多线程 ARM 虚拟机的设计与实现

胡小龙,周艳科
(中南大学 信息科学与工程学院,湖南 长沙 410075)

摘 要:虚拟机技术广泛应用于代码移植、跨平台计算和模拟硬件机器、嵌入式系统模拟等领域。该技术以软件的方式构建通用机器的硬件的仿真环境,实现机器指令在处理器中的运算过程。在介绍了虚拟机原理的基础上,设计并实现了基于多线程的 ARM 虚拟机,初步模拟了 ARM9 的指令执行过程。

关键词:虚拟机;多线程;ARM
中图分类号:TP311 文献标识码:A

Design and implementation of the ARM virtual machine for multithreading

HU Xiao Long, ZHOU Yan Ke
(School of information Science and Engineering, Central South University, Changsha 410075, China)

Abstract: The technology of virtual machine is widely applied in many fields, such as code transplanting, cross-platform computing, hardware simulation, and embedded system simulation. The main purpose is to simulate the environment of general hardware by means of software and realize the operation of byte codes in processor. After introducing the basic theory, this paper designs and realizes a solution to ARM virtual machine based on multithreading, which gives a preliminary study on simulating the process of instructions in ARM9.

Key words: virtual machine; multithreading; ARM

所谓虚拟化就是把计算机的资源,如运算能力,存储空间以及 IO 设备抽离出来,让资源的使用方式更具效率。虚拟化技术可以提高硬件的处理能力,简化软件的重新配置过程。其主要作用体现在服务器整合、动态负载均衡、快速应用部署、灾难恢复、处理器的前期开发的软件模拟、代码移植、反病毒等领域。

虚拟化技术最早产生于上世纪 60 年代[1],但是被广泛地应用却是近两年的事情。虚拟化技术成为研究热点的主要原因有两个:首先是服务的细分,为了提高应用的灵活性,很多技术都将 IT 应用从大型系统转向“模块化”,这就需要一个个应用独立出来,这种需求也正符合如今人们所熟知的 SOA 思想;同时硬件发展为这种细分需求提供了一个平台,硬件发展速度越来越快,如双核、四核 CPU 的推动,服务器内存扩展能力越来越大。也就是说,仅仅一个简单的应用已经无法充分利用服务器的资源,很多资源被浪费,虚拟化技术可以帮助用户大大地提高服务器资源的利用率,降低成本,并且能够迎合细分的服务。

