高技术通讯 2012 年 第22 卷 第11 期:1149 ~ 1154

doi:10.3772/j.issn.1002-0470.2012.11.007

系统虚拟化中指令去特权化的软硬件协同设计①

台运方②*,**,*** 蔡万伟*,**,*** 刘 奇*,**,**** 张 戈****,***** 章隆兵*,**,****

(*中国科学院计算机系统结构重点实验室 北京 100190)

(**中国科学院计算技术研究所 北京 100190)

(***中国科学院研究生院 北京 100049)

(****龙芯中科技术有限公司 北京 100190)

(*****中国科学院重庆绿色智能技术研究院电子所 重庆 401122)

摘 要 提出了一种软硬件协同设计的去特权化方式,用来减少系统虚拟机中特权指令和敏感指令产生的额外开销。其主要思想是使用修改操作系统源代码的软件去特权化方式减少敏感指令的额外开销,使用修改硬件方式减少非敏感指令的特权指令的额外开销。两者配合使用可最大限度减少虚拟机中这两类指令的额外开销,从而能提高系统虚拟机性能。在 MIPS 架构 CPU 的测试板上的实验显示,使用这种方法之后虚拟机的特权指令和敏感指令造成的异常数减少了近 97%,同时 SPEC CINT2000 测试集中大部分程序性能提升 100% 以上。

关键词 虚拟化,去特权化,软硬件协同设计,特权指令,敏感指令

0 引 言

系统虚拟化是指用虚拟化软件虚拟出虚拟机——具有完整硬件功能的计算机的技术。根据经典虚拟化的三个特征[1],虚拟机指令可分为敏感指令和无害指令两类。敏感指令也分为两类[2]:一类是控制敏感指令,指的是那些试图改变系统资源配置的指令,例如 x86 架构中访问外设的指令 out;另一类是行为敏感指令,指的是那些行为或运行结果依赖于系统资源配置的指令,例如 MIPS 架构通用 0 号协处理器访问指令 mfc0。敏感指令之外的指令都可称作无害指令,直接执行不会妨碍虚拟机监控程序(virtual machine monitor, VMM)的正常高效工作。然而敏感指令不一定就直接等同于特权指令,例如 MIPS 架构中的 cache 指令就是一种非敏感的特权指令。

特权指令和敏感指令的模拟是 CPU 虚拟化的瓶颈。特权指令和敏感指令会造成额外开销,降低这种开销的方法一般是减少这些指令造成的异常的个数,称为去特权化。主流的去特权化方式有多种,VMware 采用的是二进制翻译方式[3]:执行过程中用一系列非特权的无害指令代替特权指令及敏感指令[4-6]。Xen 使用了更简单的方式处理特权指令和敏感指令[7]:修改客户操作系统源代码,将操作系统中的这些指令用效果相同的非特权的无害指令替代。与上述两种软件去特权化方式不同,Intel 和 AMD 各自提出了自己的硬件辅助虚拟化结构[8,9]。硬件辅助虚拟化结构增加了额外的操作模式,在这种模式下大多数特权指令和敏感指令不会修改系统资源,也就不会产生异常。对于 MIPS、Alpha 等缺少硬件辅助虚拟化规范的架构,虚拟机需要使用软件去特权化方式提高性能。然而存在某些非敏感的特权指令例如 MIPS 架构中的 cache 指令,虚拟机无法使用二进制翻译、修改操作系统代码等软件去特权化方式进行优化。鉴于此,本文设计一种指令去特权化的软硬件协同设计方法:修改客户系统减少敏感指令;修改硬件使得非敏感的特权指令可在非特权态下运行,以降低额外开销。

① 国家"核高基"科技重大专项课题(2009ZX01028-002-003,2009ZX01029-001-003)和国家自然科学基金(60921002,61003064,61050002,61070025,61100163,61133004,61173001)资助项目。

② 男,1988 年生,博士生;研究方向:计算机系统结构,操作系统和虚拟化;联系人,E-mail: taiyunfang@ict.ac.cn

(收稿日期:2011-12-26)

