



(12) 发明专利申请

(10) 申请公布号 CN 106970821 A

(43) 申请公布日 2017. 07. 21

(21) 申请号 201610019289. 0

(22) 申请日 2016. 01. 12

(71) 申请人 阿里巴巴集团控股有限公司

地址 英属开曼群岛大开曼资本大厦一座四
层 847 号邮箱

(72) 发明人 张献涛 丁圣阁 田双太

(74) 专利代理机构 北京润泽恒知识产权代理有
限公司 11319

代理人 赵娟

(51) Int. Cl.

G06F 9/455(2006. 01)

G06F 3/038(2013. 01)

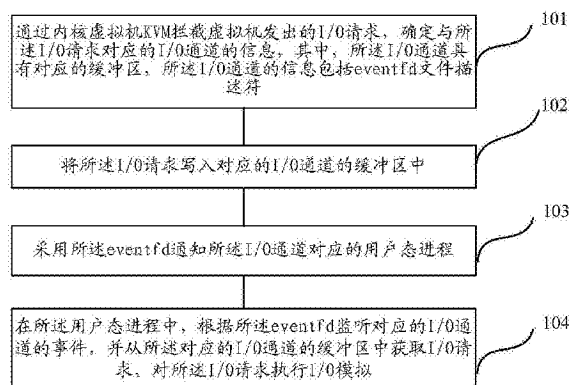
权利要求书3页 说明书14页 附图2页

(54) 发明名称

一种 KVM 虚拟化下处理 I/O 请求的方法和装置

(57) 摘要

本申请实施例提供了一种 KVM 虚拟化下处理 I/O 请求的方法和装置,其中所述方法包括:通过内核虚拟机 KVM 拦截虚拟机发出的 I/O 请求,确定与所述 I/O 请求对应的 I/O 通道的信息,其中,所述 I/O 通道具有对应的缓冲区,所述 I/O 通道的信息包括 eventfd 文件描述符;将所述 I/O 请求写入对应的 I/O 通道的缓冲区中;采用所述 eventfd 通知所述 I/O 通道对应的用户态进程;在所述用户态进程中,根据所述 eventfd 监听对应的 I/O 通道的事件,并从所述对应的 I/O 通道的缓冲区中获取 I/O 请求,对所述 I/O 请求执行 I/O 模拟,其中,所述用户态进程包括第一设备模拟进程以及与所述第一设备模拟进程独立的第二设备模拟进程。本申请可以在内核态中实现 I/O 请求的分发,针对单个虚拟机可以支持多个设备模拟进程,提高了 I/O 模拟的效率。



1. 一种KVM虚拟化下处理I/O请求的方法,其特征在于,所述的方法包括:

通过内核虚拟机KVM拦截虚拟机发出的I/O请求,确定与所述I/O请求对应的I/O通道的信息,其中,所述I/O通道具有对应的缓冲区,所述I/O通道的信息包括eventfd文件描述符;

将所述I/O请求写入对应的I/O通道的缓冲区中;

采用所述eventfd通知所述I/O通道对应的用户态进程;

在所述用户态进程中,根据所述eventfd监听对应的I/O通道的事件,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟,其中,所述用户态进程包括第一设备模拟进程以及与所述第一设备模拟进程独立的第二设备模拟进程。

2. 根据权利要求1所述的方法,其特征在于,所述在所述通过内核虚拟机KVM拦截虚拟机发出的I/O请求,确定与所述I/O请求对应的I/O通道的信息的步骤之前,还包括:

采用所述第一设备模拟进程创建虚拟机,获得所述虚拟机的文件描述符;

采用所述第一设备模拟进程,基于所述虚拟机的文件描述符,创建虚拟处理器vCPU,获得所述vCPU的文件描述符;

采用所述第二设备模拟进程,创建eventfd文件描述符;

采用所述第二设备模拟进程,基于所述虚拟机的文件描述符,创建I/O通道,并将所述I/O通道与所述eventfd文件描述符绑定,获得所述I/O通道的文件描述符;

采用所述第二设备模拟进程,对所述eventfd文件描述符进行监听;

采用所述第二设备模拟进程,基于所述I/O通道的文件描述符,为所述用户态进程映射所述I/O通道的缓冲区;

采用所述第二设备模拟进程,基于所述I/O通道的文件描述符,为所述I/O通道绑定I/O地址区间。

3. 根据权利要求2所述的方法,其特征在于,所述I/O请求包括I/O地址,所述通过内核虚拟机KVM拦截虚拟机发出的I/O请求,确定与所述I/O请求对应的I/O通道的信息的步骤包括:

通过内核虚拟机KVM及Intel VT-d硬件技术拦截虚拟机的vCPU线程发出的I/O请求;

从所述I/O请求中提取对应的I/O地址;

确定所述I/O地址所属的I/O地址区间;

通过所述I/O地址区间确定对应的I/O通道,并获取所述I/O通道对应的I/O通道的信息。

4. 根据权利要求3所述的方法,其特征在于,所述I/O通道的地址区间包括多个I/O模拟设备的地址子区间;

所述在所述用户态进程中,根据所述eventfd确定对应的I/O通道,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟的步骤包括:

在所述用户态进程中,根据所述eventfd确定对应的I/O通道;

从所述对应的I/O通道的缓冲区中获取I/O请求;

依据所述I/O请求中的I/O地址确定对应的I/O模拟设备;

将所述I/O请求分发至所述I/O模拟设备中,通过所述I/O模拟设备对所述I/O请求进行I/O模拟。

5. 根据权利要求3或4所述的方法,其特征在于,在所述在所述用户态进程中,根据所述

eventfd确定对应的I/O通道,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟的步骤之前,还包括:

挂起所述vCPU线程。

6.根据权利要求5所述的方法,其特征在于,在所述在所述用户态进程中,根据所述eventfd确定对应的I/O通道,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟的步骤之后,还包括:

唤醒所述挂起的vCPU线程。

7.根据权利要求4所述的方法,其特征在于,所述I/O请求包括读请求,所述将所述I/O请求分发至所述I/O模拟设备中,通过所述I/O模拟设备对所述I/O请求进行I/O模拟的步骤为:

将所述读请求分发至所述I/O模拟设备中,通过所述I/O模拟设备将所述读请求对应的数据写入所述I/O通道的缓冲区中。

8.根据权利要求7所述的方法,其特征在于,所述方法还包括:

若所述I/O请求为读请求,将所述读请求对应的数据从缓冲区中读出,读入到vCPU结构体中相应的成员变量中;

对所述数据进行对应的模拟处理。

9.一种KVM虚拟化下处理I/O请求的装置,其特征在于,所述的装置包括:

I/O通道确定模块,用于通过内核虚拟机KVM拦截虚拟机发出的I/O请求,确定与所述I/O请求对应的I/O通道的信息,其中,所述I/O通道具有对应的缓冲区,所述I/O通道的信息包括eventfd文件描述符;

