KVM 虚拟化技术中处理器隔离的实现①

黄 煜，罗省贤

(成都理工大学 信息科学与技术学院，成都 610059)

摘 要：KVM 是基于 Intel VT 技术并结合 QEMU 来提供设备虚拟化的虚拟机。通过分析和研究 KVM 虚拟机的创建、运行机制，从进程控制的角度对 KVM 虚拟机进行隔离，实现了 KVM 虚拟机在创建、运行时都保持在指定的核上运行，达到最大化利用处理器资源的目的。

关键词：KVM；虚拟化；处理器隔离

Implementation of Processor Isolation Under the KVM Virtual Technology

HUANG Yu, LUO Sheng-Xian

(School of Information Science and Technology, Chengdu University of Technology, Chengdu 610059, China)

Abstract: KVM (Kernel-based Virtual Machine) is a virtual machine based on Intel VT technology and works with QEMU, which provides device virtualization. From the process point of view, this paper focuses on the analysis of the initiative and the running mechanism of KVM virtual machine, based on the analysis, a method of isolating KVM virtual machine is implemented, which makes the KVM virtual machine run on the specified core whenever it's created or running, and achieves the maximized utilization of processor resources.

Key words: KVM; virtualization; CPU isolation

随着计算机硬件的发展，人们对最大化利用硬件资源的需求日益迫切。从上世纪六、七十年代虚拟机概念的提出，到现在虚拟化技术的日益成熟，为人们这些需求的实现提供了有利的解决方案。基于 Intel VT[1]技术的 KVM[2]虚拟机，是一种采用硬件辅助虚拟化的全虚拟化方案，并在 Linux 内核版本 2.6.20 之后，以模块的形式集成到内核的各个主要发行版本。KVM 虚拟机吸收了 QEMU[3]、Bochs、UML、Virtual PC 等传统虚拟机的长处和优势[4]，利用硬件辅助的虚拟化技术，使虚拟机的大多数指令可以直接在物理处理器上运行，具有更加优越的效率和性能。为了充分利用处理器的资源，往往需要虚拟机在某一指定个核上运行，而其它的核上运行特定的任务。本文基于 CPU 的亲和性(Affinity)，采用进程绑定，实现了一种解决上述问题的有效方法。

1 Intel VT技术下的KVM虚拟机架构

KVM 虚拟机由两部分构成：一部分是以模块形式集成到 Linux 内核的 KVM Driver[5]，负责管理虚拟硬件资源，并创建一个字符设备（/dev/kvm），通过 ioctl 接口与用户空间的设备通信，以及完成虚拟机内存的分配、虚拟机寄存器的读写以及虚拟机 CPU 的运行等。因此，每个虚拟机被 Linux 核心当作是一个标准的进程。另一个部分是 KQEMU，即针对有虚拟化支持的 X86 处理器而修改过的 QEMU。KQEMU 主要是完成对硬件的模拟，通过/dev/kvm 与 KVM Driver 交互。KVM 虚拟机架构如图 1 所示：

在 Intel VT-x 技术下有两种工作模式：VMX root operation 模式[1]和 VMX non-root operation 模式[1]。在这两种模式下，KVM 虚拟机充分利用硬件的支持，实现了系统的虚拟化。KVM 的 KVM Driver 和 KQEMU 分别运行于 Kernel 模式的和 User 模式。这里的 Kernel

① 收稿时间:2011-05-03;收到修改稿时间:2011-07-16

模式和 User 模式实际上指的是 VMX root operation 模式下的特权级 0 和特权级 3。在图 1 中，Linux Kernel、KVM Driver 和/dev/kvm 都属于 VMX root operation 模式。另外，KVM 将客户机所在的运行模式称为 Guest 模式，即 VMX 的 VMX non-root operation 模式。