1 方法的可行性分析

软硬件协同设计去特权化的主要思想是使用软件去特权化方式减少敏感指令(包括特权指令中是敏感指令的部分),修改硬件使得不属于敏感指令的特权指令能在非特权态下运行。软件去特权化由于得到广泛使用,其正确性可以保证。但是这种硬件去特权化是否正确则需要分析:这些不属于敏感指令的特权指令,即使可以在非特权态直接执行也不会影响到宿主系统的资源分配,不会对虚拟机的行为产生有害的影响。因此这种硬件去特权化的正确性可以保证。

然而特权指令在非特权态下执行会带来安全性问题。这种安全性问题可以通过如下虚拟机监控程序(VMM)设计解决。假设 CPU 拥有三个运行态:一个特权态(A)和两个非特权态(B 和 C)。VMM 可以将虚拟机的操作系统运行于一个非特权态 B,将虚拟机的用户程序运行于另一个非特权态 C 下。同时,VMM 还可以捕获虚拟机的所有的操作系统和用户程序切换,因为这些行为都属于敏感行为。如果 VMM 在虚拟机切到 B 运行态下时打开硬件去特权化(即使某些非敏感指令的特权指令能在非特权态下运行),在切换到 C 运行态下关闭硬件去特权化,那么只要保证在 B 运行态下程序无不安全的行为则虚拟机安全性可以保证。在 B 运行态下运行的是操作系统,其安全性可以轻易保证,因此硬件特权化的安全性可以保证。

以上分析表明,软硬件协同设计去特权化的正确性和安全性都可以得到保证,因此方法是可行的。但方法也有局限性:一是要求 CPU 有至少三个运行态;二是使用时需要保证客户操作系统安全性。下面介绍软硬件协同设计去特权化在 MIPS 架构的 CPU 上的实现及效果。

2 虚拟机性能瓶颈分析

在引言中提到,虚拟机执行指令的瓶颈主要在于特权指令和敏感指令的模拟。由于这些指令的额外开销主要体现在其造成的异常上,虚拟机运行过程中产生的异常个数可以很明显地表示出这两类指令造成的性能损耗。

以 MIPS 架构平台为例,本文首先在 MIPS 架构的 CPU 上实现一个陷入模拟[1,10] VMM(具体实现方式将在第 4 节中介绍)。MIPS 架构满足敏感指令是特权指令的子集,没有类似 x86 架构的漏洞[6],因此实验只需要关注特权指令即可。表 1 列出了在实验平台上 Linux 2.6.36 内核从开始运行到进入用户空间这段时间内的异常分布(异常数为 0 的异常未列出)。可以看到,特权指令所造成的协处理器不可用异常(coprocessor unusable,CpU)占了所有异常的很大一部分,大约 78%。剩余的大都是因为访存造成的旁路转换缓冲(TLB)缺失异常。可以说在虚拟机中特权指令的模拟是性能瓶颈。

表 1 异常分布

异常类型	异常数量(个)
中断	3651
TLB 修改	1
TLB 读	60
TLB 写	14
系统调用	1017
保留指令	44
协处理器	729395
溢出	1
TLB 缺失	204671
总计	938854

表 2 列出了各个特权指令造成的异常分布。可以看出,通用 0 号协处理器访问指令(mfc0,dmfc0,mtc0,dmtc0)占了绝大多数,除此之外 cache 指令占了绝大多数。因此 MIPS 架构上指令的去特权化主要针对通用 0 号协处理器访问指令和 cache 指令。

表 2 特权指令造成的异常分布

特权指令	异常数量(个)
mfc0	344538
dmfc0	2750
mtc0	316748
dmtc0	3791
tlbr	0
tlbwi	139
tlbwr	33
tlbp	107
eret	3518
cache	57768
其他指令	3

3 MIPS 架构上实现实例

由第 2 节中分析结果可以看到,指令去特权化

在 MIPS 架构上主要针对通用 0 号协处理器访问指令和 cache 指令使用。这两类指令的特点决定了我们可以分别使用修改操作系统代码方式和硬件去特权化方式去优化。

