多核虚拟机监控系统①

蒋 楠[1], 吴俊敏[1,2], 朱晓东[1], 李利锋[1], 张朋飞[2], 黄 景[1]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230059)
[2](中国科学技术大学 苏州研究院, 苏州 215123)

摘 要: 随着多核处理器时代的到来, 虚拟化技术被广泛使用, 而多核虚拟机就是其中一种. 目前多核虚拟机监控一般都是采用硬件虚拟化的技术, 即通过虚拟化技术虚拟出多个串口来达到监控目的. 给出一种基于系统级共享内存的多核虚拟化监控系统方案, 并提供了完整的设计与实现方案.

关键词: 多核; 虚拟化; 虚拟机; 监控系统; 共享内存

Multi-Core Virtual Machine Monitoring System

JIANG Nan[1], WU Jun-Min[1,2], ZHU Xiao-Dong[1], LI Li-Feng[1], ZHANG Peng-Fei[2], HUANG Jing[1]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230059, China)
[2](Suzhou Institute, University of Science and Technology of China, Suzhou 215123, China)

Abstract: With the era of multi-core processors, virtualization technology is widely used, and multi-core virtual machine is one of them. The current multi-core virtual machine monitors are generally used hardware virtualization technology through simulating a number of virtual serial ports to monitor for the purpose. This inventive, given based on shared memory multi-core system-level monitoring system virtualization solution, and gives a complete design and implementation.

Key words: multi-core; virtualization; virtual machine; monitoring system; shared memory

1 概述

受功耗、散热以及工艺等因素的限制, 一味增加单处理器主频的设计策略已经走到尽头, 取而代之的是片上多核处理器(CMP, Chip Multi-core Processor)的诞生. 随着硬件资源规模的不断扩展、处理能力的快速增强、资源种类的日益丰富、应用需求灵活多样, 对新的计算机模式的需求变得日益强烈.

虚拟化[1]技术作为一种新的计算模式应运而生. 虚拟化技术能够动态组织多种计算资源, 隔离具体的硬件体系与软件体系的紧密依赖关系, 实现透明化的可伸缩性计算, 从而灵活构建满足多种需求的计算环境, 提高资源的利用率. 使用虚拟化技术, 可以实现在一个硬件平台同时运行多个操作系统[2].

监控系统行为是虚拟机系统的核心任务, 监控系统可用于调度任务、负载均衡、向管理员报告软硬件故障, 并广泛控制系统的使用情况. 监控系统具体实现上方法各异, 本文选择基于系统级共享内存的方案来实现对系统行为的监控.

本文余下内容结构: 第 2 部分背景介绍; 第 3 部分对整体设计策略给出详细描述; 第 4 部分针对第 3 部分的设计方案给出具体的实现; 第 5 部分为实验结果与总结.

2 背景

2.1 虚拟化技术

虚拟化是从逻辑角度出发的资源配置方案, 是对物理资源的一种抽象. 抽象的结果是, 在只有一台计算机硬件的情况下, 通过虚拟化技术, 可以让多个操

① 基金项目:中央高校基本科研业务费专项资金(WK0110000020)
收稿时间:2012-02-09;收到修改稿时间:2012-03-14

作系统同时运行在此计算机硬件上，并且让这些操作系统都认为自己独享整个硬件，资源划分对操作系统是透明的[3]. 本文中提到的多核虚拟机指的是硬件平台的CPU拥有多个CPU核，使用硬件辅助虚拟化技术使得各个虚拟机运行在独立的 CPU 核上. 多核虚拟机系统的框架图见图 1.

