二、 概述

在这一章中，我们先介绍一些贯穿全书的概念，比如 Hypervisor，VT-x，VT-d，SVM 等等。然后我们会简略介绍下 NewBluePill 项目背景及其所采用的硬件虚拟化技术。

这一章只是介绍这些技术大致的轮廓，详细内容会在后面各章节中逐一介绍。

Hypervisor 概述

虚拟化的历史

在讨论 Hypervisor 之前首先谈谈虚拟，虚拟（virtualization）指对计算机资源的抽象，一种常用的定义是“虚拟就是这样的一种技术，它隐藏掉了系统，应用和终端用户赖以交互的计算机资源的物理性的一面，最常做的方法就是把单一的物理资源转化为多个逻辑资源，当然也可以把多个物理资源转化为一个逻辑资源（这在存储设备和服务器上很常见）”

实际上，虚拟技术早在 20 世纪 60 年代就已出现，最早由 IBM 提出，并且应用于计算技术的许多领域，模拟的对象也多种多样，从整台主机到一个组件，其实打印机就可以看成是一直在使用虚拟化技术的，总是有一个打印机守护进程运行在系统中，在操作系统看来，它就是一个虚拟的打印机，任何打印任务都是与它交互，而只有这个进程才知道如何与真正的物理打印机正确通信，并进行正确的打印管理，保证每个 job 按序完成。

长久以来，用户常见的都是进程虚拟机，也就是作为已有操作系统的一个进程，完全通过软件的手段去模拟硬件，软件再翻译内存地址的方法实现物理机器的模拟，比如较老版本的 VMWare, VirtualPC 软件都属于这种。

在 2005 年和 2006 年，Intel 和 AMD 都开发出了支持硬件虚拟技术的 CPU，也就是在这时，x86 平台才真正有可能实现完全虚拟化[1]。[1]在 2007 年初的时候，Intel 还进一步的发布了 VT-d 技术规范，从而在硬件上支持 I/O 操作的虚拟化。随着硬件虚拟化技术越来越广泛的采用，开发者也开始虚拟技术来做一些其他的事情：当前 HVM 已经在虚拟机，安全，加密等领域上有所应用，例如 VMware Fusion, Parallels Desktop for Mac, Parallels Workstation 和 DNGuard HVM，随着虚拟化办公和应用的兴起，相信虚拟化技术也会在未来得到不断发展。

硬件虚拟化技术（HEV）

有了虚拟技术的基本概念，下面我们谈谈硬件虚拟化技术。硬件虚拟化技术（Hardware Enabled Virtualization，本书中简称 HEV），也就是在硬件层面上，更确切的说是在 CPU 里(VT-d 技术是在主板上北桥芯片支持)，对虚拟技术提供直接支持。在硬件虚拟化技术诞生前，编写虚拟机过程中，为了实现多个虚拟机上的真实物理地址隔离，需要编程实现把客户机的物理地址翻译为真实机器的物理地址。同时也需要给不同的客户机操作系统编写不同的虚拟设备驱动程序，使之能够共享同一真实硬件资源。硬件虚拟化技术则在硬件上实现了内存地址甚至于 I/O 设备的映射，因此大大简化了编写虚拟机的过程。而其硬件直接支持二次寻址和 I/O 映射的特性也提升了虚拟机在运行时的性能。[2]

[1] 完全虚拟化(Full Virtualization)，完整虚拟底层硬件，这就使得能运行在该底层硬件上的所有操作系统和它的应用程序，也都能运行在这个虚拟机上。

[2] 一些优化技术也在硬件中被采用，比如专门用于二次寻址的 TLB，详细信息可以参考 Intel 和 AMD 的手册

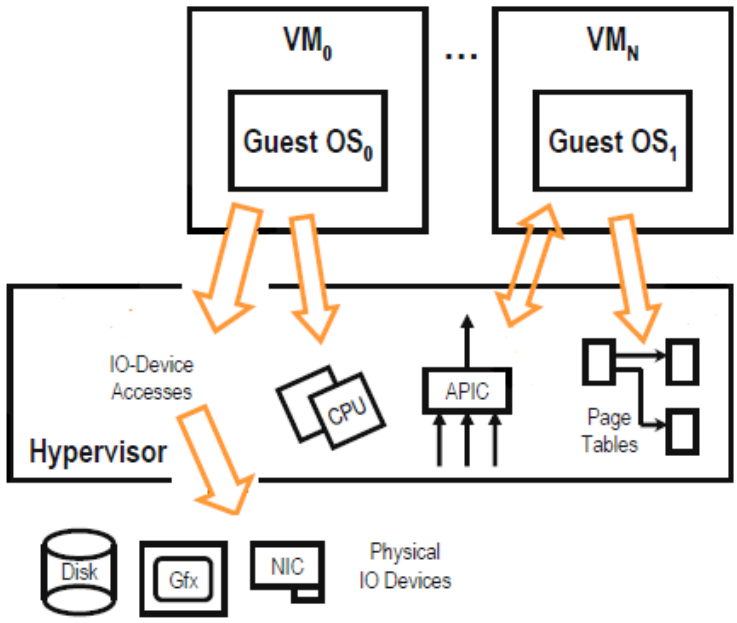

图 2.1 硬件虚拟化技术架构示意图

在硬件虚拟化技术中，一个重要的概念就是 HVM。HVM, Hypervisor Virtual Machine 的缩写（在本书中简称为 Hypervisor），是在使用硬件虚拟化技术时创建出来的特权层，该层提供给虚拟机开发者，用来实现虚拟硬件与真实硬件的通信和一些事件处理操作（如图 2.1），因此 Hypervisor 的权限级别要高于等于操作系统权限。