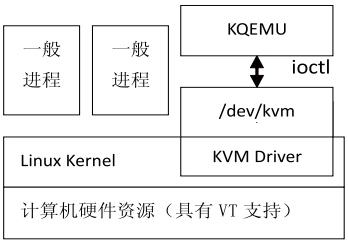

图 1　KVM 虚拟机架构

2　KQEMU和KVM工作原理

Intel VT-x 是与处理器管理相关的硬件虚拟化技术[6]。在 VT-x 的支持下，KVM 中的每个虚拟机可具有多个虚拟处理器 VCPU，每个 VCPU 对应一个 KQEMU 的线程，这个 KQEMU 线程代表了 KVM 虚拟机的运行。没有硬件支持的纯 QEMU 虚拟机和利用硬件支持实现虚拟化的 KVM 虚拟机的差异也就在这里。为了更加形象地说明 KVM 和 KQEMU 间的关系，本文称它为 KVM 线程，而将创建客户机的应用程序 KQEMU 称为 KQEMU 进程。KQEMU 应用程序在创建或者说在加载一个 Guest OS 后，就转换为 KVM 线程运行。VCPU 的创建、初始化、运行以及退出处理都在 KVM 线程上下文中进行，需要 Kernel、User 和 Guest 三种模式相互配合，其工作模型如图 2 所示：

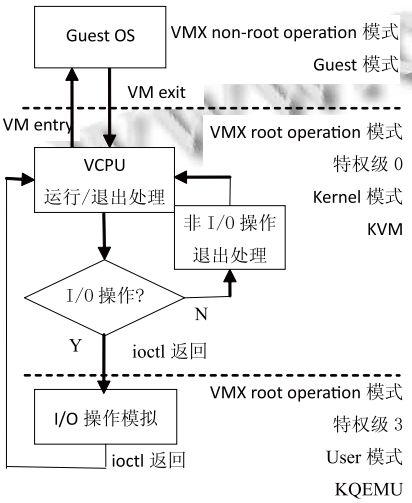

图 2　KVM 工作模型

从图 2 可看出，KVM 线程以 ioctl 的方式与 KVM 内核模块间进行交互，指示 KVM 内核模块进行 VCPU 的创建和初始化等操作，而 KVM 内核模块与客户机之间通过 VM Exit 和 VM entry 操作进行切换。

初始化工作完成之后，KVM 线程以 ioctl 的方式向 KVM 内核模块发出运行 VCPU 的指示，后者执行虚拟机进入操作（VM entry），将处理器由 Kernel 模式切换到 Guest 模式，转而运行客户机。但此时仍处于 KVM 线程上下文中，且正在执行 ioctl 系统调用的 kernel 模式处理程序。

客户机在运行过程中，如发生异常或外部中断等事件，或执行 I/O 操作，可能导致虚拟机退出操作（VM exit），将处理器状态由 Guest 模式切换回 Kernel 模式。KVM 内核模块检查发生 VM exit 的原因，如果 VM exit 由 I/O 操作导致，则执行系统调用返回操作，将 I/O 操作交给处于 User 模式的 KQEMU 进程来处理，这时 KQEMU 会创建一个新的 KQEMU 线程来完成这些 I/O 处理，之后再次执行 ioctl，指示 KVM 将处理器切换到 Guest 模式，恢复客户机的运行；如果 VM exit 由其它原因导致，则由 KVM 内核模块负责处理，并在处理后切换处理器到 Guest 模式，恢复客户机的运行。

由以上分析可知，KVM 虚拟化技术中物理 CPU 核的隔离实现，与 KQEMU 应用程序、KVM 线程、异步 IO 事件处理这三方面有着密切的联系。

3　虚拟机进程隔离实现

3.1 隔离方案设计

在 KVM 虚拟机运行的过程中，与虚拟机处理器紧密相关的重要数据结构 VMCS[1]保存在内存中，包含了虚拟 CPU 的相关寄存器的内容和虚拟 CPU 相关的控制信息，每个 VMCS 对应一个虚拟 CPU。但 VMCS 具有“迁移性”[7]，使用时需要与物理 CPU 绑定。例如，某个 VMCS 先和物理 CPU0 绑定，并在某个时刻解除绑定关系，下一个时刻可能会重新绑定到物理 CPU1 上。在任意给定时刻，VMCS 与物理 CPU 是一对一的绑定关系，即一个 VMCS 只能与一个物理 CPU 绑定。

因此，多核条件下要实现 Guest OS 在一个虚拟的单核平台上运行，就需要考虑 KQEMU 应用程序、KVM 线程、异步 IO 事件处理与 CPU 的绑定关系。利用 CPU 的亲和性(Affinity)，通过设置这三方面的 Affinity 属性，可以将 Guest OS 进程绑定到指定的核