3.1 修改操作系统代码方式

这里使用的修改操作系统方式思想是将操作系统源代码中的特权指令用一段效果相同的代码替代。通用 0 号协处理器访问指令在虚拟机上的执行效果是访问 VMM 分配的一段空间。如果这段空间客户操作系统可以访问,那么访问这段空间的一系列指令可以替代通用 0 号协处理器访问指令。这便是优化通用 0 号协处理器访问指令的最基本思想。

因此,优化最关键步骤在于 VMM 分配的空间如何能让客户系统访问到。但是 VMM 空间是宿主机空间一部分,原则上不能被客户系统访问。为了解决这个矛盾,客户系统可以先分配虚拟 0 号协处理器空间,然后通过系统调用传给 VMM,让 VMM 获得其物理地址对应的内核虚拟地址去访问。这样便实现了一个简易的共享地址方式。流程如图 1 所示:VMM 先分配一段空间暂存虚拟 0 号协处理器内容;然后客户系统执行代码,等到分配完虚拟 0 号协处理器空间后通过系统调用将地址传递到 VMM;VMM 计算其物理地址获得内核态虚拟地址,拷贝暂存空间内容到共享空间;最后 VMM 用新空间替代以前的暂存空间,释放暂存空间。

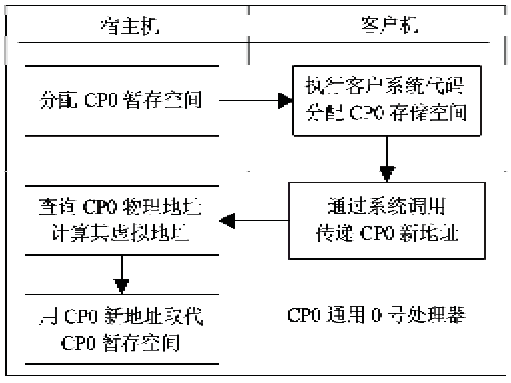

图 1 简易共享内存空间流程

在实现了简易共享空间方式后,操作系统中特权指令便可以被替换掉。以 mfc0 指令为例,其修改前后如图 2 所示。图中的 mfc0 指令是访问 0 号处理器的异常程序计数器(exception program counter, EPC)寄存器。paravirt _ cp0 是在客户系统中分配的虚拟 0 号协处理器地址。PV _ CP0 _ EPC 是 EPC 寄存器在虚拟 0 号协处理器的偏移。由于 mfc0 指令在虚拟机中执行效果也是访问虚拟 0 号协处理器,而且修改后的也是访问相同的地方,两段代码是等价的。虽然修改后由一条指令变为三条看起来开销多了,但由于没有模拟 mfc0 所需要的几百条保存和恢复现场指令,替换后的性能要远远好于之前的方式。

修改前	修改后

mfc0 k0, CP0_EPC	la paravirt_cp0
	addiu k0,PV_CP0_EPC
	lw k0,(k0)

图 2 mfc0 指令修改前后

修改操作系统代码可以减少大部分通用 0 号协处理器访问指令,但仍有无法减少某些通用 0 号协处理器指令。原因有两点:一是因为修改中有时会使用到暂存寄存器,有些地方寄存器不够用;一是因为某些 0 号协处理器的寄存器值的改变需要反馈到硬件中(比如影响浮点部件的寄存器)。

3.2 硬件去特权化

与硬件辅助虚拟化设计中在 CPU 增加额外的操作模式相比,增加硬件去特权化要简单许多。为了支持 cache 指令硬件去特权化,CPU 在 0 号协处理器中某个寄存器中增加额外一位用于判断是否允许 cache 指令在非特权态执行。不妨称该位为判定位。硬件逻辑修改如图 3 所示。可以看到,CPU 只需要在 cache 指令执行过程中增加一次逻辑判断,即在非特权态下当且仅当判定位为 0 时才产生异常即可。同时,VMM 可以通过修改该位关闭或者打开 cache 指令的硬件去特权化。