虚拟机的启动过程

使用了硬件虚拟化技术的虚拟机可以有三种引导 Guest 操作系统的方式：

1. 存在特殊 OS/Host OS，后启动 Hypervisor 的虚拟机启动过程
2. 存在特殊 OS/Host OS，先启动 Hypervisor 的虚拟机启动过程
3. 不存在特殊 OS/Host OS，先启动 Hypervisor 的虚拟机启动过程

- 存在特殊 OS/Host OS，后启动 Hypervisor 的虚拟机启动过程。采用这种启动过程的虚拟机代表是 KVM，其启动过程如下：
  a) 先启动宿主 Linux 操作系统
  b) 在 Linux 中启动 KVM 设备，从而启动了 Hypervisor
  c) 启动虚拟机，作为 Linux 进程运行

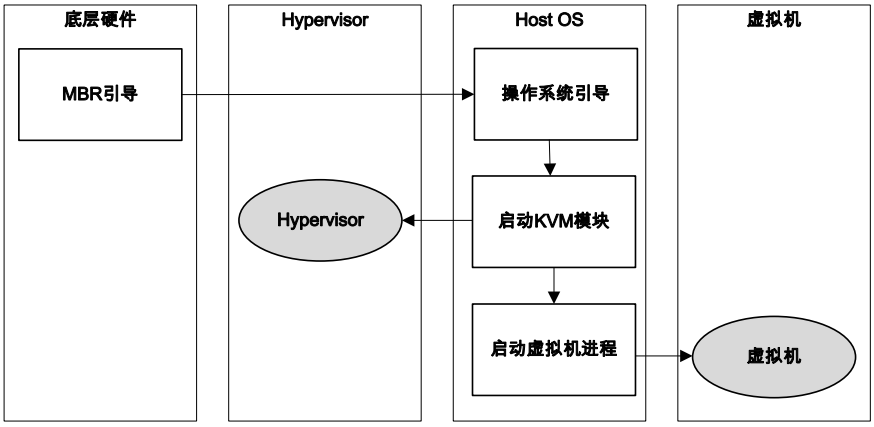

图 2.2 KVM 中虚拟机的启动过程

启动过程如图 2.2，可以看出，KVM 启动虚拟机的模式说明它不想脱离进程级虚拟机的本质，但是它要利用虚拟化技术进行加速。这样做的缺点在于需要一个 Host OS 充当载体。除 KVM 外，VMWare6.5 以上版本也是采用类似的架构，使用支持 HEV 技术的 CPU 进行加速。但是它们都需要再另外安装相应 Guest OS 上的驱动。

- 存在特殊 OS/Host OS，先启动 Hypervisor 的虚拟机启动过程。采用这种启动过程的虚拟机代表是 Xen，其启动过程如下：
  a) 先创建并启动 Hypervisor
  b) 引导 Dom0
  c) 由 Hypervisor 和 Dom0 一起协作创建虚拟机
  d) 启动该虚拟机[1]

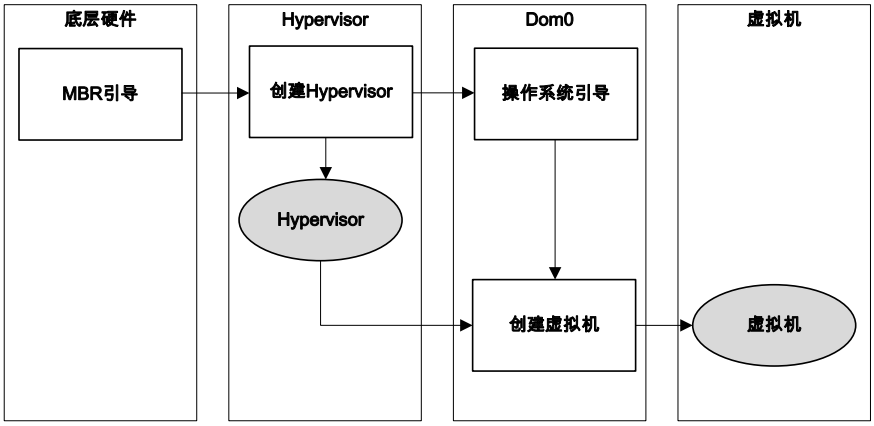

图 2.3 Xen 中虚拟机的启动过程

启动过程如图 2.3，可以看出，Xen 中仍存在 Dom0 是因为它要适应过去未出现 HEV 技术时的架构，所以无论是 Dom0 还是 Hypervisor 的实现都比较笨重，并且安装和配置也比较麻烦，同样需要另外安装相应 Guest OS 上的驱动。但是不可忽视的是

[1] Xen 中具体创建和启动虚拟机的过程会在“第 14 章 其它有关 HEV 项目”中介绍

Xen 的虚拟化效率最高。

- 不存在特殊 OS/Host OS，先启动 Hypervisor 的虚拟机启动过程。当前暂时没有采用这种启动过程的虚拟机软件（暂时称之为 UVM, Unknown Virtual Machine），其启动过程如下：
  a) 先创建并启动 Hypervisor
  b) 从 Hypervisor 中创建并启动虚拟机