缓冲区写入模块,用于将所述I/O请求写入对应的I/O通道的缓冲区中;

事件通知模块,用于采用所述eventfd通知所述I/O通道对应的用户态进程;

I/O模拟模块,用于在所述用户态进程中,根据所述eventfd监听对应的I/O通道的事件,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟,其中,所述用户态进程包括第一设备模拟进程以及与所述第一设备模拟进程独立的第二设备模拟进程。

10.根据权利要求9所述的装置,其特征在于,还包括:

虚拟机创建模块,用于采用所述第一设备模拟进程创建虚拟机,获得所述虚拟机的文件描述符;

vCPU创建模块,用于采用所述第一设备模拟进程,基于所述虚拟机的文件描述符,创建虚拟处理器vCPU,获得所述vCPU的文件描述符;

eventfd创建模块,用于采用所述第二设备模拟进程,创建eventfd文件描述符;

I/O通道创建模块,用于采用所述第二设备模拟进程,基于所述虚拟机的文件描述符,创建I/O通道,并将所述I/O通道与所述eventfd文件描述符绑定,获得所述I/O通道的文件描述符;

监听模块,用于采用所述第二设备模拟进程,对所述eventfd文件描述符进行监听;

缓冲区映射模块,用于采用所述第二设备模拟进程,基于所述I/O通道的文件描述符,为所述用户态进程映射所述I/O通道的缓冲区;

地址区间绑定模块,用于采用所述第二设备模拟进程,基于所述I/O通道的文件描述

符,为所述I/O通道绑定I/O地址区间。

11.根据权利要求10所述的装置,其特征在于,所述I/O请求包括I/O地址,所述I/O通道确定模块包括:

I/O请求拦截子模块,用于通过内核虚拟机KVM及Intel VT-d硬件技术拦截虚拟机的vCPU线程发出的I/O请求;

I/O地址提取子模块,用于从所述I/O请求中提取对应的I/O地址;

第一I/O通道确定子模块,用于确定所述I/O地址所属的I/O地址区间,并通过所述I/O地址区间确定对应的I/O通道,并获取所述I/O通道对应的I/O通道的信息。

12.根据权利要求11所述的装置,其特征在于,所述I/O通道的地址区间包括多个I/O模拟设备的地址子区间;

所述I/O模拟模块包括:

第二I/O通道确定子模块,用于在所述用户态进程中,根据所述eventfd确定对应的I/O通道;

I/O请求读取子模块,用于从所述对应的I/O通道的缓冲区中获取I/O请求;

I/O模拟设备确定子模块,用于依据所述I/O请求中的I/O地址确定对应的I/O模拟设备;

分发子模块,用于将所述I/O请求分发至所述I/O模拟设备中,通过所述I/O模拟设备对所述I/O请求进行I/O模拟。

13.根据权利要求11或12所述的装置,其特征在于,还包括:

挂起模块,用于在采用所述eventfd通知所述I/O通道对应的用户态进程以后,挂起所述vCPU线程。

14.根据权利要求13所述的装置,其特征在于,还包括:

唤醒模块,用于在对所述I/O请求执行I/O模拟以后,唤醒所述挂起的vCPU线程。

15.根据权利要求12所述的装置,其特征在于,所述I/O请求包括读请求,所述分发子模块还用于:

将所述读请求分发至所述I/O模拟设备中,通过所述I/O模拟设备将所述读请求对应的数据写入所述I/O通道的缓冲区中。

16.根据权利要求15所述的装置,其特征在于,所述装置还包括:

数据读出模块,用于在所述I/O请求为读请求时,将所述读请求对应的数据从缓冲区中读出,读入到vCPU结构体中相应的成员变量中,对所述数据进行对应的模拟处理。

一种KVM虚拟化下处理I/O请求的方法和装置

技术领域

[0001] 本申请涉及KVM虚拟化的技术领域,特别是涉及一种KVM虚拟化下处理I/O请求的方法和一种KVM虚拟化下处理I/O请求的装置。

背景技术

[0002] 虚拟机(Virtual Machine)是指通过软件模拟的、具有完整硬件系统功能的,运行在一个完全隔离环境中的完整计算机系统。通过虚拟机软件,可以在一台物理计算机上模拟出一台或多台虚拟的计算机。虚拟机可以像真正的计算机那样进行工作,例如可以安装操作系统、安装应用程序、访问网络资源等等。通常,虚拟机所在的物理计算机称为宿主机(Host),虚拟机自身称为虚拟机(Guest)。

[0003] KVM(Kernel-based Virtual Machine)即内核模式的虚拟机的简称,是一个开源的系统虚拟化模块,它使用Linux自身的调度器进行管理,KVM的虚拟化需要硬件支持(如Intel VT技术或者AMD VT技术)。

[0004] 在现有技术中,在KVM虚拟化中,当虚拟机发生I/O操作时,支持VMX(Virtual Machine eXtensions,是intel实现的x86指令集虚拟化扩展)或者SVM(Secure Virtual Machine,是amd实现的x86指令集的虚拟化扩展)扩展的CPU就会停止执行,退出非根模式,进入根模式执行KVM代码。KVM根据退出的原因得知虚拟机需要进行I/O操作,于是从内核态退出至用户态的QEMU进程。

[0005] QEMU进程在初始化的时候映射了一块KVM的空间,KVM在退出前把I/O请求放在其中,QEMU从中拿到I/O请求之后将其分发给相应的I/O模拟设备,I/O模拟设备执行I/O模拟以后,QEMU重新调用IOCTL进入KVM内核态,KVM稍作处理后进入非根模式继续运行vCPU(虚拟处理器)。

[0006] 然而,发明人在实施上述I/O请求的模拟过程中,发现如下问题:

[0007] 1、模拟vCPU的时候QEMU通过一个vCPU的IOCTL进入到KVM内核态,发生I/O请求的时候要退出内核态回到QEMU用户态的VCPU线程进行模拟,因此,I/O模拟必须和CPU模拟在同一个进程中,无法支持多个设备模拟进程。

[0008] 2、KVM已有的基于eventfd的I/O通知机制只能应用于写请求,且对所写数据不关心,只关心所写地址的场景,例如Virtio设备的队列通知寄存器写访问。这种机制不能传递I/O请求访问的实际地址和长度,也无法将读操作得到的数据传递给KVM。

[0009] 3、如果要在不同线程中实现CPU模拟和设备I/O模拟,现在缺乏一种机制来同步IO模拟线程和VCPU线程。

发明内容

[0010] 鉴于上述问题,提出了本申请实施例以便提供一种克服上述问题或者至少部分地解决上述问题的一种KVM虚拟化下处理I/O请求的方法和相应的一种KVM虚拟化下处理I/O请求的装置。

