CN43-1258/TP
ISSN 1007-130X

计算机工程与科学
COMPUTER ENGINEERING & SCIENCE

2008年第30卷第A1期
Vol. 30, No. A1, 2008

文章编号:1007-130X(2008)A1-0111-05

现代计算机ISA支持虚拟化的研究*

Research on Modern Computer ISA Supporting Virtualization

刘宇航,郝沁汾,肖利民,祝明发
LIU Yu-hang, HAO Qin-fen, XIAO Li-min, ZHU Ming-fa
(北京航空航天大学计算机学院,北京 100083)
(School of Computer Science and Technology, Beijing University of Aeronautics and Astronautics, Beijing 100083, China)

摘 要:本文在Popek,Goldberg等人的研究基础上,进一步讨论了ISA支持虚拟化的内容。首先形式化地定义了与这一议题相关的概念,将指令重新进行了分类,讨论如何缩小被虚拟机监控器干预并解释执行的指令在整个指令集的比例,从而在保证可虚拟化的前提下,进一步挖掘了可以提高的效率空间;给出并证明了关于在VMM不存在时的任一指令序列 t 与其对应于VMM存在时的等价序列 t' 之间的映射的一个定理,这些结果不仅给可虚拟化计算机ISA的设计以及高效VMM的构造提供了指导,也有助于评估已有体系结构并对其作修改,以使一个虚拟机系统可被构造。通过本文,我们对虚拟机"现象"有一个更好的理论认识,同时还能向机器架构师和系统设计者提供实际的指导方针。

Abstract: Based on the study of Popek, Goldberg and others, this paper discusses the issue of ISA supporting virtualization in further, and defines formally the concepts that concerns with the issue; reclassifies the instructions, and discusses how to reduce the proportion of instructions that are executed with intervention and interpretation of VMM(virtual machine monitor) in the entire instruction set so as to expand the efficiency space in further with the premise of virtualizable; not only gives but proves a theorem about the mapping between any instruction sequences t when VMM doesn't exist and its counterpart t' when VMM does exist. These results not only provide guidelines for the design of ISA and the construction of efficient VMM, but also help to assess the existing ISA and make some necessary modification to enable a virtual machine system can be constructed. By this paper, we can reach a better theoretic-understanding of the Virtual Machine "phenomenon", but also to provide practical guidelines for machine architects and system designers.

关键词:指令集体系结构;敏感指令;可虚拟化;效率;虚拟机;虚拟机监控器

Key words: instruction set architecture (ISA); sensitive instructions; virtualizable; efficiency; virtual machine(VM); virtual machine monitor(VMM)

中图分类号:TP309

文献标识码:A

1 引言

虚拟计算机技术是近几年来发展比较迅速的技术之一,虚拟机的研究和设计是当前计算机系统设计和开发中一个重要的活跃方向。

Popek和Goldberg[1]通过建立第三代计算机系统的一个简化模型,试图引出一个标准来测试一种体系结构是否支持虚拟机,他们提出关于ISA支持虚拟化的基本定理为:对任何传统的第三代计算机,一个VMM可能被构造,如果这台计算机的敏感指令集是其特权指令集的一个子集。这个定理提供了一个简单的充分条件来保证可虚拟化。事实上,作者只是提出了一个充分不必要条件,而使用一个充分不必要条件作为测试一种体系结构是否支持虚拟

* 收稿日期:2007-04-13;修订日期:2007-07-10
基金项目:国家863计划资助项目(2006AA01Z108,2007AA01A127)
作者简介:刘宇航(1985-),男,安徽亳州人,博士生,研究方向为虚拟机;郝沁汾,副教授,硕士生导师,研究方向为高性能计算机体系结构、并行计算、高性能计算机评测与应用;肖利民,教授,博士生导师,研究方向为计算机体系结构、计算机系统软件、高性能计算机系统、高端服务器系统;祝明发,教授,博士生导师,研究方向为计算机体系结构、并行处理、高性能计算机系统和网络、网格计算、高性能计算、系统级VLSI芯片(SoC)设计、人工智能、图像处理。
通讯地址:100083 北京市北京航空航天大学计算机学院;E-mail:liuyuhang@cse.buaa.edu.cn
Address: School of Computer Science and Technology, Beijing University of Aeronautics and Astronautics, Beijing 100083, P. R. China