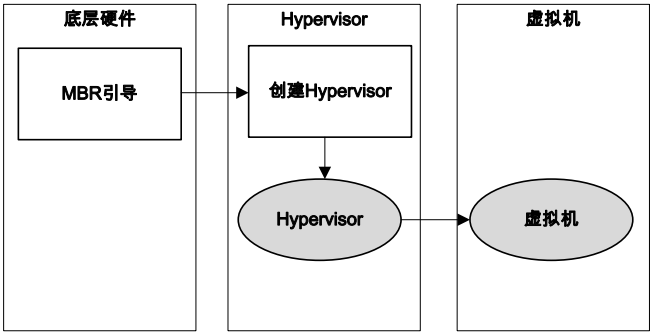

图 2.4 UVM 中虚拟机的启动过程

启动过程如图 2.4，这种虚拟机的设计目标在于：不需要在 Guest OS 中安装任何支持驱动。换句话说，Hypervisor 对于 Guest OS 完全透明，从而实现完全虚拟化（Full Virtualization）。这种方式的缺点是：Hypervisor 可能实现会很笨重，因而虚拟化效率不高，也会影响到系统安全，虚拟机的配置和管理可能也不易呈现给用户。

Hypervisor 的使用架构

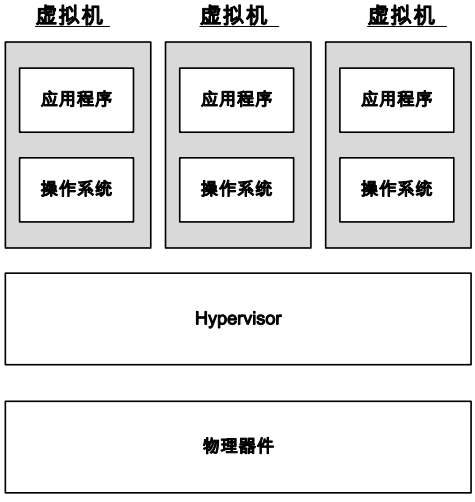

图 2.5：Hypervisor 使用架构图

前文中已经提过,Hypervisor 层的权限要高于等于操作系统的权限。操作系统的内核态已经处在了 Ring0 特权级上，因此 Hypervisor 层实际上要运行在一个新的特权级别上，我们称之为“Ring -1”特权级。同时需要新的指令，寄存器以及标志位去实现这个新增特权级的功

能。

作为一种最佳实践方案，一般 Hypervisor 层的实现都是越简单越好。一方面，简单的实现能够尽量降低花在 Hypervisor 上的开销[1]，毕竟大多数这些开销在原先的操作系统上是不存在的。另一方面，复杂的程序实现容易引入程序漏洞，Hypervisor 也是如此，且一旦 Hypervisor 中的漏洞被恶意使用，由于其所处特权级高于操作系统，将使隐藏在其中的病毒、恶意程序很难被查出。

批注 [S1]: 后面要描述 Hypervisor 的开销问题

Virtualization.pdf 10

HEV 技术所带来的性能损耗

新技术在使得开发人员的世界变得更加美好的同时，也不可避免的带来性能上的冲击。HEV 技术中，性能损耗最大的地方在于 Hypervisor 的引入及其所造成的需要进出 Hypervisor。一个最简单的例子，在普通的 x86 保护模式下，运行时刻执行到 CPUID 指令时，处理器会根据 EAX（RAX）寄存器的值直接读取 MSR 寄存器，并把结果写到 EAX~EDX（RAX~RDX）寄存器。但是在 Guest 模式下并且设置对 CPUID 指令进行拦截，那么每当 Guest OS 执行到 CPUID 指令时，处理器都会产生#VMEXIT 事件，从而陷入 Hypervisor 中对该指令进行相应处理，这个过程中涉及到 Guest 模式寄存器的保存，Host 模式寄存器的恢复，填充 VMCS/VMCB 中相应的内容（都是一系列处理器自动完成的内存操作），然后 Hypervisor 中不可缺少的有 CPUID 指令陷入的处理，最后在 Hypervisor 处理完后，处理器要回到 Guest 模式，这又涉及到 Host 当前寄存器的保存，Guest 模式寄存器的恢复，以及 VMCS/VMCB 中相应内容的填充。显然，花费在这上面的指令周期数将是保护模式下 CPUID 一条汇编指令所消耗的指令周期数的成千上万倍以上。

而在更一般的情况下，下列四种产生#VMEXIT 事件的情况都是需要处理的：

1. 访问特权级别的 CPU 的状态（Access to Privileged CPU State）
2. 中断虚拟化（Interrupt Virtualization）
3. 页表虚拟化（Page-Table Virtualization）
4. IO 设备虚拟化（IO-device virtualization）