[0011] 为了解决上述问题,本申请公开了一种KVM虚拟化下处理I/O请求的方法,所述的方法包括:

[0012] 通过内核虚拟机KVM拦截虚拟机发出的I/O请求,确定与所述I/O请求对应的I/O通道的信息,其中,所述I/O通道具有对应的缓冲区,所述I/O通道的信息包括eventfd文件描述符;

[0013] 将所述I/O请求写入对应的I/O通道的缓冲区中;

[0014] 采用所述eventfd通知所述I/O通道对应的用户态进程;

[0015] 在所述用户态进程中,根据所述eventfd监听对应的I/O通道的事件,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟,其中,所述用户态进程包括第一设备模拟进程以及与所述第一设备模拟进程独立的第二设备模拟进程。

[0016] 优选地,所述在所述通过内核虚拟机KVM拦截虚拟机发出的I/O请求,确定与所述I/O请求对应的I/O通道的信息的步骤之前,还包括:

[0017] 采用所述第一设备模拟进程创建虚拟机,获得所述虚拟机的文件描述符;

[0018] 采用所述第一设备模拟进程,基于所述虚拟机的文件描述符,创建虚拟处理器vCPU,获得所述vCPU的文件描述符;

[0019] 采用所述第二设备模拟进程,创建eventfd文件描述符;

[0020] 采用所述第二设备模拟进程,基于所述虚拟机的文件描述符,创建I/O通道,并将所述I/O通道与所述eventfd文件描述符绑定,获得所述I/O通道的文件描述符;

[0021] 采用所述第二设备模拟进程,对所述eventfd文件描述符进行监听;

[0022] 采用所述第二设备模拟进程,基于所述I/O通道的文件描述符,为所述用户态进程映射所述I/O通道的缓冲区;

[0023] 采用所述第二设备模拟进程,基于所述I/O通道的文件描述符,为所述I/O通道绑定I/O地址区间。

[0024] 优选地,所述I/O请求包括I/O地址,所述通过内核虚拟机KVM拦截虚拟机发出的I/O请求,确定与所述I/O请求对应的I/O通道的信息的步骤包括:

[0025] 通过内核虚拟机KVM及Intel VT-d硬件技术拦截虚拟机的vCPU线程发出的I/O请求;

[0026] 从所述I/O请求中提取对应的I/O地址;

[0027] 确定所述I/O地址所属的I/O地址区间;

[0028] 通过所述I/O地址区间确定对应的I/O通道,并获取所述I/O通道对应的I/O通道的信息。

[0029] 优选地,所述I/O通道的地址区间包括多个I/O模拟设备的地址子区间;

[0030] 所述在所述用户态进程中,根据所述eventfd确定对应的I/O通道,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟的步骤包括:

[0031] 在所述用户态进程中,根据所述eventfd确定对应的I/O通道;

[0032] 从所述对应的I/O通道的缓冲区中获取I/O请求;

[0033] 依据所述I/O请求中的I/O地址确定对应的I/O模拟设备;

[0034] 将所述I/O请求分发至所述I/O模拟设备中,通过所述I/O模拟设备对所述I/O请求进行I/O模拟。

[0035] 优选地,在所述在所述用户态进程中,根据所述eventfd确定对应的I/O通道,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟的步骤之前,还包括:

[0036] 挂起所述vCPU线程。

[0037] 优选地,在所述在所述用户态进程中,根据所述eventfd确定对应的I/O通道,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟的步骤之后,还包括:

[0038] 唤醒所述挂起的vCPU线程。

[0039] 优选地,所述I/O请求包括读请求,所述将所述I/O请求分发至所述I/O模拟设备中,通过所述I/O模拟设备对所述I/O请求进行I/O模拟的步骤为:

[0040] 将所述读请求分发至所述I/O模拟设备中,通过所述I/O模拟设备将所述读请求对应的数据写入所述I/O通道的缓冲区中。

[0041] 优选地,所述方法还包括:

[0042] 若所述I/O请求为读请求,将所述读请求对应的数据从缓冲区中读出,读入到vCPU结构体中相应的成员变量中;

[0043] 对所述数据进行对应的模拟处理。

[0044] 本申请还公开了一种KVM虚拟化下处理I/O请求的装置,其特征在于,所述的装置包括:

[0045] I/O通道确定模块,用于通过内核虚拟机KVM拦截虚拟机发出的I/O请求,确定与所述I/O请求对应的I/O通道的信息,其中,所述I/O通道具有对应的缓冲区,所述I/O通道的信息包括eventfd文件描述符;

[0046] 缓冲区写入模块,用于将所述I/O请求写入对应的I/O通道的缓冲区中;

[0047] 事件通知模块,用于采用所述eventfd通知所述I/O通道对应的用户态进程;

[0048] I/O模拟模块,用于在所述用户态进程中,根据所述eventfd监听对应的I/O通道的事件,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟,其中,所述用户态进程包括第一设备模拟进程以及与所述第一设备模拟进程独立的第二设备模拟进程。

[0049] 优选地,所述装置还包括:

[0050] 虚拟机创建模块,用于采用所述第一设备模拟进程创建虚拟机,获得所述虚拟机的文件描述符;

[0051] vCPU创建模块,用于采用所述第一设备模拟进程,基于所述虚拟机的文件描述符,创建虚拟处理器vCPU,获得所述vCPU的文件描述符;

[0052] eventfd创建模块,用于采用所述第二设备模拟进程,创建eventfd文件描述符;

[0053] I/O通道创建模块,用于采用所述第二设备模拟进程,基于所述虚拟机的文件描述符,创建I/O通道,并将所述I/O通道与所述eventfd文件描述符绑定,获得所述I/O通道的文件描述符;

[0054] 监听模块,用于采用所述第二设备模拟进程,对所述eventfd文件描述符进行监听;

[0055] 缓冲区映射模块,用于采用所述第二设备模拟进程,基于所述I/O通道的文件描述

符,为所述用户态进程映射所述I/O通道的缓冲区;

[0056] 地址区间绑定模块,用于采用所述第二设备模拟进程,基于所述I/O通道的文件描述符,为所述I/O通道绑定I/O地址区间。

[0057] 优选地,所述I/O请求包括I/O地址,所述I/O通道确定模块包括:

[0058] I/O请求拦截子模块,用于通过内核虚拟机KVM及Intel VT-d硬件技术拦截虚拟机的vCPU线程发出的I/O请求;

[0059] I/O地址提取子模块,用于从所述I/O请求中提取对应的I/O地址;

[0060] 第一I/O通道确定子模块,用于确定所述I/O地址所属的I/O地址区间,并通过所述I/O地址区间确定对应的I/O通道,并获取所述I/O通道对应的I/O通道的信息。

[0061] 优选地,所述I/O通道的地址区间包括多个I/O模拟设备的地址子区间;

[0062] 所述I/O模拟模块包括:

[0063] 第二I/O通道确定子模块,用于在所述用户态进程中,根据所述eventfd确定对应的I/O通道;

[0064] I/O请求读取子模块,用于从所述对应的I/O通道的缓冲区中获取I/O请求;

[0065] I/O模拟设备确定子模块,用于依据所述I/O请求中的I/O地址确定对应的I/O模拟设备;

[0066] 分发子模块,用于将所述I/O请求分发至所述I/O模拟设备中,通过所述I/O模拟设备对所述I/O请求进行I/O模拟。

[0067] 优选地,所述装置还包括:

[0068] 挂起模块,用于在采用所述eventfd通知所述I/O通道对应的用户态进程以后,挂起所述vCPU线程。

[0069] 优选地,所述装置还包括:

[0070] 唤醒模块,用于在对所述I/O请求执行I/O模拟以后,唤醒所述挂起的vCPU线程。

[0071] 优选地,所述I/O请求包括读请求,所述分发子模块还用于:

[0072] 将所述读请求分发至所述I/O模拟设备中,通过所述I/O模拟设备将所述读请求对应的数据写入所述I/O通道的缓冲区中。

[0073] 优选地,所述装置还包括:

[0074] 数据读出模块,用于在所述I/O请求为读请求时,将所述读请求对应的数据从缓冲区中读出,读入到vCPU结构体中相应的成员变量中,对所述数据进行对应的模拟处理。

[0075] 本申请实施例包括以下优点:

[0076] 在本申请实施例中,可以通过I/O通道来传递I/O请求,当KVM拦截虚拟机发出的I/O请求以后,可以确定该I/O请求对应的I/O通道,并将I/O请求写入对应的I/O通道的缓冲区中,随后通过eventfd通知I/O通道对应的用户态进程,用户态进程监听到eventfd事件通知以后,从该eventfd对应的I/O通道的缓冲区中获取I/O请求,并对该I/O请求执行I/O模拟。本申请可以在内核态中实现I/O请求的分发,针对单个虚拟机可以支持多个设备模拟进程,提高了I/O模拟的效率。

附图说明

[0077] 图1是本申请的一种KVM虚拟化下分发I/O请求的方法实施例一的步骤流程图;

[0078] 图2是本申请的一种KVM虚拟化下分发I/O请求的方法实施例二的步骤流程图；

[0079] 图3是本申请的一种KVM虚拟化下分发I/O请求的方法实施例二中的KVM初始化流程的步骤流程图；

[0080] 图4是本申请的种KVM虚拟化下处理I/O请求的装置实施例的结构框图。

具体实施方式

[0081] 为使本申请的上述目的、特征和优点能够更加明显易懂，下面结合附图和具体实施方式对本申请作进一步详细的说明。

[0082] 参照图1，示出了本申请的一种KVM虚拟化下分发I/O请求的方法实施例一的步骤流程图，具体可以包括如下步骤：

[0083] 步骤101，通过内核虚拟机KVM拦截虚拟机发出的I/O请求，确定与所述I/O请求对应的I/O通道的信息，其中，所述I/O通道具有对应的缓冲区，所述I/O通道的信息包括eventfd文件描述符；

[0084] 步骤102，将所述I/O请求写入对应的I/O通道的缓冲区中；

[0085] 步骤103，采用所述eventfd通知所述I/O通道对应的用户态进程；

[0086] 步骤104，在所述用户态进程中，根据所述eventfd监听对应的I/O通道的事件，并从所述对应的I/O通道的缓冲区中获取I/O请求，对所述I/O请求执行I/O模拟。

[0087] 其中，所述用户态进程包括第一设备模拟进程以及与所述第一设备模拟进程独立的第二设备模拟进程。

[0088] 在本申请实施例中，可以通过I/O通道来传递I/O请求，当KVM拦截虚拟机发出的I/O请求以后，可以确定该I/O请求对应的I/O通道，并将I/O请求写入对应的I/O通道的缓冲区中，随后通过eventfd通知I/O通道对应的用户态进程，用户态进程监听到eventfd事件通知以后，从该eventfd对应的I/O通道的缓冲区中读取I/O请求，并对该I/O请求执行I/O模拟。本申请可以在内核态中实现I/O请求的分发，针对单个虚拟机可以支持多个设备模拟进程，提高了I/O模拟的效率。

[0089] 参照图2，示出了本申请的一种KVM虚拟化下处理I/O请求的方法实施例二的步骤流程图。其中，KVM(Kernel-based Virtual Machine，内核模式的虚拟机)是一个基于Linux环境的开源虚拟化解决方案。KVM的思想是在Linux内核的基础上添加虚拟机管理模块，重用Linux内核中已经完善的进程调度、内存管理、I/O管理等部分，使之成为一个可以支持运行虚拟机的超级管理程序Hypervisor。

[0090] KVM是基于硬件辅助虚拟化技术(例如：Intel的VT-x或者AMD-V)的全虚拟化解决方案，虚拟机操作系统能够不经过修改直接在KVM的虚拟机中运行，每一台虚拟机能够享有独立的虚拟硬件资源：网卡、磁盘、图形适配器等。

[0091] KVM可以作为单独的模块编译进内核中，也可以作为内核模块在Linux系统启动完成之后加载。

[0092] KVM的API是通过/dev/kvm设备进行访问的。/dev/kvm是一个字符型设备，其可以使用常见的系统调用如open、close、IOCTL等指令进行操作。因为KVM的字符型设备的实现函数中，没有包含write、read等操作，所以所有对KVM的操作都是通过IOCTL发送相应的控制字实现的。

[0093] KVM所提供的用户空间API从功能上划分,可以分为三种类型,分别为system指令、VM指令以及vCPU指令。其中,system指令针对虚拟化系统的全局性参数设置和用于虚拟机创建等控制操作;VM指令针对具体的VM虚拟机进行控制,如进行内存设置、创建vCPU等;vCPU指令针对具体的vCPU进行参数设置(MRU寄存器读写、中断控制等)。

[0094] 在发明实施例中,可以通过不同的用户态模拟器来控制用户空间,从而可以在不同的用户态进程中模拟同一台虚拟机的设备I/O,支持针对单个虚拟机的多个设备模拟进程。该不同的用户态进程可以包括第一设备模拟进程以及第二设备模拟进程,作为本申请实施例的一种优选示例,第一设备模拟进程可以为QEMU(全称Quick Emulator)进程,所述第二设备模拟进程可以为独立于QEMU进程的I/O设备模拟进程,为方便后续的描述,可以将该第二设备模拟进程称为DEMU(Dedicated Emulator)进程或DEMU。