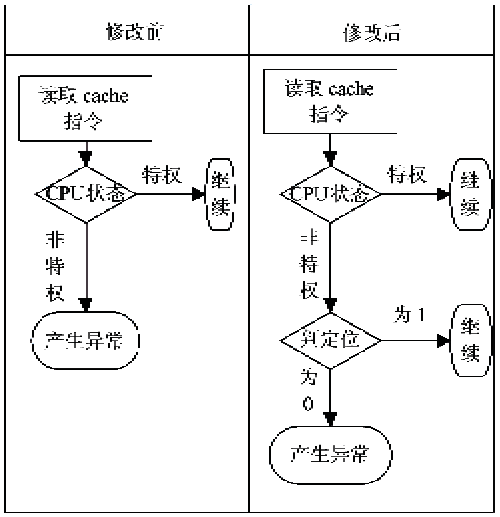

图 3 硬件逻辑改变

与通用 0 号协处理器访问指令不同，cache 指令在模拟的效果和在硬件上执行的效果一样，而且是处理器中唯一能访问 cache 的指令。因此修改操作系统代码方式无法减少 cache 指令。但是 cache 指令并不是像通用 0 号协处理器访问指令那样属于敏感指令，其本质上没有改变系统资源分配。因此如果 cache 指令在硬件上能直接执行，虚拟机的行为是正确的。虚拟机的安全性则可以通过图 4 的设计和保证虚拟机操作系统的行为安全来保证。

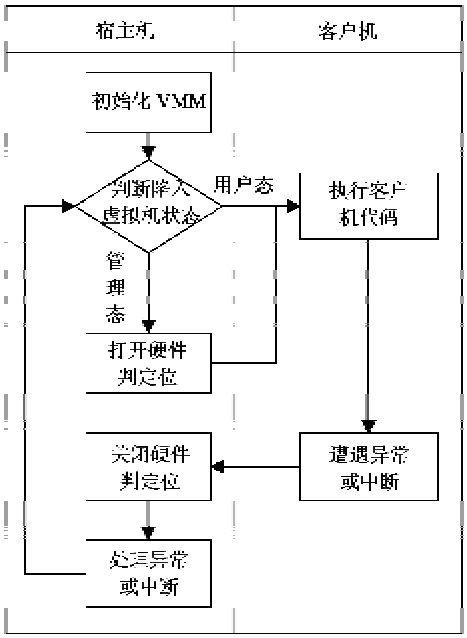

图 4 cache 指令硬件去特权化流程

MIPS 架构 CPU 拥有三个执行态：核心态，管理态，用户态。VMM 将客户操作系统运行在管理态下，将客户系统中用户程序运行在用户态下。图 4 描述了 cache 指令硬件去特权化的流程：在进入客户系统模拟时 VMM 判断如果客户系统需要在管理态下模拟（即客户系统执行操作系统代码），那么打开硬件去特权化，使 cache 指令可以非特权态下执行；如果客户系统不需要在管理态下模拟（即客户系统执行普通程序），那么进入客户系统之前则不打开硬件去特权化；VMM 在退出客户系统模拟时先关闭硬件去特权化，然后再做异常或中断处理。

按照如上流程，由于客户系统在管理态和用户态下切换均可被 VMM 捕获到，硬件去特权化可以保证仅在管理态下打开。由于在管理态下运行的是虚拟机的操作系统，保证客户操作系统的安全性便可保证硬件去特权化的安全性。

4 实验平台和优化结果分析

4.1 实验平台

实验平台是基于 MIPS 架构且主频 900MHz ~ 1GHz 的龙芯 3A 四核处理器，搭配 AMD RS780E 和 AMD SB700 作为南北桥的测试板。文件系统是 Debian squeeze 版本。宿主机系统和客户机系统内核均基于 Linux 2.6.36 版本修改。QEMU 是基于 0.14.0 版本修改[11]。

实验开始之前，首先利用当前流行的 KVM 平台[12]实现一个 MIPS 架构的简单的陷入模拟 VMM。以下从 CPU 虚拟化，内存虚拟化，I/O 虚拟化简要介绍其实现方式。