[1] 关于 Hypervisor 的开销问题，后面的章节会有介绍

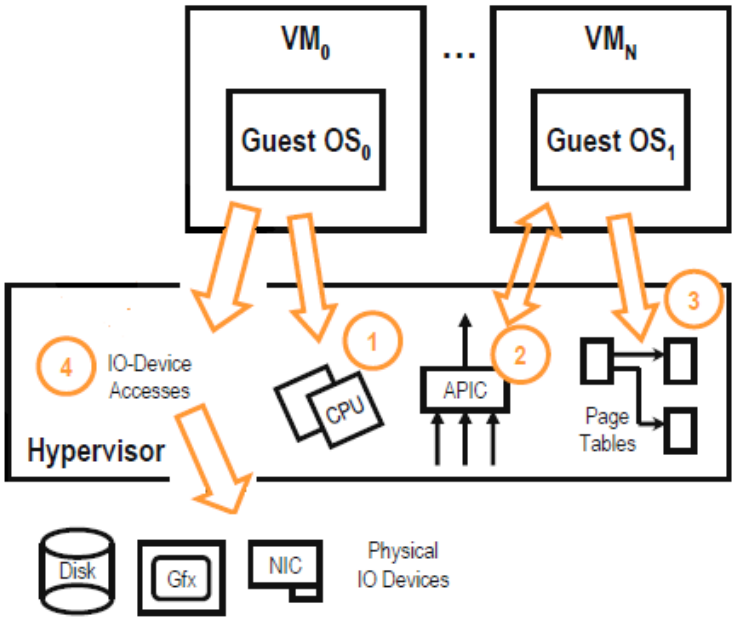

图 2.6 一般情况下产生#VMEXIT 事件的情况[1]

所有这些虚拟化都带来了如下方面的性能损失：

1. 进出 Hypervisor 的额外时间
2. 处理函数的执行时间
3. TLB/Cache Miss 造成的时间损失

最后，对那些无条件产生#VMEXIT 事件的指令的处理也给开发人员带来一些开发时间的损失，不过还好这部分时间通常不多。

已有的 HEV 技术平台介绍

现今两大主要硬件厂商 Intel 和 AMD 均以推出了支持硬件虚拟化技术的产品，两者大体功能和实现方法近似（意料之中，因为两家公司在过去你死我活的市场拼斗中，每次也都是实现功能和方法类似，只不过名字不同罢了）。下面我们简略介绍下这两家公司的支持 HEV 技术的平台，读者可以首先对这两种平台有概念。后面的章节中会有对这两个平台技术细节的更详细描述。

[1] 此图摘自 Intel® Virtualization Technology Processor Virtualization Extensions and Intel®Trusted execution Technology, Gideon Gerzon

1. AMD-V

概述

AMD 芯片支持硬件虚拟化的技术被称作 AMD-V（在技术文档中也被称为 SVM,其全称是 AMD Secure Virtual Machine）。其主要是通过一组能够影响到 Hypervisor 和 Guest Machine（客户机，下文简称 Guest）的中断实现的。AMD-V 技术设计目标如下：[2]

- 引入客户机模式（Guest Mode）[1]
- Hypervisor 和 Guest 之间的快速切换
- 中断 Guest 中特定的指令或事件(events)
- DMA 访问保护
- 中断处理上的辅助并对虚拟中断（virtual interrupt）提供支持
- 新的嵌套页表用来实现地址翻译
- 一个新的 TLB（其实就是一个 Cache）来减少虚拟化造成的性能下降。
- 对系统安全的支持

新的客户机模式 通过 VMRUN 指令即可进入这种新的处理器模式，当进入客户机模式后，为了辅助虚拟化过程，一些 x86 汇编指令的语义会发生变化。

外部访问保护 过去客户机（Guest）可以直接访问选定的 I/O 设备。现在硬件上已经实现这样的安全功能，能够阻止某个虚拟机拥有的某个设备访问其它虚拟机的内存。

中断上的支持 为了辅助中断的虚拟化，下列各项现在已经得到硬件支持，并且可以通过配置 VMCB 结构体[2]的方法使用

1) 拦截物理中断分发（Intercepting physical interrupt delivery） 发生在物理硬件上的中断能够让虚拟机发生一个中断，陷入 Hypervisor，从而使得 Hypervisor 可以首先处理这个中断
2) 虚中断（Virtual Interrupts） Hypervisor 能够将将提供给客户机（Guest）一套虚拟的中断机制。它是这样实现的，Hypervisor 会给这个客户机复制出来一份 EFLAGS.IF 用做中断屏蔽位（Interrupt Mask Bit），同时复制 APIC[3]中的中断优先级寄存器提供给客户机，从而客户机就会去操纵这套假的中断机制，而不是直接去操纵物理中断。
3) 共享物理 APIC AMD 的 SVM 技术能够允许多个 Guest 共享同一物理 APIC，同时又能保护这个 APIC 以免某个客户机不慎或恶意的在未经其它客户机许可的情况下，将可接收中断优先级设置为高优先级，从而清空了所有其它 Guest 的中断。

被标记的 TLB（Tagged TLB） 为了降低 Guest 模式和 VMM 模式切换开销，，TLB 上新加了一个 ASID 标记（Address Space Identifier），这个标记可以区分 TLB 上的一块地址是 Hypervisor 范围内的地址还是 Guest 的地址，从而加速了地址翻译。

批注 [S2]: 后面的章节要详细介绍 VMCB 结构体

[1] x86 上原有处理器模式包括保护模式(Protected Mode)，管理模式(SMM)，实模式(Real Mode)
[2] VMCB 结构体, Virtual Machine Control Block，也称 VMCB 控制块，Intel 的相应结构体名称为 Virtual Machine Control Sector, VMCS。这个控制块用于通知物理 Processor 要拦截的事件，以及在进出 Hypervisor 上下文切换时保存 Hypervisor 和 Guest 的各项寄存器，后面的章节中会有对这个结构体的详细介绍
[3] APIC, Advanced Programmable Interrupt Controller，高级可编程中断控制器，第三章有关于此主题内容。