[0095] 其中,QEMU是一款开源的模拟器及虚拟机监管器(Virtual Machine Monitor, VMM),用户可以通过不同Linux发行版所带有的软件包管理器来安装QEMU。QEMU在主机用户态模拟虚拟机的硬件设备,QEMU作为系统模拟器时,可以模拟出一台能够独立运行操作系统的虚拟机,每个虚拟机对应主机(Host)中的一个QEMU进程,而虚拟机的vCPU对应QEMU进程的一个线程。

[0096] 在具体实现中,可以通过执行QEMU和DEMU的二进制可执行文件来创建QEMU进程与DEMU进程,并可以通过脚本或者libvirt工具启动这两个进程。

[0097] 应用于本申请实施例,在对I/O请求进行处理之前,首先执行初始化流程。在平台相关的KVM模块中通过module_init宏正式进入KVM的初始化阶段,并且执行相关的硬件初始化准备。进入kvm_main.c中的kvm_init函数进行正式的初始化工作。

[0098] 参考图3的KVM初始化流程的步骤流程图,所述初始化流程至少可以包括如下步骤:

[0099] 子步骤S1,采用所述第一设备模拟进程创建虚拟机,获得所述虚拟机的文件描述符;

[0100] 具体的,KVM同用户空间进程的交互接口为/dev/kvm,对于KVM API的操作是从打开/dev/kvm设备文件开始的,通过使用系统调用open方法之后,可以获得针对kvm subsystem的一个fd文件描述符。然后通过IOCTL系统指令针对该文件描述符进行进一步的操作。

[0101] IOCTL系统指令可以包括KVM_CREATE_VM指令,通过KVM_CREATE_VM指令,即可创建一个虚拟机VM。通过调用anon_inode_getfd可以获得一fd文件描述符,作为虚拟机的文件描述符,即vm_fd,该vm_fd指向内核空间中该新创建的虚拟机,然后根据该描述符来控制虚拟机的行为。

[0102] vm IOCTL系统调用实现了对虚拟机的控制。vm IOCTL控制指令的参数大多需要从KVM_CREATE_VM中返回的vm_fd来进行操作,涉及的操作主要针对该虚拟机进行控制,如配置内存、配置vCPU等。

[0103] 针对VM的文件操作中,提供了IOCTL和mmap两个操作函数,其中mmap对应着客户操作系统GUEST OS的物理地址,可以直接对GUEST OS的地址空间进行读/写,IOCTL则用于发送KVM的控制字。

[0104] 应用于本申请实施例,还可以对该创建的虚拟机设置对应的通用唯一识别码

UUID,以通过该UUID识别虚拟机。

[0105] 虚拟机的创建过程实质为kvm结构体的创建和初始化过程,在创建虚拟机时,kvm_dev_IOCTL_create_vm函数通过调用kvm_create_vm函数对KVM结构体进行创建。

[0106] KVM结构体保存了虚拟机运行的上下文及其他相关状态,例如,可以包括vCPU(虚拟处理器,虚拟机内的CPU)、内存、APIC(Advanced Programmable Interrupt Controller,高级可编程中断控制器)、IRQ(Interrupt Request,中断请求)、MMU(Memory Management Unit,内存管理单元)、Event事件管理等信息,应用于本申请实施例,该KVM结构体还可以包括虚拟机的UUID。

[0107] KVM结构体的初始化过程,可以包括:初始化KVM的memslot结构体、Bus总线结构体信息、scru读/写锁信息、eventfd事件通知信息、mmu内存管理结构体信息等。

[0108] 虚拟机的KVM结构体创建完成以后,可以将其加入到全局链表vm_list中。

[0109] 在本申请实施例中,可以使用QEMU进程完成虚拟机的创建。

[0110] 步骤S2,采用所述第一设备模拟进程,基于所述虚拟机的文件描述符,创建虚拟处理器vCPU,获得所述vCPU的文件描述符;

[0111] 具体的,全新创建的虚拟机没有vCPU,需要通过后续的IOCTL指令进行配置。在获得vm_fd之后,通过IOCTL调用KVM_CREATE_vCPU指令,可以对该vm_fd对应的虚拟机创建vCPU,其入口函数地址在kvm_vm_IOCTL函数中,通过switch之后,程序流程将选择进入kvm_vm_IOCTL_create_vCPU函数中进行处理,返回该vCPU对应的文件描述符fd,即vCPU_fd,在KVM虚拟化环境中,硬件虚拟化使用vCPU_fd来描述vCPU。

[0112] 在KVM中,vCPU对应的数据结构体为kvm_vCPU,vCPU的创建过程实质为kvm_vCPU结构体的创建和初始化过程,可以包括如下过程:调用kvm_arch_vCPU_create函数创建一个kvm_vCPU结构体;调用kvm_arch_vCPU_setup函数对kvm_vCPU中的数据结构进行初始化;创建当前vCPU对应的文件描述符vCPU_fd,并且将kvm_vCPU添加入KVM的vCPU数组中。

[0113] 在本申请实施例中,可以使用QEMU进程完成vCPU的创建,每个vCPU对应一个vCPU线程。

[0114] 步骤S3,采用所述第二设备模拟进程,创建eventfd文件描述符;

[0115] 应用于本申请实施例,可以采用DEMU进程调用标准的Linux系统API eventfd()来创建eventfd文件描述符。具体来说,eventfd()是Linux系统提供的系统接口,函数原型是int eventfd(unsigned int initval,int flags),调用这个函数用户态进程可以获取一个eventfd文件描述符,同时在内核中创建一个eventfd对象用于实现等待和通知的机制,主要用于用户态互相通知,和内核态向用户态的通知。

[0116] eventfd对象中可以包含一个64位的整形变量作为计数器,函数原型中传入的参数initval用于初始化这个计数器。eventfd文件描述符用于用户态引用eventfd对象,用户态进程可以对eventfd文件描述符进行read,write,poll,select操作。

[0117] 步骤S4,采用所述第二设备模拟进程,基于所述虚拟机文件描述符,创建I/O通道,并将所述I/O通道与所述eventfd文件描述符绑定,获得所述I/O通道的文件描述符;

[0118] 应用于本申请实施例,还可以采用DEMU进程来创建虚拟机的一个或多个I/O通道。

[0119] 具体来说,DEMU进程可以获得当前虚拟机的UUID,然后在全局链表vm_list中查找该虚拟机的UUID,若查找成功,则获得对应的KVM结构体,从而获得虚拟机的文件描述符vm_

fd,之后,可以通过vm_fd新增的IOCTL指令(例如,KVM_CREATE_IOCH指令)对该vm_fd对应的虚拟机创建I/O通道。创建I/O通道的时候,会生成I/O通道与eventfd的绑定关系,随后返回I/O通道的文件描述符,即I/O_fd,I/O通道的文件描述符提供IOCLT指令给用户态进程,每个vm_fd可挂载多个I/O_fd。

[0120] 在KVM中,I/O通道的创建过程实质为I/O通道的结构体的创建和初始化过程,I/O通道的结构体可以包括:一个eventfd,用于内核态向用户态的通知;一组存放I/O请求的缓冲区及其保护锁;一个保存所有此通道负责传递的I/O地址空间的链表。