实现虚拟化,主要有三种方法:完全虚拟化,如图 1 所示;准虚拟化,如图 2 所示;操作系统虚拟化,如图 3 所示[2]。

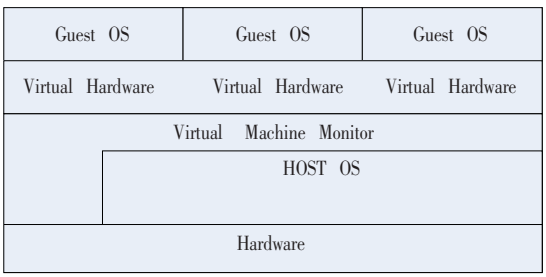

图 1 完全虚拟化示意图

1 虚拟机原理

虚拟机用软件模拟指令在硬件机器中执行的全过

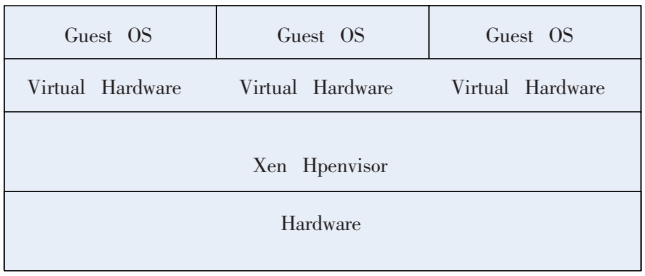

图 2 准虚拟化示意图

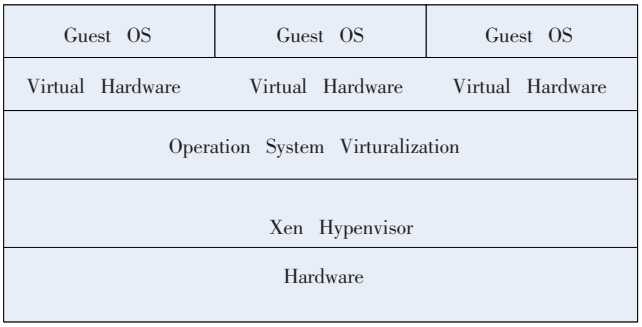

图 3 操作系统虚拟化示意图

程,即先从磁盘中读取字节码文件,存储到虚拟机开辟的内存段中,再将指令加载到虚拟机寄存器并予以执行。它并不是指某个特定的软件,而是一整套完整的规范,可以用不同的程序设计语言在不同的硬件平台上实现[3]。不论是硬件机器还是虚拟机系统,只要遵循相同的规范,即可运行遵循该规范的可执行代码。下面将从处理器引擎、内存管理、任务管理和输入输出 4 个方面分别讨论 ARM 虚拟机的实现规范。

1.1 处理器引擎

根据运算数据交换平台的不同,主要有基于堆栈的处理器和基于寄存器的处理器两种引擎。前者在运行时对系统资源的消耗要远小于后者,但堆栈将会延伸到内存;后者的计算处理则全部在芯片内部的寄存器上完成,因此后者的处理速度要远远高于前者。由于基于堆栈的处理器已经退出主流的行列,ARM 虚拟机将采用基于寄存器的执行引擎来模拟具体硬件实现,并在 ARM 虚拟机中构建堆栈,从而继承堆栈在处理函数调用和递归等问题上的优势。虚拟寄存器是系统的核心,ARM 虚拟机通过寄存器之间的运算来执行字节码指令。作为处理器中的数据存储单元,各类寄存器均有不同用途,例如通用寄存器用来存储和转换数据和地址,指针寄存器指向堆栈段的栈顶位置,段寄存器则标识出代码段、数据段和堆栈段之间的界限。

1.2 内存管理

程序指令和数据在内存中的存储方式有两种:降序(big-endian)和升序(little-endian)。降序是指数据的高位字节存放在内存中的低位地址中;升序则相反,是指数据的低位字节存放在内存中的低位地址中。内存管理负责内存的分配和回收问题,主要有显式内存管理法 EMM(Explicit Memory Management)和自动内存管理法 AMM(Automatic Memory Management)两种方法[4]。其中,前者由程序员人工完成内存的分配和回收工作,例如 C 语言用 malloc()调用来实现分配一段内存空间,用 free()函数来释放不再使用的内存。后者则由程序员完成内存的分配,由系统负责自动回收,例如 Java 语言对于已分配的内存,由内存管理根据其作用域完成与否决定是否收回该段内存。一般来说,显示内存管理法要比自动内存管理法的速度要快,内存分配算法相对简单。

1.3 任务管理

现代计算机使用多任务技术[5],即多个任务共享同一处理器。该技术主要依靠队列和堆栈,为排队等候的每个任务分配一段时间,并在系统中建立一张进程表,记录每一个进程的状态。相应的状态有 3 种:正在执行、被挂起和阻塞。通常采用抢占式多任务技术,即由操作系统决定进程间的切换。例如在著名的时间片轮转(round-robin)算法里,每个进程所分配的执行时间的长度是固定的,进程调度不停地依次遍历整个进程表,让每个进程都有机会占用同样长度的处理器时间。

1.4 输入/输出

虚拟机依托于宿主平台,因此输入/输出功能的实现相对较简单,在系统调用层基础上的标准输入/输出函数即可胜任。运算速度有所限制,但无须接触机器级的中断调用和输入/输出操作。