安全方面的支持 现在提供的安全方面的支持主要是利用和 TPM 模块（Trusted Platform Module）[1]的交互，基于与安全 Hash 值的比较。

批注 [S3]: 第 17 章 其它安全技术写些有关 TPM 技术的介绍

新的地址翻译机制

AMD 引入了新的地址翻译技术——嵌套页表翻译（Nested Page Table，NPT），用于支持两级地址翻译，这样就使得虚拟机管理器不用再自己软件维护一套影子页表[2]

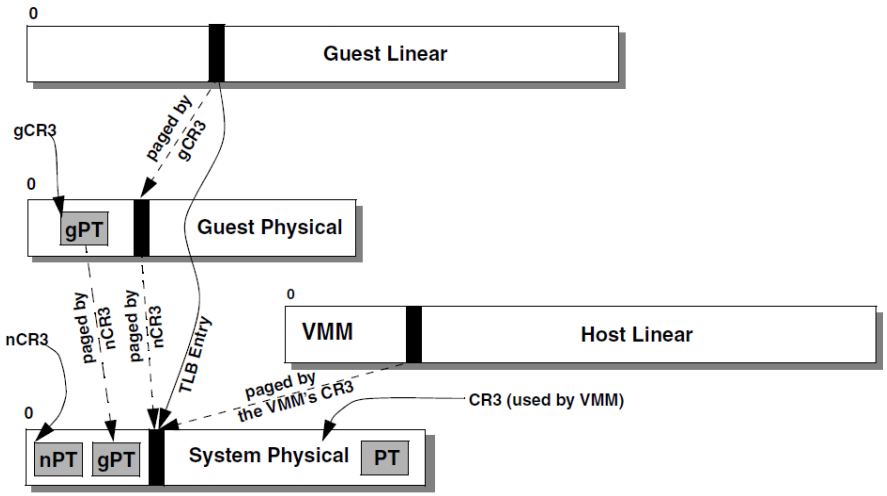

图 2.7 嵌套页表翻译地址过程[3]

嵌套页表翻译地址的过程如图 2.7 所示，这套机制的实现允许了从 Guest 线性地址到真实物理地址的翻译，也允许了在 Hypervisor 范围内的 Host 线性地址到真实物理地址的翻译。同时专门附加了一个 TLB 寄存器，用于缓存从 Guest 线性地址到真实物理地址的映射，从而提升了虚拟机的运行性能。

Note 关于嵌套页表技术的详细解释会在“第四章 深入 HEV 技术细节”一章中介绍，完整的描述请参考 AMD64 Architecture Programmer’s Manual, Volume 2: System Programming

相关结构和汇编指令

在 SVM 中，VM 控制块被称为 VMCB（Virtual Machine Control Blocks），其信息主要分为

[1] TPM 技术会在“第 17 章 其它安全技术”一章中介绍。
[2] 影子页表（Shadow Page Tables），常见于过去传统的虚拟机管理器中，由于存在从 Guest 线性地址到 Guest 物理地址，从 Guest 物理地址到真实物理地址两层地址翻译，所以在过去一般是要虚拟机软件自己维护两套页表去做这样的地址翻译。
[3] 此图摘自 AMD64 Architecture Programmer’s Manual, Volume 2: System Programming

两块，第一块是控制信息存储部分，同时也包含是否允许拦截某特定异常的遮罩（interception enable mask），Guest 中不同的指令和事件都能以修改 VMCB 中相应控制位的方法拦截，SVM 支持的两类主要的拦截是异常拦截和指令拦截，第二部分则是 guest 的状态信息保存，这里会保存段寄存器以及大部分的虚拟内存的入口控制寄存器，不过浮点寄存器信息不会被保存。需要注意的是 VMCB 在不同的处理器间不共享，并且 VMCB 一定要保证是在 4K 页对齐的连续物理内存空间中。

SVM 中主要的指令有以下这些：

- VMLOAD 从VMCB加载guest的状态，VMCB与guest是有对应关系的。
- VMMCALL 通过该方法guest可以与VMM显式的交流，方法是利用生成#VMEXIT从guest层退到VM层。
- VMRUN 加载VMCB，并开始执行guest层的指令，VMCB的物理地址将通过RAX获得，这个VMCB对应于要执行的guest
- VMSAVE 存储处理器状态的子集到VMCB中，这个VMCB的物理地址由RAX寄存器给出。
- STGI 用于设置全局中断标志（Global Interruption Flag）为1，这个指令属于Secure Virtual Machine。
- CLGI 用于设置全局中断标志（Global Interruption Flag）为0，同样这个指令属于Secure Virtual Machine。
- INVLPGA 使得TLB上一个ASID和一个虚拟页(Virtual Page)之间的映射关系无效，这个指令属于Secure Virtual Machine。
- SKINIT 安全的重新初始化CPU，使得CPU可以开始执行一段受信任的程序（trusted software）其方法是将该代码进行安全的哈希比较(secure hash comparison)。这也就是的开发者可以开发一个更安全的VMM loader。这种安全手段可以在TPM的帮助下发挥更大作用
- 改进的MOV指令 现在的MOV指令可以直接读写CR8寄存器（任务优先级寄存器 Task Priority Register），因此可以用来提高SVM应用的性能。