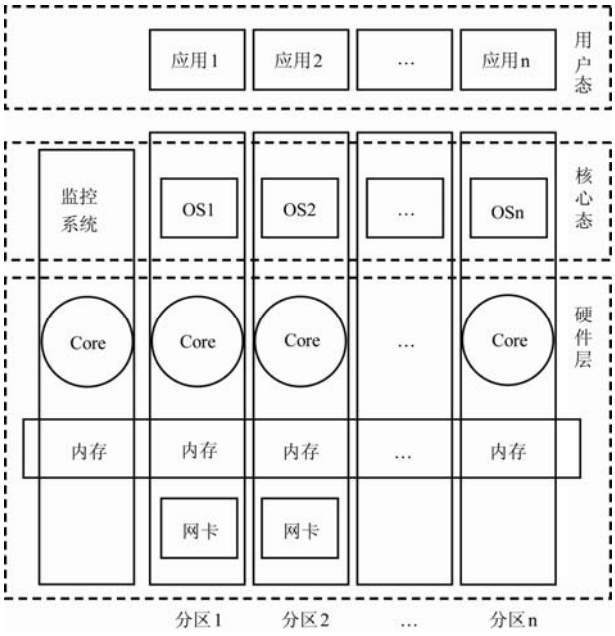

图 1　多核虚拟机系统框架图

2.2 多核虚拟机监控技术

通过虚拟化技术，使得在一套硬件平台上，同时运行多个操作系统. 此时，我们很自然的需要监控整个系统的状态以及监控特定的操作系统行为. 目前的多核虚拟机的监控通常都是采用硬件虚拟化的技术来实现，即通过虚拟化技术虚拟出多个串口，使得每个操作系统都认为自己拥有串口资源，通过各自虚拟串口即可进行监控[4].

虚拟化系统下的 I/O 访问需要在客户操作系统、VMM、设备驱动程序、I/O 设备共同参与下才能完成. 所谓的虚拟设备就是由 VMM 创建的，提供给客户操作系统进行 I/O 访问的虚拟 I/O 设备. 客户操作系统只能观察到属于它的虚拟 I/O 设备，客户操作系统的所有 I/O 访问都被发往它的虚拟 I/O 设备，然后 VMM 软件从虚拟 I/O 设备中获取客户操作系统的访问请求，继而完成真正的 I/O 访问. 使用 I/O 虚拟化技术，由于中间层的存在，性能上往往损失较多，然而为了优化性能，需要对 VMM 软件进行过多的修改，其中主要修改是对设备驱动程序的开发支持. 设备驱动程序是导致系统崩溃的一个重要原因，这种对 VMM 的修改将给系统的安全性和可靠性留下重大隐患. 而且还会使得 VMM 软件设计变得更加复杂，不利于 VMM 软件更新升级. 也就是说使用 I/O 虚拟化技术需要在性能与复杂性、可靠性上进行一个折中，很难同时满足高性能、高可靠性以及低复杂性.

硬件虚拟化技术的框架图见图 2.

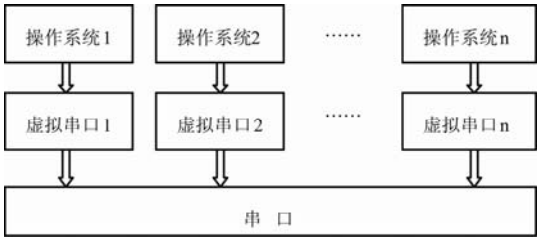

图 2　串口虚拟化框架图

2.3 基于共享内存的多核虚拟机监控

本文采用一种很新颖的方法来实现多核虚拟机的监控，即利用操作系统级的共享内存、伪终端(PTY)等来实现多核虚拟机监控. 基于共享内存的多核虚拟机监控的框架图如图 3.

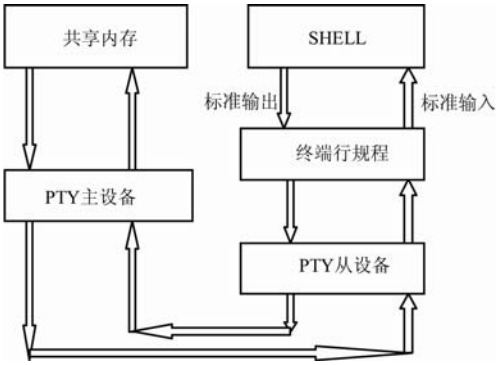

图 3　基于系统级共享内存监控系统逻辑图

基于共享内存的多核虚拟机监控方案中，所有操作系统通过将命令发往系统级共享内存，然后交由后端(相应的操作系统)去执行，最后将命令执行结果发往共享内存[5]. 所谓的共享内存，就是在物理内存中开辟一段可以被操作系统和监控系统同时访问的内存空间，因为高级语言只能操作逻辑地址，所以操作的

时候需要将物理地址映射成逻辑地址. 系统级共享内存逻辑图如图 4.