2 32 位 ARM 虚拟机的设计

32 位处理器的地址空间仅能支持 4 GB,ARM 虚拟机采用 32 位地址空间技术,在 Win32 位平台上开辟一个 4 B 的空间来表示 32 位整型,并涉及一组虚拟寄存器、一段虚拟内存空间和一个类 ARM9 的指令集,分别对应于需要仿真的机器部件。在 32 位地址空间的基础上,采用寄存器引擎、显示内存管理法、抢占式多任务技术等基本工作机制,构建虚拟机软件的基本结构。

2.1 虚拟寄存器

ARM 虚拟机共有 37 个寄存器,包括 31 个 32 位的通用寄存器(包括程序计数器 PC 在内),6 个 32 位的状态寄存器[6]。ARM 处理器共有 7 种不同的处理模式,在每一处理器模式中有一组相应的寄存器组。在任意的处理器模式下,可见的寄存器包括 15 个通用寄存器(R0~R14),一个或两个状态寄存器及程序计数器(PC)。在所有的寄存器中,有些是各模式共用同一个物理寄存器,有一些寄存器是各模式自己拥有的独立的物理寄存器。表 1 列出了各处理器模式下可见的寄存器情况。

2.2 虚拟的 MMU 的设计和管理

在 ARM 虚拟机设计中,通过 C++的 vector 来开辟一块连续的内存地址作为虚拟内存空间,存储程序的字节码指令。由于在 32 位机器上无法提供这么多的物理内存,因此 ARM 虚拟机采用先预读后分配的原则来开辟

表 1 各处理器模式下可见的寄存器情况

用户模式	系统模式	特权模式	中止模式	未定义模式	外部中断模式	快速中断模式
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

虚拟内存，即先对字节码文件的大小进行预读，根据其大小再从物理内存中分配足够的内存作为虚拟内存，如果剩余的物理内存无法满足需要，则退出该次分配。由于当前主流硬件平台(如 Intel 公司的系列处理器)均采用降序方式，在 ELF 文件的加载过程中无须改变数据存储方式，因此在 ARM 虚拟机的设计中使用降序方式，提高了虚拟机的执行效率，同时保证了设计的通用性和简单性。另外，在内存的管理和回收方面，采用显式内存管理法，从而确保虚拟机的运行速度、内存分配的灵活性和对程序的控制权。

2.3 虚拟机基本结构

图 4 是 ARM 虚拟机的总体结构，由以下模块组成：

指令集功能仿真模块和系统调用仿真模块都属于功能仿真模块，前者按序仿真每条指令(现在不包含时序模拟)，其主要用于性能模拟器执行结果正确性验证；后者采用宿主机代理方式仿真系统调用处理过程[7]。

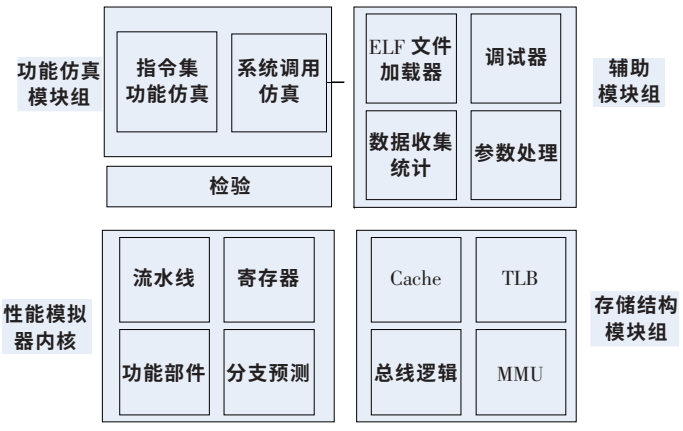

图 4 ARM 虚拟夹的总体结构

流水线模块、功能部件模块、寄存器模块和分支预测模块共同组成了性能模拟器内核，为 ARM 虚拟机的主要组成部分。其中，流水线模块包含所有和流水线相关的数据结构和处理函数；功能部件和寄存器模块用于建模功能部件和 ARM 寄存器组；分支预测模块用于建模分支预测器。