基于 AMD-V 的 Hypervisor 开发逻辑

其实由上文的描述可以看出，开发基于AMD-V的Hypervisor最主要是编写一个循环，这个循环要包含VMRUN命令以便从VM层启动一个Guest虚拟机，也要包含一段程序用于处理当#VMEXIT发生后的异常情况，这其中可能要手动做一些必要的保存现场和恢复现场的工作，具体造成异常的起因等均可通过读取VMCB中的数据获得。不过SVM没有提供一个显示终止Hypervisor的指令，因此若有需要，则要用其它方法关闭Hypervisor。NewBluePill中对SVM的支持就是这样实现的，我们会在深入探究NewBluePill的章节中详细展示怎样使用这些指令。

批注 [S4]: 后面要介绍 NBP 中关于 SVM 技术的运用

2. Intel-VTx

概述

Intel 芯片支持硬件虚拟化的技术被称为 Intel VT 技术（Intel ® Virtualization Technology）。

与 SVM 一样，其主要也是通过一组能够影响到 Hypervisor 和 Guest Machine 的中断实现的。

在 VT 技术中，与 SVM 类似的，设计架构上同样存在两种角色——虚拟机管理器（Virtual Machine Monitors，VMM）和客户机（Guest），两者分处在 VMX root 模式和 VMX non-root 两种模式下。VT 技术的设计目标是：

对于 VMM 层：（进入此层则代表进入了 VMX root 模式）

- 为每个虚拟机提供虚拟处理器，并且可以在恰当的时候把它放在真正的物理处理器上，从而使得这个虚拟处理器可以处理指令。
- VMM 层可以控制处理器资源，物理内存，管理中断和 I/O 操作

对于 Guest Machine：（进入此层则代表进入了 VMX non-root 模式）

- 每个虚拟机使用相同的接口来使用虚拟处理器，内存，存储设备等资源
- 每个虚拟机可以独立的不受干扰的运行，虚拟机间都是相互独立的
- 对于虚拟机来说，VMM 层像是完全透明的。

在 VT 技术下的 Hypervisor 生命周期如图 2.8 所示：

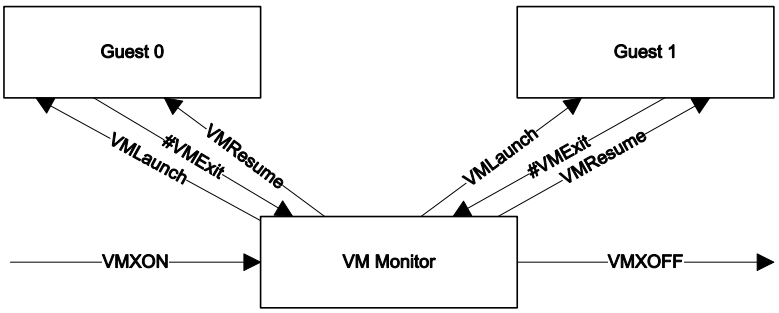

图 2.8 VT 技术中 Hypervisor 的生命周期

图示表明，软件通过执行 VMXON 指令进入 VMX Root 模式下，开启了虚拟机管理器的运行环境。然后通过使用 VMLaunch 指令使得目标系统正式运行在虚拟机中。当某条指令产生了#VMEXIT 事件后，会陷入虚拟机管理器中，待其处理完这个事件，可以通过 VMXResume 指令将控制权移交回发生#VMEXIT 事件的虚拟机。直到某个时刻，在 Hypervisor 中显示的调用了 VMXOFF 指令，Hypervisor 才会被关闭。

新的地址翻译机制

批注 [S5]: 仔细看完 EPT 技术后补充这一部分

Intel 同样引入了新的地址翻译技术——扩展页表翻译（Extended Page Table，EPT），用于支持两级地址翻译。

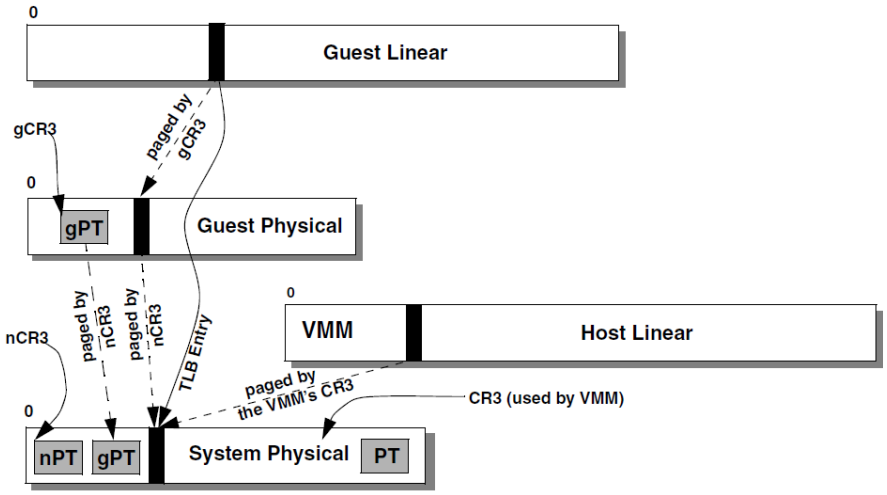

图 2.8 嵌套页表翻译地址过程[1]

嵌套页表翻译地址的过程如图 2.7 所示，这套机制的实现允许了从 Guest 线性地址到真实物理地址的翻译，也允许了在 Hypervisor 范围内的 Host 线性地址到真实物理地址的翻译。同时专门附加了一个 TLB 寄存器，用于缓存从 Guest 线性地址到真实物理地址的映射，从而提升了虚拟机的运行性能。