[0121] 应用于本申请实施例,在虚拟环境中,每个虚拟机对应的KVM结构体还可以新增一个链表,用于保存所有注册在该虚拟机上的I/O通道。

[0122] 需要说明的是,每个I/O模拟进程仅需要且对应一个I/O通道。

[0123] 子步骤S5,采用所述第二设备模拟进程,对所述eventfd文件描述符进行监听。

[0124] 在具体实现中,可以在进程主循环中对eventfd文件描述符使用poll或者select方法,一旦发现该eventfd文件描述符有待处理的I/O,就调用相关的处理函数,以实现对该eventfd的监听。

[0125] 步骤S6,采用所述第二设备模拟进程,基于所述I/O通道的文件描述符,为所述用户态进程映射所述I/O通道的缓冲区;

[0126] 在具体实现中,KVM会在内核态为全新创建的I/O通道分配一段缓冲区,随后,用户态程序可以对I/O通道的文件描述符使用mmap()系统调用,映射该I/O通道的缓冲区,则用户态进程可以访问该缓冲区。

[0127] 所述缓冲区还具有保护锁,以保证对该缓冲区读写访问互斥性。

[0128] 步骤S7,采用所述第二设备模拟进程,基于所述I/O通道的文件描述符,为所述I/O通道绑定I/O地址区间。

[0129] 应用本申请实施例,还可以采用DEMU进程通过I/O通道的文件描述符为需要监听的I/O通道绑定I/O地址区间,不同的I/O通道具有不同的I/O地址区间,例如,针对I/O地址区间1~1000,可以将1~300绑定给I/O通道A,将301~1000绑定给I/O通道B。

[0130] 在具体实现中,该I/O通道的地址区间还可以包括多个地址子区间,每个地址子区间对应一个I/O模拟设备(一个IO模拟进程可以模拟多个IO模拟设备),每个I/O模拟设备只模拟落在其地址区间范围内的请求。例如,I/O通道A的地址区间为1~300,其中,1~100可以对应I/O模拟设备1,101~200可以对应I/O模拟设备2,201~300可以对应I/O模拟设备3。

[0131] 在本申请实施例中,在原有的QEMU进程的基础上,新增独立的DEMU进程,两者作为用户态模拟器,能够共享虚拟机的一部分底层数据结构。

[0132] 在完成初始化流程以后,如图2所示,本申请实施例具体可以包括如下步骤:

[0133] 步骤201,通过内核虚拟机KVM拦截虚拟机的vCPU线程发出的I/O请求,确定与所述I/O请求对应的I/O通道的信息;

[0134] KVM可以拦截虚拟机的I/O操作,并根据拦截的I/O操作确定对应的I/O通道的信息。

[0135] 在本申请实施例的一种优选实施例中,步骤201可以包括如下子步骤:

[0136] 子步骤S11,通过内核虚拟机KVM及Intel VT-d硬件技术拦截虚拟机的vCPU线程发出的I/O请求;

[0137] 在具体实现中,KVM在内核空间非根模式下运行vCPU线程,当vCPU线程发起I/O请求时,该敏感指令将会触发Intel VT-d硬件技术的VM exit操作,进入根模式下KVM相应的处理函数。

[0138] 具体的,当虚拟机运行时(虚拟机的运行通过/dev/kvm设备IOCTL vCPU接口的KVM_RUN指令实现),触发VM Entry,使虚拟机进入客户Guest模式,即non root模式。而当在虚拟机中(Guest模式)执行特权指令或者外部事件,比如I/O访问,对控制寄存器的操作,MSR的读写数据包到达等时,可以触发(由硬件触发)VM Exit,使当前CPU从Guest模式(非根non-root模式)退出,切换到根root模式,当前CPU的控制权随之转交给KVM,由KVM进行相应的处理。

[0139] KVM取得控制权以后,通过读取VMCS中VM_EXIT_REASON字段得到引起VM Exit的原因,KVM根据退出的原因得知虚拟机需要进行I/O操作。

[0140] 子步骤S12,从所述I/O请求中提取对应的I/O地址;

[0141] I/O请求中可以携带发出I/O请求的虚拟机的地址信息,当获得I/O请求以后,可以从该I/O请求中提取对应的I/O地址。

[0142] 子步骤S13,确定所述I/O地址所属的I/O地址区间;

[0143] 子步骤S14,通过所述I/O地址区间确定对应的I/O通道,并获取所述I/O通道对应的I/O通道的信息。

[0144] 由于I/O通道具有I/O地址区间,因此可以首先判断该I/O请求的I/O地址属于哪个地址区间,然后根据匹配的地址区间确定对应的I/O通道,并确定该I/O通道对应的缓冲区、eventfd文件描述符等信息。

[0145] 步骤202,将所述I/O请求写入对应的I/O通道的缓冲区中;

[0146] 确定该I/O请求对应的I/O通道通道以后,KVM可以将该I/O请求写入该对应的I/O通道的缓冲区中,从而在KVM中完成I/O请求的分发。

[0147] 步骤203,采用所述eventfd通知所述I/O通道对应的用户态进程;

[0148] 在具体实现中,在内核空间中将I/O请求写入对应的I/O通道的缓冲区时,用户空间对此并不知情,因此需要内核空间通过eventfd向用户空间发出通知,用户态进程通过监听该eventfd从而获知其对应的I/O通道的缓冲区中写入了I/O请求。

[0149] 步骤204,挂起所述vCPU线程;

[0150] 应用于本申请实施例,由于I/O请求通过I/O通道上传到I/O通道的缓冲区中,由I/O模拟线程进行I/O模拟,而I/O模拟线程与vCPU线程可以处于不同的进程中,因此发生I/O请求的时候无需退出vCPU线程,此时只需要挂起vCPU线程,并等待用户态进程的完成I/O请求后再将其唤醒。

[0151] 步骤205,在所述用户态进程中,根据所述eventfd监听对应的I/O通道的事件,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟;

[0152] 当内核态向用户态发出eventfd事件通知时,用户态进程可以监听到eventfd事件通知,相应的处理函数被调用,通过用户态进程进行I/O模拟,其中,I/O模拟主要指的是针对虚拟机进行的I/O操作,通过软件的模拟行为使其获得一个对应的处理结果。

[0153] 在本申请实施例的一种优选实施例中,步骤205可以包括如下子步骤:

[0154] 子步骤S21,在所述用户态进程中,根据所述eventfd确定对应的I/O通道;

[0155] 由于eventfd与I/O通道是绑定的,当用户态进程监听到eventfd进程注册的函数被调用以后,可以根据eventfd与I/O通道的对应关系,确定与eventfd对应的I/O通道。