对于 CPU 虚拟化，MIPS 架构满足敏感指令是特权指令的子集，因此其不存在 x86 具有的虚拟化漏洞。不仅如此，MIPS 架构的特权资源基本上位于 0 号协处理器中，而且访问 0 号协处理器是固定的几条指令（mfc0，mtc0 等）。因此 MIPS 架构特权指令模拟要比 x86 架构简单。对于内存虚拟化，MIPS 架构 CPU 的 TLB 是程序员可见的，而且缺页异常也由软件完成。因此本文不需要实现 x86 架构中的影子页表[13]，可以使用更为简单的影子 TLB。对于 I/O虚拟化，MIPS 架构是统一编址的，没有类似 x86 架构中的 in，out 等特权指令。因此捕获虚拟机的 I/O访问不是通过特权指令的异常，而是通过操纵 TLB 实现的。

4.2 优化结果分析

实现软硬件协同去特权化方式后，实验平台上 Linux 2.6.36 内核从开始运行到进入用户空间这段时间内的异常分布如表 3 所示（异常数为 0 的异常未列出）。可以看到，总的异常数由原先的90多万

表 3 优化后异常分布

异常类型	异常数量（个）
中断	1355
TLB 修改	1
TLB 读	48
TLB 写	12
系统调用	1018
保留指令	44
协处理器	25823
溢出	1
TLB 缺失	200228
总计	228530

减少到了 20 多万,减少了大约 75%。其中特权指令造成的协处理器不可用异常由 70 多万减少到了 2 万多,减少了近 97%。需要注意的是中断(Interrupt)由以前的3600 多减少到了1300 多。原因主要是优化后系统启动到执行用户空间程序时间减少了,时钟中断比以前减少了近 63%。同时,这个数据也表明使用的时间减少了近 63%。可见从总体性能上来说使用软硬件协同去特权化取得了非常好的效果。

表 4 列出了优化后各个特权指令造成的异常。可以看到,通用 0 号协处理器访问指令(mfc0,mtc0等)都减少了 90% 以上,cache 指令造成的异常完全消失。可见修改代码方式可以减少大部分的敏感指令造成的异常,硬件去特权化则可以完全消除非敏感指令的特权指令造成的异常。实验结果与理论预期一致。

表 4　优化后特权指令造成的异常分布

特权指令	异常数量(个)
mfc0	11683
dmfc0	217
mtc0	10290
dmtc0	194
tlbr	0
tlbwi	125
tlbwr	33
tlbp	93
eret	3185
cache	0
其他指令	3

异常的减少同时也带来了性能的大幅度提高。图 5 显示了 SPEC CINT2000 测试程序分别在优化后的虚拟机和优化前的虚拟机上的运行性能提升的百分比。可以看到,大多数测试程序提升了 100% 以上。其中,最好的情况提升了近 900%,最差的情况也提升了 62%。因此,这种软硬件协同优化方法可以大幅提高虚拟机的性能。

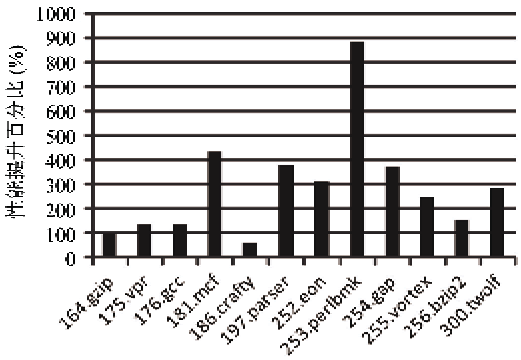

图 5　优化后 SPEC CINT2000 性能提升百分比

5　结　论

特权指令和敏感指令的处理是虚拟机对 CPU 虚拟化的主要内容。本文提出的软硬件协同设计方法对敏感指令和非敏感的特权指令采用不同方式处理,并在 MIPS 架构 CPU 的测试中取得了很好的效果。这种方法对于其他精简指令集计算机(RISC)体系结构虚拟化的优化方式上也有很好的借鉴意义。另外,这种软硬件协同设计方法也引出了对于非敏感指令的特权指令硬件优化方式的探讨,并在 MIPS 架构上验证了使用硬件去特权化的正确性。同时,本文也提出了对于硬件去特权化造成的安全性问题的解决方法,对于硬件辅助虚拟化设计也有很好的参考意义。未来的工作将针对软硬件协同设计的特点对 CPU 虚拟化做进一步研究。

