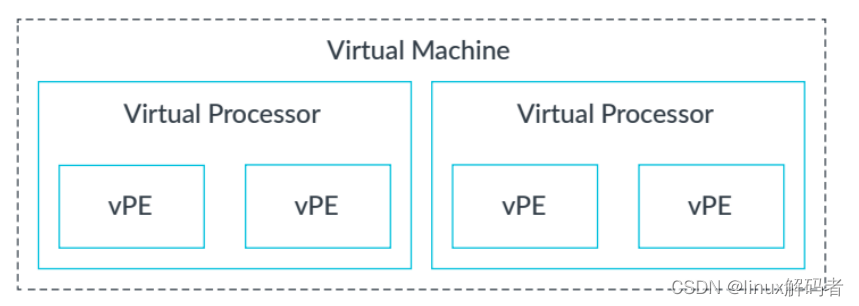
对于虚拟中断其映射关系如下：  
（1）eventID到虚拟中断ID的映射

（2）eventID到vPE表的映射，vPE表包含了虚拟PE number（vPEID）



## vPE

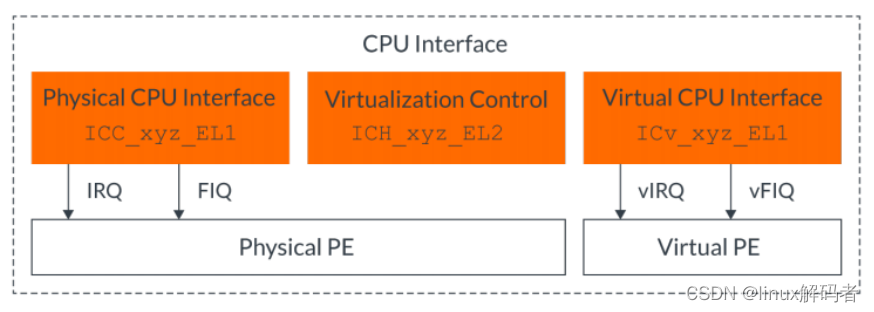
[Hypervisor](https://so.csdn.net/so/search?q=Hypervisor&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/flyingnosky/article/details/_blank)创建，控制并管理虚拟机VM。一个虚拟机在功能上等于一个物理系统并包含一个或多个虚拟处理器。每个处理器包含一个或多个虚拟PE。



**CPU Interface分成三组：**

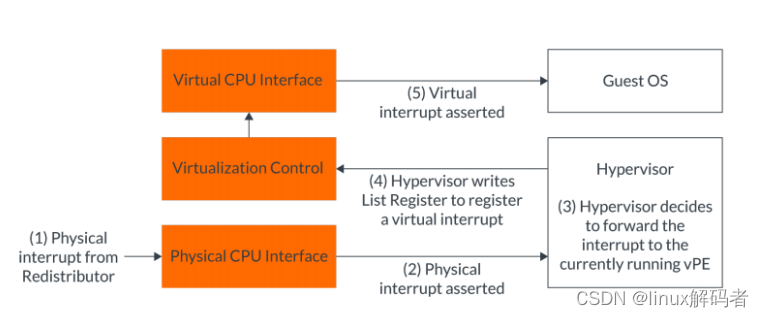
1. ICC：物理CPU Interface寄存器
2. ICH：虚拟化控制寄存器
3. ICV：虚拟CPU interface寄存器

        下图呈现了CPU interface寄存器的三个组：



**一个物理中断发给一个vPE的例子**

下图呈示了一个物理中断发往一个vPE的例子：



这个过程如下：

1. 一个物理中断从Redistributor发往一个物理CPU interface
2. 物理CPU interface判断是否将物理中断发往PE。在这种情况下，判断通过，发出物理异常
3. 中断被带到EL2。Hypervisor读IAR，它返回pINTID。当前pINTID处于Active状态。Hypervisor决定中断被发往当前运行的vPE。Hypervisor写pINTID到ICC\_EOIR1\_EL1。在ICC\_CTLR\_EL1.EOImode=1时，这时仅发出priority drop而没有deactivating物理中断
4. Hypervisor写List寄存器将一个虚拟中断注册为pending。List寄存器指定了被发送的vINTID和原始pINTID。Hypervisor发出一个异常，将异常发送给vPE
5. 虚拟CPU interface判断是否虚拟中断被发送给一个vPE。这个判断与物理中断相同，除了它们使用ICV寄存器外。在这种情况下，判断通过，发出一个虚拟异常
6. 虚拟异常被带到EL1。当软件读IAR时，返回vINTID且虚拟中断处于Active状态
7. Guest OS处理中断。当它完成中断处理时，它写EOIR来发出priority drop和deactivation。因为List寄存器记录的是pINTID，它将对vINTID和pINTID同时deactivate。