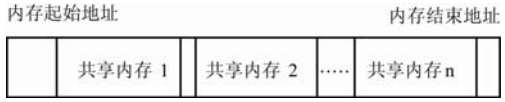

图 4 共享内存逻辑图

图中的共享内存 1 标识操作系统 1 和监控系统共享的一段内存空间, 这段内存空间是操作系统 1 和监控系统都可以访问的, 同理, 共享内存 2 标识操作系统 2 和监控系统共享的一段内存, 共享内存 n 是操作系统 n 和监控系统共享的一段内存空间.

共享内存在实现上采用以下步骤: 第一, 系统初始化的时候, 在物理内存中预留一定大小的内存空间, 这部分内存空间不分配给任何操作系统, 也不分配给监控系统; 第二, 操作系统与监控系统在使用共享内存的时候, 通过共享内存 API 进行调用. 共享内存 API 封装了对预留内存空间申请以及由物理地址空间到虚拟地址空间的转换. 通过共享内存 API 即可在预留的内存空间内申请一块一定大小的内存, 同时返回申请内存的虚拟地址. 为了实现该内存在操作系统与操作系统以及操作系统与监控系统之间共享, 实现上使用标识符对各个共享内存段进行标识(系统为所有标识符建立一个表, 当第一次用一个标识符调用共享内存 API 的时候, 就在该表插入一行数据, 记录标识符到虚拟内存地址的映射关系, 当再次以相同的标识符调用共享内存 API 的时候, 则直接返回虚拟内存地址), 两个操作系统或者操作系统与监控系统使用同一个标识符调用共享内存 API 的时候返回相同的虚拟地址; 第三, 封装一些共享内存同步与互斥机制的 API, 方便对共享内存的使用.

使用基于系统级共享内存的实现方法, 无需对 VMM 软件与操作系统内核进行任何修改, 所有修改都在操作系统上层进行, 拥有高安全性与低复杂性.

3 基于系统级共享内存的多核虚拟机监控系统的设计方案

3.1 监控系统概述

我们的多核虚拟机监控系统称之为 KMON, 运行在一单独的处理器核上. 同时本文中提到的操作系统指的是 Linux 操作系统.

监控系统主要完成的功能包括:

基于串口查询整个系统的信息.

包括键入命令查询 CPU, 内存, 网卡等资源的基本信息; 键入命令查看正在运行的操作系统实例基本信息和状态, 包括 CPU 利用率、内存使用状况、网卡使用状况等信息.

基于串口查看操作系统实例的输出.

包括加载 1 份操作系统实例, 可以监控该操作系统实例的输出; 加载另 1 份操作系统实例, 可以先后或同时监控多个操作系统的输出, 可以任意切换; 可以切换到初始的系统信息查看.

监控系统主要包括以下几个模块: 进入分区模块、退出分区模块、系统监控模块.

进入分区模块: 使得监控系统 KMON 切换到需要监控的分区, 以监控该分区. 所有用户操作将直接作用到该分区, 也只作用到该分区, 不会影响其他分区的状态.

退出分区模块: 使得监控系统 KMON 退出所处的分区, 此时终端退回到初始状态, 用户操作都作用于监控系统 KMON 本身, 而不会影响任何分区操作系统的状态.

监控系统模块: 此模块用于监控多核虚拟机系统的资源使用情况以及所有操作系统的状态.

3.2 监控系统设计图

设计图说明:

(1) 初始状态: 这个状态下, 在控制台键入的命令直接由监控系统 KMON 解析执行, 不会将命令写到任何共享内存空间, 从而也不会被任何操作系统执行.

(2) OS#的共享内存空间: “#”是 OS 的编号, 编号在系统初始化的时候由监控系统 KMON 进行自动分配, 同时, KMON 为每个 OS 在内存的共享区域分配一块单独的空间, 也就是说每个 OS 有自己的专属共享内存空间.

由于共享内存的存在, 使得数据可以在操作系统与监控系统之间传递. 用户首先在监控系统键入带有某个操作系统编号的命令, 然后, 用户键入的其他命令就会被写到该操作系统与监控系统共享的那段内存. 同时, 操作系统端以一个后台守护进程定时检测共享内存, 一旦发现有用户命令, 该守护进程就读取用户命令, 并交给操作系统来执行, 并将命令的执行

结果写回共享内存. 监控系统定时检测共享内存, 一旦发现有命令执行结果, 就读取该结果显示到终端. 利用一段可以被监控系统与操作系统共同访问的内存区域, 实现用户命令行、命令执行结果等数据在监控系统与操作系统之间传递来达到监控操作系统行为的目的.