机的评判标准是不精确的，或者说有时是失效的。如果一种体系结构满足这个条件，那么它支持虚拟机，但是，如果一种体系结构不满足这个条件，我们就不能确定它是否支持虚拟机。并且可能为了满足一些不必要的条件而损失了效率。

因此，寻求 ISA 支持虚拟化的充分必要条件是必要的。本文的一个思路是将指令根据其行为进行划分，扩大无害指令的范围。下面给出本文用到的一些概念的确切定义。

在某一个指令集体系结构(ISA)的计算机上，如果具有如下三个特性的一个控制程序可被构造，那么称这一 ISA 支持虚拟机，或者说 ISA 可虚拟化。(1)该控制程序提供的环境在实质上与真实机器提供的环境相一致；(2)运行在该控制程序提供的环境中的程序的执行是高效率的；(3)该控制程序对系统资源是完全控制的。

该控制程序称为虚拟机监控器(Virtual Machine Monitor，VMM)，如图 1。这里，因为 VMM 有资源控制的特性，且被称为控制程序(Control Program，CP)，故称为虚拟机监控器，而不是许多中文文献中所译的虚拟机监视器。由控制程序提供的环境称为虚拟机(Virtual Machine)。

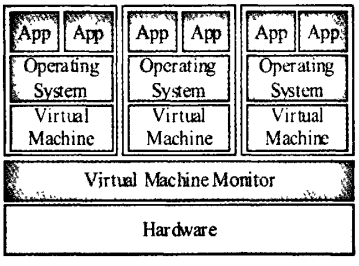

图 1　虚拟机的通用模型

一台虚拟机被认为是真实机器的一个复制品，而同一真实机器上的不同虚拟机之间是隔离的[11]。由 VMM 提供的环境等价于真实机器，这是 VMM 的首要特性。

VMM 的第二条特性是有效率。这要求虚拟处理器的指令在统计意义上占主要部分的一个子集被真实处理器直接执行，而没有被 VMM 软件干涉。显然传统模拟器和完全的软件解释器(complete software interpreter machine，CSIM)[11]不属于虚拟机的范畴。

第三个特性，VMM 对存储器、外设等资源完全控制，在虚拟机中运行的程序不可能访问任何没有显式分配给它的资源，且 VMM 可以重新获得已分配资源的控制权。

2　现代机的一个模型

现代计算机系统的体系结构同第三代计算机相比，并没有太大的变化。而且现代计算机系统已经普遍地给用户提供了没有显式的 I/O 设备或指令的扩展了的软件机器，例如 I/O 设备被作为存储器单元对待，I/O 操作通过恰当的存储器转移实现。为了使模型反映系统本质同时又非常简单易于后面讨论，可以将 I/O 设备或指令排除在模型之外。因此，本文在 Popek 和 Goldberg 构建的一个第三代机简化模型[1]的基础上继续讨论 ISA 支持虚拟化的问题。

机器可存在于有限数目的状态之一，$S=<E,M,P,R>$，即每个状态 S 有四个组成部分：可执行存储器 E，处理器模式 M，程序计数器 P 和重定位界限寄存器 R。

存储器寻址是相对于重定位寄存器的内容完成的。可执行存储器是一个普遍的字编址或字节编址的大小为 q 的存储器。记号 $E[i]$将是指在 E 中第 i 个存储单元的内容，从而 $E=E'$ 当且仅当对任意的 $0\leqslant i<q$，$E[i]=E'[i]$。不管机器的当前模式是什么，重定位界限寄存器 $R=(l,b)$一直是活跃的。寄存器的重定位部分 l 给出一个绝对地址。界限部分 b 将给出虚拟存储器的绝对大小。如果期望访问所有存储器，重定位部分必须设置为 0，界限部分必须设置为 $q-1$。