Note 关于扩展页表技术的详细解释会在“第四章 深入 HEV 技术细节”一章中介绍，完整的描述请参考 Intel 64 and IA-32 Architectures Software Developer’s Manual Volume 3B, Chapter 24. Support for Address Translation

相关结构和汇编指令

在 VT 技术中，VM 控制块被称为 VMCS（Virtual Machine Control Structure）。VMCS 包括三个组成部分：

表 2.1 VMCS 区域的组成部分

Byte 偏移量	内容
0	VMCS 版本标志（Revision Identifier）
4	VMX 退出原因指示器（VMX-abort indicator）[2]
8	VMCS 数据区

批注 [S6]: 后文会详细介绍 VMX Abort

如表 2.1 所示，VMCS 区域的前四个字节用于 VMCS 版本标志，不同的 VMCS 格式对应的版本号也不同，而这个物理处理器可以加载的 VMCS 结构体的版本号会存储在 MSR 寄存器中，因此这样的设计也就给未来发展留下了空间。

[1] 此图摘自 AMD64 Architecture Programmer’s Manual, Volume 2: System Programming
[2] 如果在 VM Exit 的时候遇到问题，就会发生 VMX Abort，一旦发生，那么这个逻辑处理器会进入关闭状态（Shutdown State）

在 VMCS 数据区中，主要有如下几个组成部分：

表 2.2 VMCS 数据区主要组成部分

名称	作用
虚拟机状态保存区（Guest State Area）	当发生了#VMEXIT 事件时虚拟机当前状态保存于此，在重新进入虚拟机的时候再利用此处的数据恢复虚拟机的状态
宿主机状态保存区（Host State Area）	当发生了#VMEXIT 事件时宿主机的状态利用此处数据恢复
虚拟机运行控制域（VM Execution Control Fields）	此处数据定义了虚拟机在什么情况下发生#VMEXIT 事件，对 VMX non-root 模式有影响
VMEXIT 行为控制域（VM Exit Control Fields）	此处数据定义了在#VMEXIT 事件发生时要做的附加工作（比如保存调试寄存器，加载全局性能控制寄存器等等这些工作）
VMEntry 行为控制域（VM Entry Control Fields）	此处数据定义了在发生#VMEntry 事件时（通常是因为调用了 VMResume 汇编指令）要做的附加工作。
VMEXIT 相关信息域（VM Exit Information Fields）	此处数据在发生#VMEXIT 事件时自动记录了发生原因和该事件的具体种类。这个域是只读的

VMX Abort 和 VMCS 数据区结构和用法会在后续章节中详细介绍。

批注 [S7]: VMX Abort 和 VMCS 数据区结构和用法会在后续章节中详细介绍

VT 技术在设计时注明，没有任何标志位用于区分一个逻辑处理器（Logical Processor）当前正在执行 VMX root 模式下的指令还是执行的 VMX non-root 模式下的指令，这就确保了 Hypervisor 对虚拟机完全透明——因为虚拟机无从判断它当前是否运行在一个虚拟机下。最后要注意的是 VMCS 同样要求保证是在 4K 页对齐的连续物理内存空间中。

VT 中主要的指令有以下这些：

维护VMCS结构体的指令

- VMPTRLD 该指令用来激活一块VMCS。修改该处理器的当前VMCS指针（Current-VMCS Pointer）指向传入的VMCS物理地址，并且激活该VMCS，如果要维护一块VMCS则必须先激活该VMCS。（否则不能用这些指令来维护）
- VMPTRST 用来存储当前VMCS指针到指定位置。
- VMCLEAR 该指令用来使一块VMCS变为不活跃状态。该指令将标记为已启动状态（Launch State）的VMCS设置为不活跃状态（Inactive State/Clear State）并且更新该VMCS块所有区域信息并确保写入VMCS块内存中（这也就把对应虚拟机和Hypervisor的最新信息同时写入到VMCS块中），如果带操作的VMCS块就是当前VMCS指针所指向的VMCS块，那么该指针会被设置为无效地址
- VMREAD 通过指定的VMCS Encoding从当前VMCS块中读取一个参数。
- VMWRITE 通过指定的VMCS Encoding从当前VMCS块中写入一个参数。

与虚拟机管理器有关的指令

- VMCALL 这条指令用于Guest和Hypervisor进行通信。执行该汇编指令会产生一个#VMEXIT事件，从而使得可以陷入Hypervisor中。
- VMLAUNCH 这条指令用于启动当前VMCS指针所指的一个虚拟机，并且移交控制权给Guest。
- VMRESUME 这条指令用于从Hypervisor中恢复虚拟机的执行，并且移交控制权给Guest。
- VMXOFF 这条指令用于关闭Hypervisor。在下次执行VMXON开启Hypervisor前不得执行虚拟机相关汇编指令。

- VMXON 这条指令用于处理器进入VMX模式下，执行该指令后也就可以运行Hypervisor。传入的参数必须是4K页对齐的物理地址，这段内存用于支持后续VMX相关的操作。

VMLAUNCH 和 VMRESUME 指令的异同

VMLAUNCH 和 VMRESUME 命令都是将控制权移交到虚拟机，那么两者的区别呢？

两者运行的时机不同!