这个例子显示了一个物理中断作为虚拟中断被发送给vPE。比如，它可以由外设通过hypervisor发送给VM。并非所有虚拟中断都是由于物理中断产生。虚拟化软件在任何时候在List寄存器中产生虚拟中断。

## MSI和vPE映射

IMG_256

Hypervisor可以通过写ICH\_LRn寄存器向guest OS注入一个虚拟中断。系统中有多个LR寄存器，即可同时注入多个虚拟中断。LR寄存器上的值会被反映在virtual CPU interface上，guest OS读CPU interface时实际会读出LR中的值。

•vINTID是报告给guest OS的虚拟中断号，这是一个guest OS domain 内的中断编号，可以是guest OS的SPI/PPI/SGI/LPI。

•pINTID是虚拟中断关联的物理中断，这意味着物理设备发出原始的pINTID物理中断，而hypervisor收到物理中断后将其映射为虚拟中断再转发给guest OS的vPE。vPE在ack虚拟中断的同时也会ack关联的物理中断。

•Priority是虚拟中断的优先级，group表明这是一个普通中断IRQ还是快速中断FIRQ，state声明了虚拟中断的状态(pending/active)。

KVM会调用physical GIC驱动配置物理ITS的interrupt translation table以建立虚拟中断到vPE的映射：device ID, event ID -> (vpe, vintid, pintid)。这里device ID和event ID来自MSIX消息，而vintid(virtual LPI)来自guest OS配置vGIC时MAPTI命令里传入的vLPI。之所以需要pintid(physical LPI)是防止中断触发时vPE不处于运行状态，这时仍然需要使用physical LPI触发hypervisor KVM介入以上一页介绍的软件慢速形式向guest OS注入中断。

## VM entry 和中断注入

* 【中断采集】

中断的采集是指将guest VM的设备中断请求送入对应的虚拟中断控制器中。Guest VM的中断有两种可能的来源：

来自于软件模拟的虚拟设备，比如一个模拟出来的串口，可以产生一个虚拟中断。从VMM的角度来看，虚拟设备只是一个软件模块，可以通过调用虚拟中断控制器提供的接口函数，实现虚拟设备的中断发送。

来自于直接分配给guest VM的物理设备的中断，比如一个物理网卡，可以产生一个真正的物理中断。一个物理设备被直接分配给一个guest VM，意味着当该设备发生中断时，中断的处理函数（ISR）应该位于guest OS中。

可是在虚拟化环境中，物理中断控制器是由VMM控制的，因而VMM在收到中断后，会首先判断该中断是不是由分配给guest VM的设备产生的，如果是的话，就将中断发送给对应的虚拟中断控制器。而后，虚拟中断控制器会在适当的时机将该中断注入guest VM，由guest OS中的ISR进行处理。那中断注入的过程是怎样的，这个“适当的时机”又该如何选择呢？

* 【中断注入】

虚拟中断控制器采集到的中断请求，将按照VMM排定的优先级，被逐一注入到对应的虚拟CPU中。在Linux的信号发送与接收机制中，只有从内核空间返回到用户空间，也就是某个进程被调度到重新获得CPU的使用权时，该进程才可以处理之前发送给它的那些[信号](https://zhuanlan.zhihu.com/p/77598393)，或者说此时内核才可以将这些未处理的信号注入到进程中。

同样地，只有在VM entry，也就是某个虚拟CPU被调度到重新获得物理CPU的使用权时，VMM才可以将中断注入到该虚拟CPU中。

为了保证中断的及时注入，就需要通过一定的手段，强制虚拟CPU发生VM exit，然后在VM entry返回guest VM的时候注入中断。强制产生VM exit最常用的办法就是往虚拟CPU对应的物理CPU发送一个IPI核间中断。这个IPI就像一把手枪，把guest VM打下来，落回到VMM中。

是不是VMM注入的中断，虚拟CPU就必须照单全收呢？Linux中的进程可以通过设置SIG\_IGN来忽略某个信号（SIGKILL和SIGSTOP除外），同样地，虚拟CPU也可以通过配置虚拟IMR(Interrupt Mask Register)来选择是否屏蔽某个中断。

# GIC 中断虚拟化

<https://zhuanlan.zhihu.com/p/535997324>