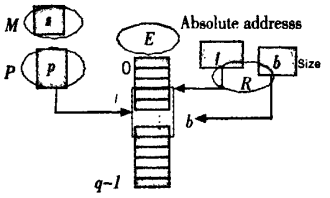

图 2　现代计算机的一个模型

处理器的模式 M 要么是 s(supervisor mode，管理程序模式)，要么是 u(user mode，用户模式)。程序计数器 P 是一个相对于 R 的内容的地址，作为 E 的一个指针而存在，指示下一条被执行的指令。三元组$<M,P,R>$即程序状态字(program status word，*psw*)。为了便于讨论，我们假设一个 *psw* 可被记录在一个存储器存储位置中，并使用 $E[0]$和 $E[1]$来分别存储旧的程序状态字(*old－psw*)和提取一个新的程序状态字(*new－psw*)。

S 的每一个构成要素只能取有限数量的值。称有限状态集为 C。那么，一条指令(instruction)就是一个由 C 到 C 的函数 $i:C\rightarrow C$。

陷入是一个重要的系统保护机制。一条指令 i 被称为陷入(trap)，如果 $i(E_1,M_1,P_1,R_1)=(E_2,M_2,P_2,R_2)$，这里 $E_1[j]=E_2[j]$，for $0<j<q$，$E_2[0]=(M_1,P_1,R_1)$，$(M_2,P_2,R_2)=E_1[1]$。

因此，当一个指令陷入，除了存放 *psw* 的位置 0 恰在指令陷入前生效外，存储器没有改变。在陷入的指令之后生效的 *psw* 被从位置 1 取出。

有一种特殊的陷入类型是存储器陷入。当指令试图访问超过寄存器 R 或者物理存储器边界的地址时会引起存储器陷入。# define memorytrap" ifa＋l≥q ‖ a≥b then trap"

3　基于指令行为对 ISA 的划分

接下来，我们根据指令作为机器状态的一个函数的行为，将指令分类。

指令 i 是特权(privileged)指令当且仅当对任意状态对 $S_1=<e,s,p,r>$和 $S_2=<e,u,p,r>$(其中 $i(S_1)$和 $i(S_2)$都不存储器陷入)，有 $i(S_2)$陷入而 $i(S_1)$不陷入。

有一组重要的指令将被称为敏感指令(sensitive instructions)，这一组的成员将与一个特定机器的可虚拟化能力有重要关联。我们定义两类敏感指令。

一条指令是控制敏感的如果存在一个状态 $S_1=<e_1, m_1, p_1, r_1>$且 $i(S_1)=S_2=<e_2, m_2, p_2, r_2>$使得 $i(S_1)$不memorytrap,且下列情况至少一个成立

(1)$r_1 \neq r_2$

(2)$m_1 \neq m_2$

一个 VMM 的第三个特性中提到对系统资源的完全控制是必须的。控制敏感指令是那些影响或潜在地影响控制的指令。

对一个整数 x,我们定义一个运算符⊕,使得 $r'=r \oplus x=(l+x,b)$,即重定位界限寄存器有自己的基准值(可通过 x 的值改变)。显然,在特定状态下唯一能被访问的存储器是那些被重定位界限寄存器 R 指定的部分。所以为了检查指令的执行结果,我们不妨在状态描述中只包含被 R 所限制的存储器部分。记号$[l+b+x]$表明了那些被限制存储器的内容。因为 $r=(l,b)$,所以 $E|r$ 表示存储器中从 l 到 $l+b$ 的内容。可用记号 $S=<e|r,m,p,r>$来指定一种状态。

直观地,一条指令是行为敏感的(behavior sensitive),如果它的执行效果依赖于重定位寄存器的值,也就是依赖于它在真实存储器中的位置,或者依赖于工作模式。定位界限寄存器或工作模式在指令执行后不匹配的两种情形划入控制敏感的种类中。

Popek 和 Goldberg 将行为敏感分为两类[1],一种情形称为位置敏感:一个指令的执行行为依赖于它在存储器中的位置。另一种情形称为模式敏感,一个指令的行为受机器工作模式的影响。

我们认为行为敏感指令应分为三类:第一类是纯粹的位置敏感指令;第二类是纯粹的模式敏感指令;第三类既是模式敏感指令又是位置敏感指令。现分别给出它们的形式化定义如下:

指令 i 是纯粹的位置敏感指令,如果存在一个整数 $x \neq 0$ 及以下状态:

$(a)S_1=<e|r,m,p,r>$,and

$(b)S_2=<e|r \oplus x,m,p,r \oplus x>$,Where

$(c)i(S_1)=<e|r,m_1,p_1,r>$,

$(d)i(S_2)=<e|r \oplus x,m_2,p_2,r \oplus x>$,and

(e)neither $i(S_1)$ *or* $i(S_2)$memorytrap,

以下两者至少一个成立:$(f)e|r \neq e|r \oplus x$,*or*

$(g)p_1 \neq p_2$,or both

指令 i 是纯粹的模式敏感指令,如果存在以下状态:

$(a)S_1=<e|r,m_1,p,r>$,and

$(b)S_2=<e|r,m_2,p,r>$,

$(c)(h)$

$(d)i(S_2)=<e|r \oplus x,m,p_2,r \oplus x>$,and

(e)neither $i(S_1)$ *or* $p_1 \neq p_2$,memorytrap,

$(f)m_1 \neq m_2$,

以下两者至少一个成立:

$(g)e|r \neq e|r \oplus x$,or

$(h)p_1 \neq p_2$,or both

指令 i 是模式位置敏感指令(注意它既不是纯粹的模式敏感指令又不是纯粹的位置敏感指令),如果存在一个整数 $x \neq 0$ 及以下状态:

$(a)S_1=<e|r,m_1,p,r>$,and

$(b)\ S_2=<e|r \oplus x,m_2,p,r \oplus x>$,Where

$(c)i(S_1)=<e|r,m_1',p_1,r>$,

$(d)i(S_2)=<e|r \oplus x,m_2',p_2,r \oplus x>$,and

(e)neither $i(S_1)$ *or* $i(S_2)$memorytrap,

$(f)m_1 \neq m_2$,

以下两者至少一个成立:

$(g)\ e|r \neq e|r \oplus x$,or

$(h)p_1 \neq p_2$,or both

Popek 和 Goldberg [1]将敏感指令划分为控制敏感指令和行为敏感指令两类,并认为行为敏感指令一定不是控制敏感的。这在逻辑上和现实实例上都是不完善的。控制敏感指令是那些试图改变资源分配的指令,行为敏感指令是那些行为和执行结果依赖于资源配置的指令。从理论上,存在一些指令既是控制敏感的又是行为敏感的,不妨称为严重敏感指令。

IA-32 架构下的 POPFD 指令是控制敏感且是行为敏感的指令,其主要功能是从栈顶弹出一个 32 位的双字,之后存储该双字到 EFLAGS 寄存器中;其执行效果是 EFLAGS 寄存器中的所有非保留位,除了 VIP,VIF,VM 之外都可能被修改,而 VIP,VIF,VM 位保持不变。它之所以是控制敏感的,是因为 EFLAGS 寄存器中的 IOPL 位可能被修改,该位表示当前程序访问 I/O 空间的特权级。而如果 POPFD 不是特权指令且在用户空间执行,则影响 VMM 对 I/O 资源空间的绝对控制权[3]。同时因为其中断使能位(interrupt-enable flag)在特权模式下可被修改,而在用户模式下不能修改,因此该指令是模式敏感的,从而是行为敏感的[5]。

在 VLIW(如现有的 Crusoe 处理器使用 128 位长的 VLIW 指令,在一个周期中最大执行 4 个运算)中若将常规的若干条控制敏感指令与若干条行为敏感指令集中成一条指令,则形成一条逻辑上的严重敏感指令,类似的情况还出现在宏融合(Macro-Fusion)技术中(例如讯驰 Pentium M 处理器在 Core 架构中采用)。因此讨论严重敏感指令既具有理论意义,也有现实意义。

根据一条纯粹模式(纯粹位置、模式位置)敏感指令是纯粹的行为敏感指令还是严重敏感指令,分成 1 类纯粹模式(纯粹位置、模式位置)敏感指令和 2 类纯粹模式(纯粹位置、模式位置)敏感指令。

我们称一条指令 i 是敏感的(sensitive),如果它是控制敏感或行为敏感的。如果 i 不是敏感的,那么称它是无害的(innocuous)。

考虑效率则意味着无害指令的直接执行。设 C 表示一个集合,则 C 中元素的个数即集合 C 的基数记为 $\|C\|$.