[0156] 子步骤S22,从所述对应的I/O通道的缓冲区中获取I/O请求;

[0157] 当确定I/O通道以后,可以进一步从已经映射的该I/O通道的缓冲区中读出I/O请求。

[0158] 子步骤S23,依据所述I/O请求中的I/O地址确定对应的I/O模拟设备;

[0159] 子步骤S24,将所述I/O请求分发至所述I/O模拟设备中,通过所述I/O模拟设备对所述I/O请求进行I/O模拟。

[0160] 由于I/O通道的地址空间中包含一个或多个I/O模拟设备的地址子区间,当用户态进程获得I/O请求以后,可以根据该I/O请求中携带的I/O地址,确定该I/O地址所属的地址子区间,并根据该地址子区间确定对应的I/O模拟设备,以及将该I/O请求分发至该对应的I/O模拟设备中,以完成I/O请求的进一步二次分发。

[0161] I/O模拟设备获得I/O请求以后,对该I/O请求进行I/O模拟。

[0162] 在一种实施方式中,若该I/O请求为读请求,则该请求对应的I/O模拟设备可以将该读请求对应的数据写入I/O通道的缓冲区中,从而完成用户空间中的I/O模拟。

[0163] 在另一种实施方式中,若该I/O请求为写请求,则该请求对应的I/O模拟设备可以将该写请求对应的数据写入对应的位置中,例如写入到了主机中的一个镜像文件中,从而完成用户空间中的I/O模拟。

[0164] 步骤206,唤醒所述挂起的vCPU线程。

[0165] I/O模拟完成后,用户态进程调用I/O通道的文件描述符中的新增IOCTL指令进入KVM内核态,唤醒该挂起的vCPU线程,以继续运行vCPU线程,达到vCPU线程与I/O模拟线程的同步,也就是说,用户态向内核态的通知是通过对vCPU线程的挂起与唤醒来实现的。

[0166] 在一种实施方式中,若该I/O请求为读请求,在唤醒vCPU线程之后,还可以将该读请求对应的数据从缓冲区中读出,读入到vCPU结构体中相应的成员变量中,并通过原有的I/O模拟代码进行后续的模拟处理。

[0167] eventfd的使用结合缓冲区的使用,可以在缓冲区中传递I/O请求访问的实际地址和长度,并将读操作得到的数据通过缓冲区传递给KVM,克服现有技术的缺陷。

[0168] 在对I/O请求处理完成以后,可以通过硬件指令(如VMLAUNCH),触发VM Enter,重新进入到Guest模式,从而进入虚拟机运行环境中继续运行。

[0169] 为了使本领域技术人员能够更好地理解本申请实施例,以下通过一个具体实例对本申请实施例加以示例性说明,但应该理解的是,本申请实施例并不限于此。

[0170] 虚拟机在运行时,设备驱动会访问虚拟VGA设备的MMIO控制寄存器来读取模拟设备的状态。该寄存器映射的地址访问触发EPT页错误,CPU退出非根模式进入根模式,执行KVM代码。

[0171] KVM判断退出原因得知是I/O请求,根据I/O请求查找对应的I/O通道,将I/O请求拷贝到I/O通道相应的缓冲区中,然后用eventfd发出通知后vCPU线程进入睡眠。

[0172] QEMU主循环发现eventfd中的事件通知,找到相应的I/O通道并从相应的缓冲区中读出I/O请求,发现该请求地址处于客户机物理地址布局中虚拟VGA设备的BAR1区间,于是调用VGA设备相应的处理函数对I/O请求进行模拟,把该寄存器的值写回到缓冲区中,然后

通过IOCTL指令唤醒睡眠的vCPU线程。

[0173] vCPU线程被唤醒后从缓冲区中读出寄存器值数据,并将其更新到vCPU上下文的相应位置,然后进入非根模式继续执行vCPU上下文。

[0174] 驱动代码处的读寄存器操作全部完成模拟。

[0175] 本申请实施例在KVM虚拟化下实现独立于QEMU的设备模拟程序提供了基础,解决了一些厂商特有的设备模拟需求。

[0176] 另外,本申请实施例在KVM内核模块中实现I/O请求的分发,并向用户态提供控制接口,从而可以在不同的进程中模拟同一台虚拟机的设备I/O。

[0177] 需要说明的是,对于方法实施例,为了简单描述,故将其都表述为一系列的动作组合,但是本领域技术人员应该知悉,本申请实施例并不受所描述的动作顺序的限制,因为依据本申请实施例,某些步骤可以采用其他顺序或者同时进行。其次,本领域技术人员也应该知悉,说明书中所描述的实施例均属于优选实施例,所涉及的动作并不一定是本申请实施例所必须的。

[0178] 参照图4,示出了本申请的一种KVM虚拟化下处理I/O请求的装置实施例的结构框图,具体可以包括如下模块:

[0179] I/O通道确定模块401,用于通过内核虚拟机KVM拦截虚拟机发出的I/O请求,确定与所述I/O请求对应的I/O通道的信息,其中,所述I/O通道具有对应的缓冲区,所述I/O通道的信息包括eventfd文件描述符;

[0180] 缓冲区写入模块402,用于将所述I/O请求写入对应的I/O通道的缓冲区中;

[0181] 事件通知模块403,用于采用所述eventfd通知所述I/O通道对应的用户态进程;

[0182] I/O模拟模块404,用于在所述用户态进程中,根据所述eventfd监听对应的I/O通道的事件,并从所述对应的I/O通道的缓冲区中获取I/O请求,对所述I/O请求执行I/O模拟。

[0183] 其中,所述用户态进程包括第一设备模拟进程以及与所述第一设备模拟进程独立的第二设备模拟进程。

[0184] 在本申请实施例的一种优选实施例中,所述装置还可以包括:

[0185] 虚拟机创建模块,用于采用所述第一设备模拟进程创建虚拟机,获得所述虚拟机的文件描述符;

[0186] vCPU创建模块,用于采用所述第一设备模拟进程,基于所述虚拟机的文件描述符,创建虚拟处理器vCPU,获得所述vCPU的文件描述符;

[0187] eventfd创建模块,用于采用所述第二设备模拟进程,创建eventfd文件描述符;

[0188] I/O通道创建模块,用于采用所述第二设备模拟进程,基于所述虚拟机的文件描述符,创建I/O通道,并将所述I/O通道与所述eventfd文件描述符绑定,获得所述I/O通道的文件描述符;

[0189] 监听模块,用于对所述eventfd文件描述符进行监听;

[0190] 缓冲区映射模块,用于采用所述第二设备模拟进程,基于所述I/O通道的文件描述符,为所述用户态进程映射所述I/O通道的缓冲区;