Cache 模块、TLB 模块、总线逻辑和 MMU 模块共同组成了存储器层次模块组，其模拟了一个 ARM 存储器的行为过程。

ELF 文件加载器模块、数据收集统计模块、参数处理模块和调试器模块为 ARM 虚拟机的辅助模块，ELF 文件加载器模块用于将可执行文件加载到模拟器内存中；数据收集统计模块用于收集统计模拟器各项状态数据，如指令执行条数；参数处理模块用于处理命令行参数；调试器模块用于调试用户程序。

检验模块用于验证性能模拟器的正确性，它通过将每条指令的性能模拟结果并与其功能仿真结果进行比较，就可以判断出指令是否被正确执行，然后可以根据用户要求采取不同的操作。

3 ARM 虚拟机的工作流程

3.1 基本流程

ARM 虚拟机从读入 ELF 文件到执行完毕的全过程，算是一个生命周期。该周期按功能不同，可分为 5 个阶段：(1)启动虚拟机，并处理命令行输入；(2)初始化虚拟机运行环境；(3)调入并加载文件；(4)格式化内存存储方式；(5)执行指令。其中，第(2)和(5)阶段是整个程序运行的核心，完成了虚拟机的初始化和执行 ELF 文件中分析出的代码段指令的主要工作。

3.2 初始化虚拟机

阶段(1)通过对命令行的处理,获取字 ELF 文件的信息。该信息通过 ELF 文件加载器得到程序代码段、数据段、堆栈段的信息传递到阶段(2)初始化虚拟机的InitVM 函数中。其中数据段和堆栈段的缺省长度均为 64 KB,而代码段的长度为 ELF 文件中代码指令所占用的实际长度。

InitVM 函数的执行流程是:(1)函数从文件头中提取必要的信息后,立即检查字节码长度,将其与数据段和堆栈段的长度之和与宿主机的可用内存比较。如果可用内存不够,则报错并退出虚拟机;否则宿主机将分配一段大小为三者之和的内存给虚拟机。(2)将段寄存器分别设置成为相应段的值。(3)将 ELF 代码加载到内存空间的代码段。具体算法如下:

If (ELF 文件大小为零或文件打开错误)

{记录并打印错误,并直接返回 }

If (代码段、数据段和堆栈段的长度之和 > 宿主机的可用内存)

{记录并打印错误,并直接返回 }

分配内存 R=malloc(代码段、数据段和堆栈段的长度之和);

初始化寄存器;

重新返回 ELF 文件的代码段部分, 调用 fseek() 定位代码段部分的位置;

读取代码段部分,并将其加载入所开辟的内存代码段中。

如果上述过程顺利执行,所获取的虚拟机内存地址空间如图 5 所示。

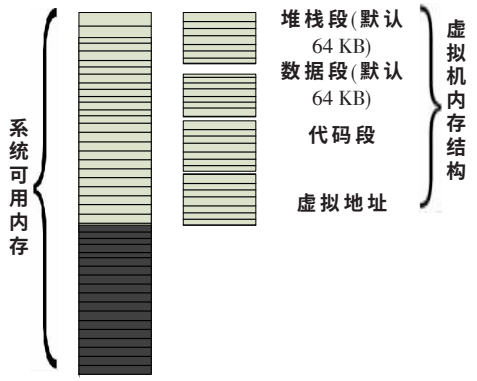

图 5 初始化后的内存地址空间

3.3 执行代码段

run()函数是 ARM 虚拟机的指令执行引擎,负责完成代码段指令的执行工作。该函数将字节码指令从虚拟地址 0 处开始逐条执行,当遇到 HALT 指令或发生错误时停止。指针寄存器 IP 始终指向下一条所需执行的指令。参照基于寄存器的处理器的设计方法,第(5)阶段指令执行过程的算法如下(其中,CPSR 表示指令指针寄存器值;CPSR 中表示内存地址为 CPSR 所存值的地址处的指令):

```
void run()
 {
   while(内存地址为 CPSR 不等于中止指令 HALT)
   {
   switch(内存地址为 CPSR 的指令)
   {
    case MOV:{…;}break;
          …各种指令的具体执行;
          default:{ERROR;}break;
       }
     将指令指针 CPSR 指向下一条指令;
 }
}
```