虚拟机的效率 $Eff \propto \frac{\|\text{无害指令集}\|}{\|\text{敏感指令集}\|}$,设计算机指令集体系结构 ISA 的指令条数为 n,即 $n=\|\text{ISA}\|=\|\text{无害指令集}\|+\|\text{敏感指令集}\|$,从而 $Eff \propto \frac{n-\|\text{敏感指令集}\|}{\|\text{敏感指令}\|}=\frac{n}{\|\text{敏感指令}\|}-1$。

即虚拟机的效率与敏感指令所占的比例负相关。

综上所述,ISA 的基于指令行为的分类结果如图 3。

既然我们已经将指令分了类，我们需要更精确地详细说明 VMM。

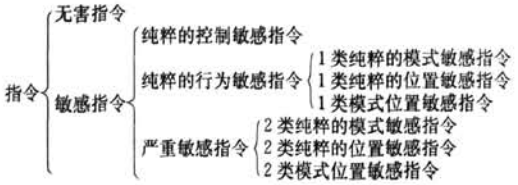

图 3　基于指令行为对指令的划分

4　指令(序列)在 VMM 存在前后的对应关系

我们在讨论 VMM 构造的基础上，论证指令(序列)在 VMM 存在前后的对应关系。

VMM=CP=$<D,A,\{v_i\}>$，如图 4。

调度器(Dispatcher) D 的初始指令位于硬件陷入的位置：位置 1 中的 P 的值，它可被认为是控制程序的顶级控制模块，由它决定调用哪一个模块，它可以唤醒第一类或第二类的模块。

分配器(Allocator) A 决定提供什么系统资源是分配器的任务。在单个 VM 的情形下，分配器只需将 VM 与 VMM 保持分离。在一个 VMM 主持许多 VM 的情形中，分配器应避免将相同资源(诸如存储器的一部分)同时分配给不止一个 VM。当一条特权指令尝试执行(将改变与虚拟机环境相关联的机器资源)时，分配器将被调度器唤醒。

解释器(interpreters)针对所有其他的引起陷入的指令，用来模拟陷入指令的执行结果。每一个特权指令对应一个解释器程序。

设 v_i 表示一个指令序列。我们就可以把解释程序的集合表示成 v_i 的集合，用记号$\{v_i\}$来表示，$i=1,\cdots,m$，其中 m 是特权指令的数目。当然，调度器和分配器也是指令的序列。

psw(在位置 1，当一个陷入发生时被硬件装入)将模式设置成 supervisor，将 *PC* 设置成指向调度器起始位置。进一步地，我们假设其他程序将运行在用户模式，也就是说，*PSW*(控制程序在最后一个操作将 *PSW* 装入，将控制权还给运行的程序)将其模式设置为 user。因此有必要用控制程序中的一个位置来记录 VM 的仿真模式。

将机器状态集 C 划分为两个部分。第一个集合 C_v 包括 VMM 在存储器中存在时机器的所有状态，而存储在位置 1 的 PSW 中的 P 的值等于 VMM 的起始位置。第二个集合 C_r 包括剩下的状态。这两个集合分别反映了真实机器在有无一个 VMM 时的可能的状态。

一个虚拟机映射(virtual machine map)(VM map) f：$C_r\to C_v$，就在指令序列集 I 中所有的操作符 e_i 而言是一个一对一的同态。也就是说，对任意状态 $S_i\in C_r$ 和任意指令序列 e_i，存在着一个指令序列 e'_i，满足 $f(e_i(S_i))=e'_i(f(S_i))$。作为对 VM 映射定义的一部分，我们要求对每一个 e_i，对应的指令序列 e'_i 都能被找到并执行。