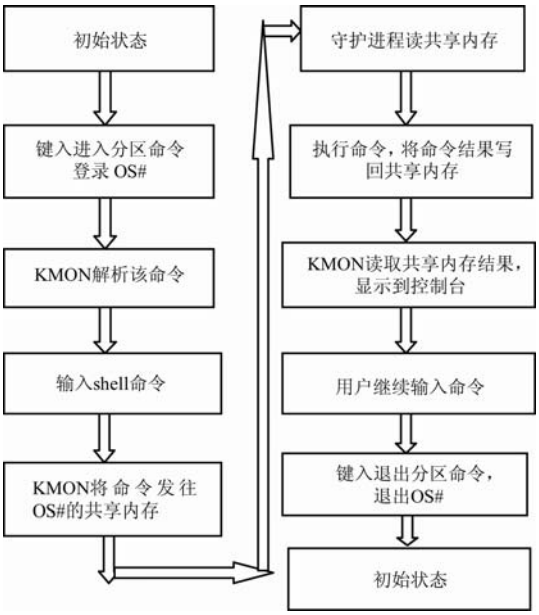

图 5　监控系统总体流程图

4　监控模块的实现方案

4.1 用户命令重定向的实现

操作系统加载完成后, KMON 系统默认将用户键入的命令交给 KMON 去执行, 当用户键入登入分区命令后, 系统将根据登入分区命令中的分区号确定将用户键入的命令写到相应的那段共享内存.

在我们的设计中, KMON 以 u-boot 作为原型系统进行二次开发而成的, 为了达到命令重定向, 我们对 u-boot 源代码进行如下的修改:

首先, 增加一个全局变量, cmd_redirection(命令重定向), 用于标识用户键入的命令会被重定向到哪个分区. cmd_redirection 初始值为-1, 标识用户键入的命令直接由监控系统 KMON 执行.

然后, 在 KMON 中增加一条登入分区命令. 当用户输入登入分区命令的时候, cmd_redirection 被设置为登入分区命令中指明的分区号. 然后 KMON 就可以根据该分区号确定命令重定向的位置.

最后, 当用户键入退出分区命令的时候, KMON 会修改 cmd_redirection 为-1, 在此之后, 用户键入的命令将继续由 KMON 执行, 而不会由操作系统执行. 这里有点值得一说, 由于在执行退出分区的时候是处在命令重定向的时候, 此时用户键入的命令是会被重定向到某个共享内存的, 所以我们需要对重定向进行一点小小修改: 当用户键入的命令不是退出分区命令才将其写到共享内存, 否则就直接修改 cmd_redirection 值为-1, 同时不会将退出分区命令写到共享内存.

流程图如下:

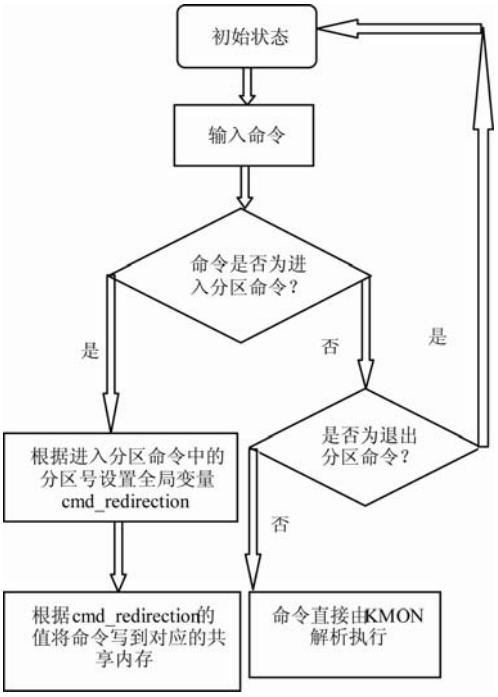

图 6　命令重定向实现流程图

4.2 操作系统获取命令、执行以及返回结果的实现

经过上一步, 已经做到将用户键入的命令进行重定向到特定的共享内存了, 接下来要做的是: 操作系统从共享内存读取用户键入的命令、执行、返回命令执行结果的实现.