ARM 指令按照指令功能的不同来分,包括数据处理指令 (如 MOV、MVN、CMP)、内存访问指令 (如 LDR、STR)、内存批访问指令、跳转指令(如 B、BL)、访问状态寄存器指令(如 MRS)、软中断指令等。

指令模拟执行过程如图 6 所示。

3.4 流水线的实现

ARM9 采用哈佛结构的设计[8],采用 5 级流水线,其功能分别描述如下:

(1)取指。从存储器(CACHE/Memory)取出指令,并将其放入指令流水线。

(2)译码。指令译码,从寄存器堆中读取寄存器操作数。在寄存器堆中有 3 个操作数读端口,大多数 ARM 指令能在一个周期内读取其操作数。

(3)执行。一个操作数移位,产生 ALU 的结果,如果指令是 LOAD 或 STORE,则在 ALU 中计算存储地址。

(4)缓冲/数据访存。如果需要,则访问数据存储器;否则,ALU 只是简单的缓冲一个时钟周期, 以便所有的指令具有同样的流水线流程。

(5)回写。将指令产生的结果回写(Write back)至寄存器堆,包括任何从存储器中读取的数据。

采用多线程模拟 5 级流水线的执行过程如图 7 所示。

ARM 指令的实现方式(单线程和多线程),如表 2 所示,在 ARM 虚拟机运行 ARMlinux,记录了 10 次两种方式的起始时间,计算平均值。

从表 2 中可以看出,多线程的方法对指令流水线模拟的性能有很大改进。

表 2 单线程和多线程 ARM 指令的实现

	单线程	多线程
执行时间/s	8.225	7.224
加速比/%	12.17	

本文主要采用显式内存管理、代码段的解析、基于寄存器的机器引擎,多线程等关键技术,在以 Win32 为宿主平台的基础上, 设计并实现了基于多线程的 ARM

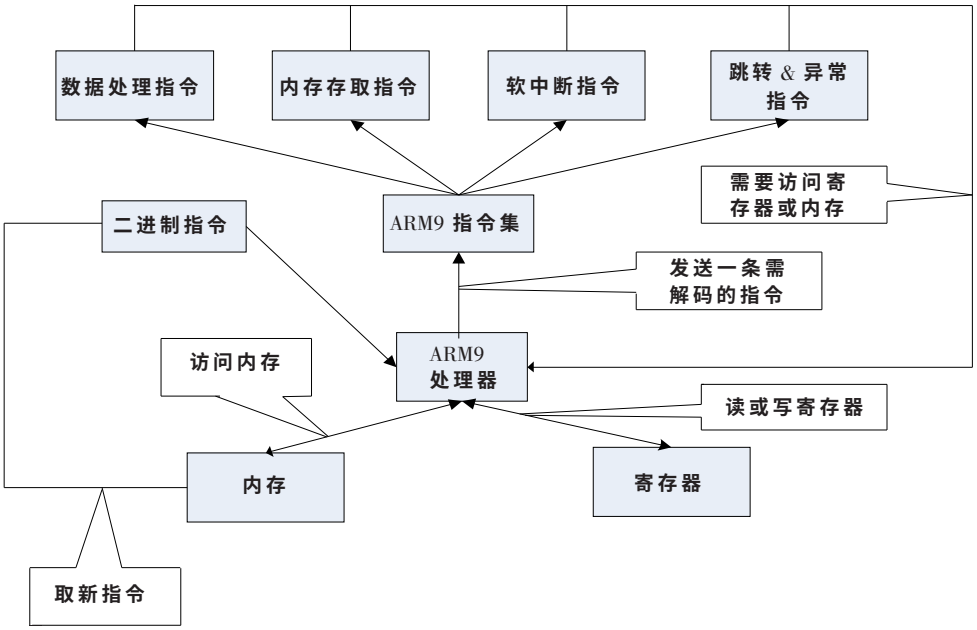

图 6　指令模拟执行过程