f 应是一对一的，为了使这一映射概念更精确，我们将给出一个特别的 VM 映射，如图 5。让该控制程序占有物

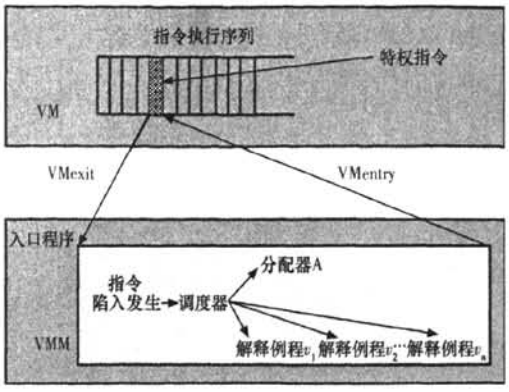

图 4　特权指令的执行过程

理存储器的前 k 个位置。也就是说，$E[0]$和 $E[1]$保留给 psw(程序状态字)，控制程序占有从 2 到 $k-1$ 的位置。之后的 w 个位置用于一个虚拟机。我们假设 $k+w<q$。所以 $f(E,M,P,R)=(E',M',P',R')$，其中 $S=<E,M,P,R>$是没有 VMM 时的虚拟机的状态。

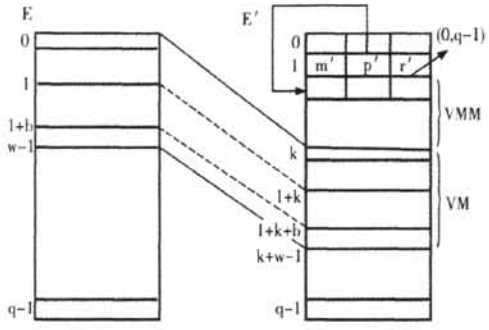

图 5　一个虚拟机映射 f

假设 $r=(l,b)$中 b 的值总是小于 w，那么有：

$E'[i+k]=E[i]$, *for* $i=0,w-1$,

$E'[i]$=the control program, for $i=2$ to $k-1$,

$E'[1]=<m',p',r'>$,

m'=supervisor,

p'=first location of the control program,

$r'=(0,q-1)$,

$E'[0]=<m,p,r>$as last set by trap handler,

$M'=u(user)$,

$ft_2t_1(S_1)=t_2'ft_1(S_1)$

$R'=(l+k,b)$, where $R=(l,b)$

上面这个虚拟机映射被认为是标准的 VM 映射[1]。现在我们可以声明“等价”意味着什么了，可以更准确地表述什么是“本质上效果一致”。假设两台机器都启动了，一个在状态 S_1，另一个在状态 $S'_1=f(S_1)$，那么由 *VMM* 提供的环境等价于真实机器，当且仅当对任意状态 S_1，如果真实机器终止在状态 S_2，那么虚拟机终止在状态(*1)。

在 VMM 不存在时的任一指令序列 t 与其对应于 VMM 存在时的等价序列 t'之间的映射，设为 $Q:t\to t'$，$Q(t)=t'\Leftrightarrow\forall S\in C, ft(S)=t'f(S)$，

则有下列命题成立：

(1) 若 $Q(t)=t'$，$Q(i)=i'$，

则 $Q(it)=i't'$.

(2) $Q(i)=\begin{cases} i, i \text{ 为无害指令;} \\ \text{InterRoution}|\,i, i \text{ 为敏感指令。} \end{cases}$

(3)对任意长度的指令序列 $t=i_1 i_2 \cdots i_r \cdots i_n (1 \leqslant n)$,

$Q(t)=\begin{cases} t, t \text{ 为无害指令序列;} \\ d_1 d_2 \cdots d_r \cdots d_n, \text{对} \forall \text{指令 } d_r (1 \leqslant r \leqslant n), d_r = Q(i) \end{cases}$

即 $Q(t)=Q(i_1)Q(i_2)\cdots Q(i_r)\cdots Q(i_n)$.

(4)若 $Q(t_1)=t_1'$,$Q(t_2)=t_2'$,则 $Q(t_1 t_2)=t_1' t_2'$。

以上,t_1,t_2 为任意长度≥1 的指令序列,i,i_1,i_2,…,i_r,…,i_n 为单条指令。

证明:设 S_1 为任意状态。

(1)∵$Q(t)=t'$

$\therefore ft(S_1)=t'f(S_1)$ (1)

$\therefore i'ft(S_1)=i't'f(S_1)$ (2)

又设 $t(S_1)=S$,

∵$Q(i)=i'$

$\therefore fi(S)=i'f(S)$ (3)

即 $fit(S_1)=i'ft(S_1)$ (4)

(4)式与(2)式联立得

$fit(S_1)=i't'f(S_1)$ (5)