这里的实现我们采用守护进程来达到目的. 主要的实现步骤包括:

第一, 编写守护进程代码.

要想使得编写的进程成为守护进程, 需要满足编写守护进程的一些规范: 1)调用 umask 将文件屏蔽字设置为 0; 2)调用 fork 函数, 然后使父进程退出; 3)调用 setsid 以创建一个新会话; 4)将当前工作目录改为根目

录; 5)关闭不再需要的文件描述符; 6)重定向其标准输入、标准输出、标准出错. 在具体实现上, 我们将这些规则封装在一个名为 init_daemon 函数中, 任何进程只要调用此函数即可成为守护进程.

在执行用户命令的实现上, 我们采用了伪终端(PTY)来实现. fork 一个子进程, 然后父进程用于获取在各项内存中的命令, 子进程用于和伪终端(PTY)绑定以执行命令以及将命令执行结果写回到共享内存.

这里之所以使用伪终端 PTY, 是因为用户输入的命令一般都是针对 shell 操作, 而 shell 是需要与标准输入输出设备联系在一起才能正常工作, 我们的系统中

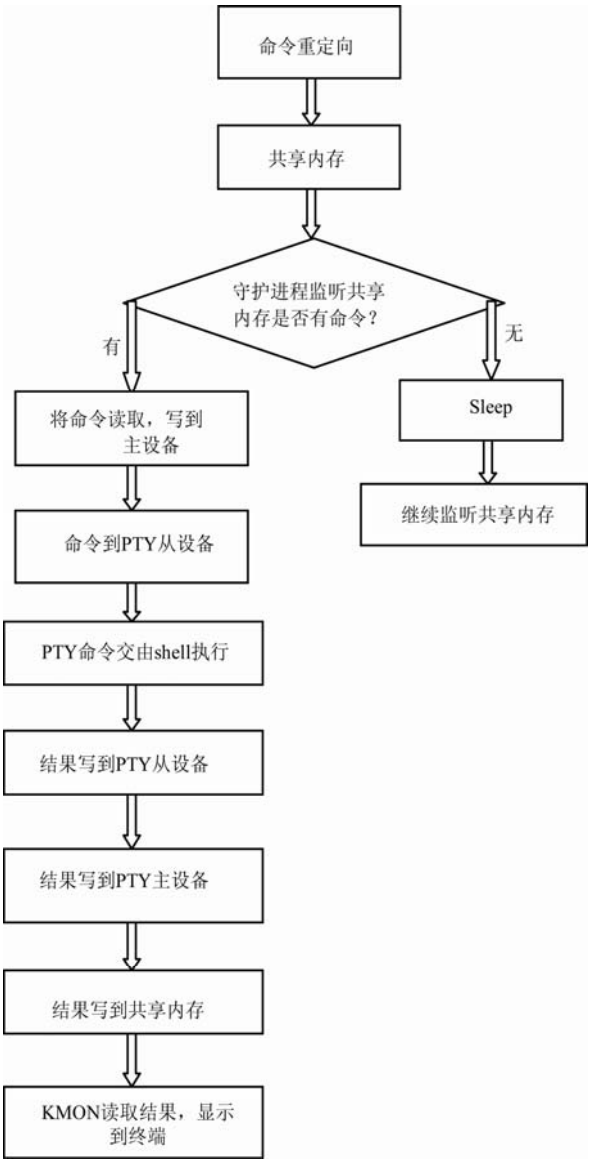

图 7 用户命令执行流程图

串口设备个数有限, 无法做到为每个操作系统分配一个串口设备, 所以我们采用伪终端来使得每个操作系统“认为”他们都拥有各自的标准输入输出设备.

最后, KMON 负责读取共享内存中的命令返回结果, 将其输出到屏幕.

第二, 将守护进程加载到操作系统的文件系统.