参考文献

[1] Popek G J, Goldberg R P. Formal requirements for virtualizable third generation architectures. *Comm ACM*, 1974, 17(7):412-421

[2] Smith J E, Nair R. Virtual Machines: Versatile Platforms for System and Processes. USA: Morgan Kaufmann/Elsevier, 2005. 384-385

[3] Hu W, Liu Q, Wang J, et al. Efficient binary translation system with low hardware cost in computer design. In: Proceedings of the 27th International Conference on Compute Design, California, USA, 2009. 305-312

[4] Agesen O, Garthwaite A, Sheldon J, et al. The evolution of an x86 virtual machine monitor. *SIGOPS Operating System Review*, 2010, 44: 3-18

[5] Adams K, Agesen O. A comparison of software and hardware techniques for x86 virtualization. *ACM SIGARCH Computer Architecture News*, 2006, 34: 2-13

[6] Robin J S, Irvine C E. Analysis of the intel pentium's ability to support a secure virtual machine monitor. In: Proceedings of the 9th Conference on USENIX Security Symposium, Berkeley, USA, 2000. 129-144

[7] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization. *SIGOPS Operating System Review*, 2003, 37: 164-177

[8] Intel Coporation. Intel Virtualization Technology Specifi-

cation for the IA-32 Intel Architecture. http://www.intel.com, 2005

[9] Doorn L. Hardware virtualization trends. In: Proceedings of the 2nd International Conference on Virtualization Execution, New York, USA, 2006. 45-45

[10] Popek G J, Kline C S. The PDP-11 virtualization architecture: a case study. In: Proceedings of the 5th ACM Symposium on Operation System Principles, Texas, USA, 1975. 97-105

[11] Bellard F. Qemu, a fast and portable dynamic translator. In: Proceedings of the USENIX 2005 Annual Technical Conference, California, USA, 2005. 41-46

[12] Kivity A, Kamay Y, Laor D, et al. KVM: the linux virtual machine monitor. In: Proceedings of the Linux Symposium 2007, Ottawa, Canada, 2007. 255-230

[13] Creasy R J. The origin of the vm/370 time-sharing system. *IBM Journal of Research and Development*, 1981, 25: 483-490

A software-hardware co-design method for deprivileging instructions in virtualization

Tai Yunfang[*,**,***], Cai Wanwei[*,**,***], Liu Qi[*,**,****], Zhang Ge[****,*****], Zhang Longbing[*,**,****]

([*] Key Laboratory of Computer System and Architecture, Chinese Academy of Sciences, Beijing 100190)

([**] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***] Graduate University of Chinese Academy of Sciences, Beijing 100049)

([****] Loongson Technology Corporation Limited, Beijing 100190)

([*****] ChongQing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 401122)

Abstract

To reduce the overhead of privileged instructions and sensitive instructions in virtual machines, this paper presents a hardware-software co-design method to deprivilege these instructions. The main idea is to modify the source code of the operating system to deprivilege sensitive instructions and modify hardwares to deprivilege privileged insensitive instructions. The two ways are coordinated to minimize the overhead of these instructions, in order to improve the performance of virtual machines. The results of the experiments on the software-hardware co-design method conducted on the boards with MIPS CPUs showed that the exceptions caused by the privileged instructions and sensitive instructions decreased by nearly 97% and the performance of majority of programs in SPEC CINT2000 was improved by more than 100%.

Key words: virtualization, deprivileging instructions, software-hardware co-design, privileged instructions, sensitive instructions