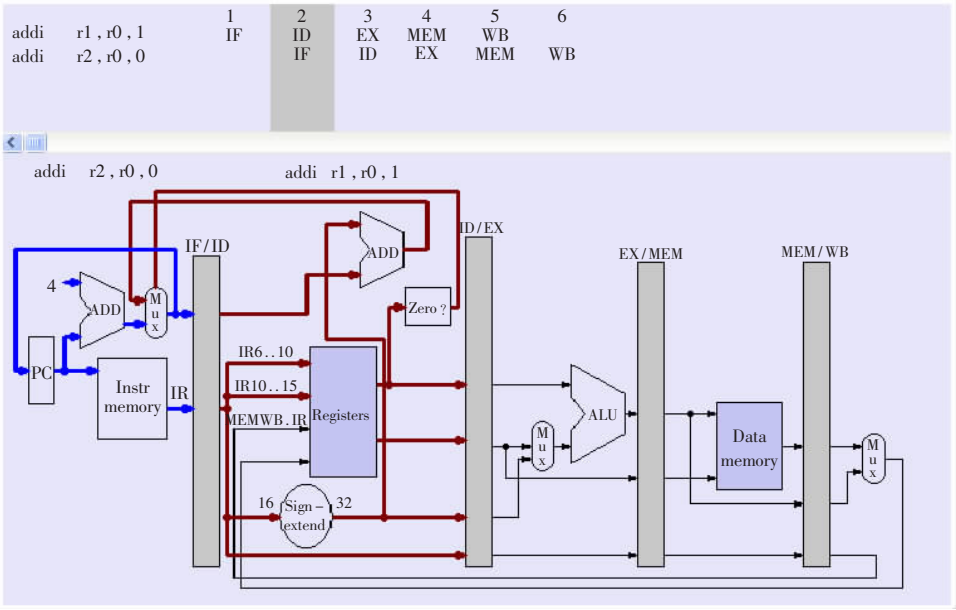

图 7　多线程模拟 5 级流水线的执行过程

虚拟机系统，能够在 32 位的内存地址空间和虚拟寄存器上执行指令，为模拟 ARM 处理器的仿真环境做出了有益的尝试。今后将进一步完善 ARM 嵌入式系统模拟的设计，充分发挥虚拟机在嵌入式开发方面的优势。

参考文献

[1] 李林华，盛浩，马世龙.基于寄存器引擎的 64 位虚拟机的实现[J].计算机工程，2005，31(2)：91-93.

[2] 刘洪浩.虚拟化，热潮不等于机会[J].程序员 2007 年精华本，2007.

[3] BLUNDEN B. Virtual Machine Design and Implementation in C/C++[M]. Wordware Publishing，2002.

[4] JONES R，LINES R. Garbage Collection：Algorithms for Automatic Dynamic Memory Management[M]. John Wiley & Sons，1996.

[5] TANENBAUM A S，WOODHULL A S. Operating System: Design and Implementation[M]. Prentice Hall，1997.

[6] 杜春雷.ARM 体系结构与编程[M].北京：清华大学出版社，2003.

[7] 万寒，高小鹏. Design and Implementation of a Simulation Framework for the ARM Embedded System. IEEE Computer Society，北京：北京大学，2008.

[8] 谭华.嵌入式系统软件仿真器的研究与实现[D].西安：电子科技大学，2006.

（收稿日期：2009-02-05）