这里提到的文件系统, 本质上是一堆文件集合, 主要包括内核镜像文件、一些基本命令的可执行文件、操作系统启动的一些脚本等文件, 此文件系统我们可以理解为 Linux 操作系统启动所需要的所有文件的集合. 由于实验环境所提供的硬件是无硬盘设备, 无法将守护进程放到硬盘, 所以需要将守护进程加载到操作系统镜像文件中, 即文件系统中. 制作文件系统时候, 我们使用 busybox 定制文件系统, 这里我们修改 busybox 的 makefile 文件来实现将守护进程添加到文件系统.

第三, 修改操作系统的启动文件使得在操作系统启动时守护进程即被加载.

在操作系统的启动脚本中加上加载监控系统守护进程的命令. 这里需要修改文件系统中的文件, 然后重新编译文件系统.

流程图如图 7.

4.3 同时监控多个操作系统的实现

Linux 操作系统的运行信息, 如 CPU 利用率、内存利用率、网络负载情况等信息是可以通过读取/proc 目录下特定文件获知的——读取 stat 文件即可获取 CPU 运行信息、读取 meminfo 即可获取内存信息、读取 net/dev 即可获取网络负载情况.

在 KMON 中添加监控系统状态的新命令, 该命令每隔一定时间(预设 3 秒)依次读取各个操作系统 proc 目录下相应文件获知各个操作系统的运行信息, 经过一定的计算后, 将结果写到共享内存, 最后将共享内存的信息输出到控制台.

4.4 操作系统之间切换的实现

基于进入分区命令和退出分区命令来实现对监控操作系统的切换. 进入分区命令实现切换到某个分区, 而退出分区命令则实现退出该分区, 切换到初始状态. 通过对这两个命令组合使用就可以轻松实现各个操作系统之间的自由切换.

5 实验结果与总结

5.1 实验结果

下面给出关于此监控模块的几个重要命令，及执行结果.

(1) kmconsole 命令(登录操作系统命令)

```
Octeon ebh5600# kmconsole 2
```

kmconsole 命令执行结果

```
###kmconsole done,cmd_redirection=2###
[part2root@~]$
```

键入命令: ls

```
[part2root@~]$ls
bin                                 proc
daemon_debug.txt                    root
dev                                 sbin
etc                                 share
examples                            sys
home                                tmp
```

(2) kmexitconsole 命令(退出操作系统命令)

```
[part2root@~]$kmexitconsole
```

执行 kmexitconsole 后:

```
Octeon ebh5600#
```

(3) kmmonotorall 命令(监控所有操作系统状态命令)

```
***************part_id:2************
           Receive          Transmit
       |bytes  packets |bytes  packets
    lo:     0     0           0     0
 veth0:     0     0           0     0
 veth0:     0     0           0     0
MEM
  total size:254544 kb,248 mb
  free size:206160 kb,201 mb
CPU
cpu_utilization_ratio:0.00%
CPU0  cpu_utilization_ratio:0.00
************************************

***************part_id:3************
           Receive          Transmit
       |bytes  packets |bytes  packets
    lo:     0     0           0     0
 veth0:     0     0           0     0
 veth0:     0     0           0     0
MEM
  total size:254544 kb,248 mb
  free size:206220 kb,201 mb
CPU
cpu_utilization_ratio:0.00%
CPU0  cpu_utilization_ratio:0.00
************************************
```

5.2 总结

本文提出了一种基于系统级共享内存来实现多操作系统的监控的方案，并给出了详细的设计与实现，这种实现方案达到了作为监控模块的需求——用户操作界面友好，系统开销小. 同时，较之 I/O 虚拟化，这种实现方案拥有低复杂度与高安全性.

参考文献

1 Ongaro D, Cox AL, Rixner S. Scheduling I/O in Virtual Machine Monitors. Proc. of 4th ACM/US ENIX International Conference on Virtual Execution Environments. Seattle, WA, 2008. 1–10.

2 Smith JE, Nair R: The Architecture of Virtual Machines; IEEE Computer Magazine, May 2005,38(5).

3 Devine S, Bugnion E, Rosenblum M. Virtualization system including a virtual machine monitor for a computer with a segmented architecture. US Patent, 6397242, October 1998.

4 Barham, Dragovic P, Fraser B, Hand K, et al. Xen and the art of virtualization. Proc. of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP. ACM Press, New York, 2003. 164–177.

5 卢仕听,尤凯迪,韩军,等.MIPS 内存管理单元的设计与实现.计算机工程,2010,36(21):270–274.