1. VMLAUNCH 指令会检查当前 VMCS 的启动状态是不是不活跃状态（相应标记位清空）。成功运行结果是该 VMCS 被标记为已启动状态。
2. VMRESUME 指令会检查当前 VMCS 的启动状态是不是已启动状态

所以，必须利用 VMLAUNCH 指令启动一个虚拟机。以后的某个时候，因为 VMEXIT 事件而陷入 Hypervisor 中，这个时候要恢复虚拟机的运行则要利用 VMRESUME 指令，正如图 2.8 所示。

管理VT相关的TLB的控制指令

- INVEPT 这条指令用于EPT地址翻译中，使TLB中缓存的地址映射失效
- INVVPID 这条指令用于在TLB中使某个VPID对应的地址映射失效

基于 VT 的 Hypervisor 开发逻辑

利用 VT 技术开发 Hypervisor 的过程不同于利用 SVM 技术的开发过程，最主要的差别是在 VT 技术中，Guest 和 Hypervisor 下面要运行的 IP 地址是可以在 VMCS 中设置的，同时 Hypervisor 就是用于处理 VMEXIT 事件，因此就像现代操作系统为系统调用设置一个统一入口，并将入口地址存入 MSR 寄存器一样，在 VT 中，通常也将 Hypervisor 的这个入口 IP 设置为事件处理函数入口地址（Event Dispatcher Address）。在事件处理的最后，又通过一个 VMXRESUME 指令统一的返回到 Guest 的指令执行流程中。对于事件发生信息，同样可以通过读取 VMCB 中相应数据获得。NewBluePill 中也有对 VT 技术的支持，我们同样会在深入探究 NewBluePill 的章节中详细展示怎样使用这些指令。

批注 [S8]: 后面要介绍 NBP 中关于 VT 技术的运用

3. Intel-VTd(如果时间充裕则写)

SVM 和 VT 不同之处和使用时应注意的地方

通过前文的描述，看上去 SVM 技术和 VT 技术十分相似，但是实际上两者还是有一些不同之处。在开发过程中必须注意到这些不同之处，它们是正确并且高效实现 Hypervisor 的关键。

SVM 的开发逻辑中，VMRUN 和事件处理程序要处于同一循环中，这是因为 Hypervisor 的事件处理程序入口在 VMRUN 的下一条地址上，而在 VT 技术中，由于可以自由指定这个

入口地址，因此可以在 VMCS 块中指定一个函数作为事件处理入口函数。

SVM 采用 ASID 作为 TLB 中 Guest 和 Hypervisor 地址的标记，而 VT 采用 VPIDs（Virtual-Processor Identifiers）作为 TLB 中不同虚拟机地址翻译的缓存标记，因此 VT 技术的缓存策略更精细所以更好些。

虽然 SVM 技术和 VT 技术都可以管理中断，管理资源，访问控制，但两者具体处理行为有一些差别，VT 技术将指令分为三种：无条件陷入的指令，有条件陷入的指令和不产生陷入的指令/事件。SVM 技术则分为了异常拦截和指令拦截，其中一些异常虽然会造成陷入，但是同时也会自动标记相应的异常寄存器（Exception Specific Registers）。应用的时候一定要根据手册上的描述给出相应的实现。

SVM 和 VT 技术在使用时一定要注意，#VMEXIT 事件的产生来源于异常而不是行为，比如用户可以拦截 RDMSR 指令，但是发生的 sysenter 指令却不能拦截到，是因为在这种情况下，虽然 sysenter 有读取 MSR 寄存器的操作，但是因为没有提前在 VMCS/VMCB 中设定处理器遇到 sysenter 产生异常，所以处理器执行到 sysenter 指令当然也就不会产生#VMEXIT 事件。

NewBluePill 项目介绍

NewBluePill 项目（http://www.bluepillproject.org/）诞生于 2007 年，由 Invisible Things Lab 开发，现在对外公开的版本是 nbp-0.32-public 版本。该项目从 2007 年第三季度以来得到了 Phoenix 公司的大力支持。[1]

该项目目的是开发这样一种恶意软件：

- 不用已有的方法的隐藏自己
- 即使它的隐藏方法众所周知，其它软件也不能探测到它
- 即使它的实现代码众所周知，其它软件也不能探测到它

可以看出，该目的与非对称密码的设计目的有异曲同工之妙。该项目通过发掘 VT 技术和 SVM 技术平台漏洞来实现，当前在公开版本上已经实现的功能包括：

- 支持 SVM 和 VT-x 技术
- 在操作系统运行时刻动态加载和卸载
- 在 AMD 平台上支持嵌套 Hypervisor
- 一套自己的页表，用于实现内存隐藏
- 反 Hypervisor 探测技术（Anti-Hypervisor Detector）：RDTSC 欺骗
- 反 Hypervisor 探测技术（Anti-Hypervisor Detector）：组织可信时间源的检测（Blue-Chicken 技术）

在其未公开的版本上，实现的功能包括：

- 在 Intel VT-x 平台上实现嵌套 Hypervisor

[1] Phoenix 公司的虚拟化技术产品 HyperSpace，具体信息可以参考网上资料。与之类似的还有华硕公司（Asus Inc.）的 Instant On 技术

```
*
* This software is protected by domestic and International
* copyright laws. Any unauthorized use (including publishing and
* distribution) of this software requires a valid license * from the copyright holder.
*
* This software has been provided for the educational use
* only during the Black Hat training and conference. This
* software should not be used on production systems.
*
*/
```

（This page is intended to be blank）