(5)式等价于 $Q(it)=i't'$。

命题(1)得证。

(2) 设 i 为任意无害指令,S 为真实机器上的任意状态,$S'=f(S)$。其中 $S=(e|r,m,p,r)$,$S'=(e'|r',m',p',r')$。但是,由 f 的定义可得,$e'|r'=e|r$,$p'=p$,在 r'和 r 中的界限值是相同的。根据定义,$i(S)$不能依赖于 m 或者 l(r 的重定位部分),所有的其他参数在 S 和 S'中都相同。因此,一定有 $i(S)=i(S')$这种情况。

(3) 命题(3) 是(2) 的自然的推广。由 *VMM* 的结构和陷入的结束后的返回位置(如图 4),可得。

(4)∵$Q(t_1)=t_1'$

$\therefore ft_1(S_1)=t_1'f(S_1)$ (6)

设 $t_1(S_1)=S_2$,

∵$Q(t_2)=t_2'$

$\therefore ft_2(S_2)=t_2'f(S_2)$ (7)

即 $ft_2t_1(S_1)=t_2'ft_1(S_1)$ (8)

由(1)式得

$t_2'ft_1(S_1)=t_2't_1'f(S_1)$ (9)

(8)、(9)式联立得

$ft_2t_1(S_1)=t_2't_1'f(S_1)$ (10)

(10)式等价于 $Q(t_1t_2)=t_1't_2'$。

命题(4)得证。

5 结束语

基于现代计算机系统的一个形式化模型,本文讨论了 ISA 支持虚拟化的议题。通过将指令重新进行了分类,讨论如何将原来定义为敏感指令的指令划归到无害指令,从而缩小被虚拟机监控器干预并解释执行的指令在整个指令集的比例,即在保证可虚拟化的前提下,进一步开发可以提高的效率潜力;给出并证明了关于在 VMM 不存在时的任一指令序列 t 与其对应于 VMM 存在时的等价序列 t'之间的映射的一个定理。这些结果有助于寻求 ISA 支持虚拟化的充分必要条件,有助于可虚拟化计算机 ISA 的设计以及高效 VMM 的构造。寻求 ISA 支持虚拟化的充分必要条件是本文进一步研究的方向。

参考文献:

[1] Popek G, Goldberg R. Formal Requirements for Virtualizable 3rd Generation Architectures. Communications of the A. C. M, 17(7): 412-421, 1974.

[2] Gagliardi U O, Goldberg R P. Virtualizable Architectures[C]// Proc of ACM AICA Int'l Computing Sympo, 1972.

[3] IA-32 Intel® Architecture Software Developer's Manual Volume 2B, 4-111, 2007, Intel Corporation.

[4] Galley S W. PDP-10 Virtual machines. Proc. ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, Cambridge, Mass., 1969.

[5] James E. Smith, Ravi Nair. 2006. 7. Virtual Machines: Versatile Platforms for Systems and Processes, pp. 385. Printed in China by Publishing House of Electronics Industry, under special arrangement with Elsevier (Singapore) Pte Ltd.

[6] Goldberg, R. P. (Ed). Proc. ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, Cambridge, Mass., 1973.

[7] IBM Corporation. IBM Virtual Machine Facility/370: Planning Guide, Pub. No. GC20-1801-0, 1972.

[8] Lauer H C, Snow C R. Is Supervisor-State Necessary[C]// Proc of ACM AICA Int'l Computing Symp, 1972.

[9] Hugh C. Lauer, David Wyeth, A recursive virtual machine architecture, Proceedings of the workshop on virtual computer systems, p. 113-116, March 26-27, 1973, Cambridge, Massachusetts, United States [doi>10.1145/800122.803951].

[10] Meyer R A, Seawright L H. A Virtual Machine Time Sharing System. IBM Systems J. 9, 3(1970).

[11] Robin J S, Irvine C E. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor[C]// Proc of the 9th Usenix Security Symp, Usenix, 2000: 129-144.

[12] Adve, V., C. Lattner, M. Brukman, A. Shukla, and B. Gaeke. LLVA: A Low-level Virtual Instruction Set Architecture[C]// Proc of the 36th Int'l Symp on Microarchitecture, 2000.