上运行，从而实现虚拟的单核平台。例如，在创建虚拟机时将两个 Guest OS 进程绑定到指定的 CPU 核上，使其在整个运行期间不发生迁移，如图 3 所示：

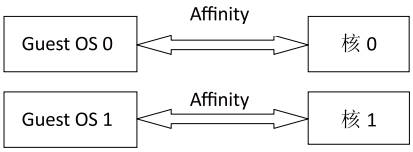

图 3 客户操作系统与 CPU 核的绑定

根据 KVM 原理和机制的分析，可以通过两个方案实现绑定操作：

方案一：直接通过内核函数设置 Guest OS 进程的属性，实现 Guest OS 保持运行在指定的核上。通过编写脚本文件，使创建虚拟机和绑定两项操作自动完成。

方案二：修改、优化 KQEMU 源代码，生成特定的 KVM 虚拟机。在创建客户机的时候实现绑定，使客户机只能在指定核上的运行，不发生迁移。

从两个方案的设计可知，方案一的主要是从 KQEMU 应用程序的应用出发的，它的实现必然要用到文件的读写和脚本的绑定，以及虚拟机管理器启动脚本的修改。方案二是基于 Qemu 源代码的修改，绑定时机明显早于方案一，实现过程也更简单明了。除此之外，方案一从理论上来说，对有异步 IO 时才产生的 aio 线程的绑定可能出现问题，所以本文最终选择方案二。

3.2 处理器隔离的实现方法

Linux 的线程在核内是以轻量级进程的形式存在的，而实现是在核外。它拥有独立的进程表项，其创建、删除等操作都是在核外 POSIX thread 库中进行。与 fork() 调用创建一个进程的方法不同，pthread_create()创建的线程并不具备与主线程同样的执行序列，而是使其运行 start_routine()函数。因此，用 fork 创建的父子进程之间的 Affinity 具有继承性，而调用 pthread_create()创建的父子进程之间则没有这种继承性，KQEMU 和 KVM 之间则是后者的情况。

因此，仅通过设置 KQEMU 的 Affinity 属性不能实现 KVM 虚拟机处理器的隔离。需分为两部分处理：①从启动 QEMU 的命令获取 cpuid 参数。这需要对 KQEMU 源代码进行修改，增加-cpuid 的命令选项，完成对参数的解析以及相关的绑定操作。主要涉及 KQEMUOption、KQEMU_options 等几个选项变量。②根据解析获得的 cpuid 参数，对 KQEMU 应用进程、KVM 线程、异步 IO 事件处理线程运用 Linux 内核的 Affinity 机制进行绑定，实现对处理器的隔离。图 4 是以双核处理器为例实现 KVM 虚拟机绑定的主要过程。

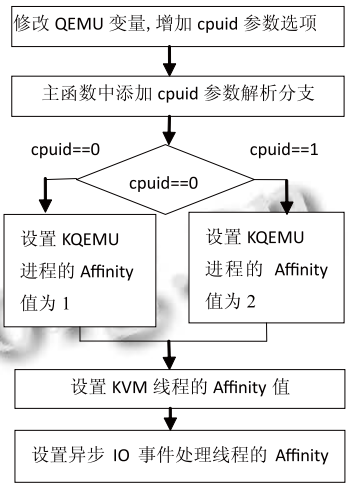

图 4 KVM 虚拟机进程绑定实现过程

4 实验结果与分析

由设计思路的分析可知，对方案二实现方法需要验证的是客户机的创建、客户机的运行，以及客户机的异步 IO 事件处理。本实验中使用的测试平台是：处理器是 Pentium(R) Dual-Core CPU E5300 2.60GHz，内存为 DDR2-800 2G，操作系统内核版本是 Linux-2.6.33.2。GuestOS 运行测试程序是计算 Fibonacci 前 35 项和，每计算一次休眠 15 毫秒。

执行命令 qemu-system-x86_64 Guestos.img –m 512–cpuid 1，在启动 Guest OS 时指定创建 Guest OS 进程的处理器参数 cpuid，使创建客户机的进程在指定的核 1 运行。用 top 命令的 p 和 H 参数，观察 KVM 虚拟机运行时的进程号和线程号。

t1 时刻（19：33：34），KQEMU 进程号为 3905，KVM 线程号为 3906，异步 IO 事件处理线程号为 3910，如图 5 所示：

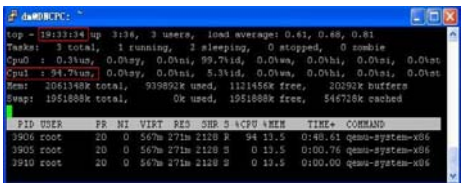

图 5 t1 时刻 KVM 虚拟机进程运行状态

t2 时刻（20：43：20），KQEMU 进程号为 3905，

KVM 线程号为 3906，异步 IO 事件处理线程号为 4212，如图 6 所示。此时图 4 中线程号为 3910 的 aio 线程消失，动态产生 aio 线程 4212。

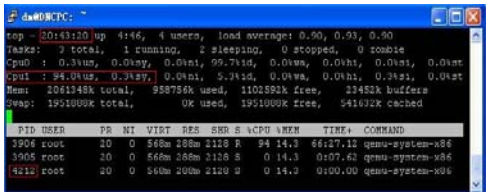

图 6　t2 时刻 KVM 虚拟机进程运行状态

根据 Linux 内核的 PID 命名规则可知，PID 逐渐递增，aio 线程的产生是一个动态过程，根据 Guest OS 的需要而产生。从而进一步证实，如果用采用方案一，则难以实现 aio 线程的绑定。

经过长时间的实验观察，虽然运行 Guest OS 时 CPU 核 1 的利用率达到 94.7%左右，KVM 虚拟机始终保持在指定的核 1 运行，即使核 0 十分空闲，仍不会发生迁移。可以通过 taskset 命令查看这三个进程的 Affinity 值，得到进一步的验证，如图 7 所示。图中 3905，3906，4212 进程的 Affinity 值为 2（即 0x00000010），表示相应进程或线程运行在核 1 上。

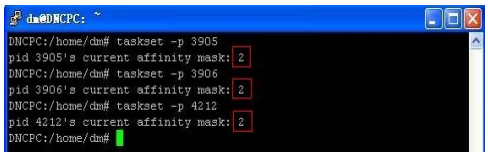

图 7　用 taskset 命令查看虚拟机进程的 Affinity 值

5　结论

本文通过对 KVM 虚拟机的绑定，实现了 KVM 虚拟机处理器核之间的隔离，使 KVM 虚拟机在指定的物理 CPU 核上运行，不发生核间的迁移，从而达到物理 CPU 资源的指定使用。当运行多个 Guest OS 时，只需在运行时指定 CPU 参数，即可让各个 Guest OS 始终运行在指定的核上，进而能够充分有效地利用计算机硬件资源[8]。

参考文献

1 The Intel®64 and IA–32 Architectures Software Developer's Manual.2009:10–110.
2 KVM 官方网址 http://www.linux-kvm.org/page/Main_Page.
3 QEMU 官方网址.http://wiki.qemu.org/Main_Page.
4 Smith JE, Nair R. Virtual machines. Publishing House of Electronics Industry, 2005: 3–10.
5 Qumranet Inc. KVM: Kernel-based Virtuallization Driver. 2006:1–5.http://www.linuxinsight.com/files/kvm_whitepaper.pdf.
6 李胜召,郝沁汾,肖利民.KVM 虚拟机分析.计算机工程与科学,2008,30(A1):129–130.
7 英特尔开源软件技术中心.复旦大学并行处理研究所.系统虚拟化.北京:清华大学出版社,2009.104–128.
8 Eddolls T. VM performance management McGraw-Hill Book Co., 1989:5–10.

(上接第 193 页)

参考文献

1 克鲁森.Rational 统一过程实践者指南.徐正生译.北京:中国电力出版社,2004.260.
2 谭云杰.Thinking in UML.北京:中国水利水电出版社,2009.15.
3 陈丽娟.一种用例驱动的应用系统分析与设计方法.计算机应用与软件,2002,19(4),42–44.
4 任兴来.基于用例的用户界面原型设计研究.青岛:青岛大学,2009.
5 孔军,等.基于 UML 的系统需求分析.计算机工程与应用,2003,15:217–218.
6 方红萍,陈和平.信息系统 UML 建模研究.计算机工程与设计,2006,27(19):3614.
7 陈鑫,李宣东.基于设计演算的形式化用例分析建模框架.软件学报,2008,19(10):2548–2549.