[0191] 地址区间绑定模块,用于采用所述第二设备模拟进程,基于所述I/O通道的文件描述符,为所述I/O通道绑定I/O地址区间。

[0192] 在本申请实施例的一种优选实施例中,所述I/O请求包括I/O地址,所述I/O通道确

定模块401可以包括如下子模块：

[0193] I/O请求拦截子模块，用于通过内核虚拟机KVM及Intel VT-d硬件技术拦截虚拟机的vCPU线程发出的I/O请求；

[0194] I/O地址提取子模块，用于从所述I/O请求中提取对应的I/O地址；

[0195] 第一I/O通道确定子模块，用于确定所述I/O地址所属的I/O地址区间，并通过所述I/O地址区间确定对应的I/O通道，并获取所述I/O通道对应的I/O通道的信息。

[0196] 在本申请实施例的一种优选实施例中，所述I/O通道的地址区间包括多个I/O模拟设备的地址子区间；

[0197] 所述I/O模拟模块404可以包括如下子模块：

[0198] 第二I/O通道确定子模块，用于在所述用户态进程中，根据所述eventfd确定对应的I/O通道；

[0199] I/O请求读取子模块，用于从所述对应的I/O通道的缓冲区中获取I/O请求；

[0200] I/O模拟设备确定子模块，用于依据所述I/O请求中的I/O地址确定对应的I/O模拟设备；

[0201] 分发子模块，用于将所述I/O请求分发至所述I/O模拟设备中，通过所述I/O模拟设备对所述I/O请求进行I/O模拟。

[0202] 在本申请实施例的一种优选实施例中，所述装置还包括：

[0203] 挂起模块，用于在采用所述eventfd通知所述I/O通道对应的用户态进程以后，挂起所述vCPU线程。

[0204] 在本申请实施例的一种优选实施例中，所述装置还包括：

[0205] 唤醒模块，用于在对所述I/O请求执行I/O模拟以后，唤醒所述挂起的vCPU线程。

[0206] 在本申请实施例的一种优选实施例中，所述I/O请求包括读请求，所述分发子模块还用于：

[0207] 将所述读请求分发至所述I/O模拟设备中，通过所述I/O模拟设备将所述读请求对应的数据写入所述I/O通道的缓冲区中。

[0208] 在本申请实施例的一种优选实施例中，所述装置还包括：

[0209] 数据读出模块，用于在所述I/O请求为读请求时，将所述读请求对应的数据从缓冲区中读出，读入到vCPU结构体中相应的成员变量中，对所述数据进行对应的模拟处理。

[0210] 对于装置实施例而言，由于其与方法实施例基本相似，所以描述的比较简单，相关之处参见方法实施例的部分说明即可。

[0211] 本说明书中的各个实施例均采用递进的方式描述，每个实施例重点说明的都是与其他实施例的不同之处，各个实施例之间相同相似的部分互相参见即可。

[0212] 本领域内的技术人员应明白，本申请实施例的实施例可提供为方法、装置、或计算机程序产品。因此，本申请实施例可采用完全硬件实施例、完全软件实施例、或结合软件和硬件方面的实施例的形式。而且，本申请实施例可采用在一个或多个其中包含有计算机可用程序代码的计算机可用存储介质(包括但不限于磁盘存储器、CD-ROM、光学存储器等)上实施的计算机程序产品的形式。

[0213] 在一个典型的配置中，所述计算机设备包括一个或多个处理器(CPU)、输入/输出接口、网络接口和内存。内存可能包括计算机可读介质中的非永久性存储器，随机存取存储

器(RAM)和/或非易失性内存等形式,如只读存储器(ROM)或闪存(flash RAM)。内存是计算机可读介质的示例。计算机可读介质包括永久性和非永久性、可移动和非可移动媒体可以由任何方法或技术来实现信息存储。信息可以是计算机可读指令、数据结构、程序的模块或其他数据。计算机的存储介质的例子包括,但不限于相变内存(PRAM)、静态随机存取存储器(SRAM)、动态随机存取存储器(DRAM)、其他类型的随机存取存储器(RAM)、只读存储器(ROM)、电可擦除可编程只读存储器(EEPROM)、快闪记忆体或其他内存技术、只读光盘只读存储器(CD-ROM)、数字多功能光盘(DVD)或其他光学存储、磁盒式磁带,磁带磁磁盘存储或其他磁性存储设备或任何其他非传输介质,可用于存储可以被计算设备访问的信息。按照本文中的界定,计算机可读介质不包括非持续性的电脑可读媒体(transitory media),如调制的数据信号和载波。

[0214] 本申请实施例是参照根据本申请实施例的方法、终端设备(系统)、和计算机程序产品的流程图和/或方框图来描述的。应理解可由计算机程序指令实现流程图和/或方框图中的每一流程和/或方框、以及流程图和/或方框图中的流程和/或方框的结合。可提供这些计算机程序指令到通用计算机、专用计算机、嵌入式处理机或其他可编程数据处理终端设备的处理器以产生一个机器,使得通过计算机或其他可编程数据处理终端设备的处理器执行的指令产生用于实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能的装置。

[0215] 这些计算机程序指令也可存储在能引导计算机或其他可编程数据处理终端设备以特定方式工作的计算机可读存储器中,使得存储在该计算机可读存储器中的指令产生包括指令装置的制造品,该指令装置实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能。

[0216] 这些计算机程序指令也可装载到计算机或其他可编程数据处理终端设备上,使得在计算机或其他可编程终端设备上执行一系列操作步骤以产生计算机实现的处理,从而在计算机或其他可编程终端设备上执行的指令提供用于实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能的步骤。

[0217] 尽管已描述了本申请实施例的优选实施例,但本领域内的技术人员一旦得知了基本创造性概念,则可对这些实施例做出另外的变更和修改。所以,所附权利要求意欲解释为包括优选实施例以及落入本申请实施例范围的所有变更和修改。

[0218] 最后,还需要说明的是,在本文中,诸如第一和第二等之类的关系术语仅仅用来将一个实体或者操作与另一个实体或操作区分开来,而不一定要求或者暗示这些实体或操作之间存在任何这种实际的关系或者顺序。而且,术语“包括”、“包含”或者其任何其他变体意在涵盖非排他性的包含,从而使得包括一系列要素的过程、方法、物品或者终端设备不仅包括那些要素,而且还包括没有明确列出的其他要素,或者是还包括为这种过程、方法、物品或者终端设备所固有的要素。在没有更多限制的情况下,由语句“包括一个……”限定的要素,并不排除在包括所述要素的过程、方法、物品或者终端设备中还存在另外的相同要素。

[0219] 以上对本申请所提供的一种KVM虚拟化下处理I/O请求的方法和一种KVM虚拟化下处理I/O请求的装置,进行了详细介绍,本文中应用了具体个例对本申请的原理及实施方式进行了阐述,以上实施例的说明只是用于帮助理解本申请的方法及其核心思想;同时,对于本领域的一般技术人员,依据本申请的思想,在具体实施方式及应用范围上均会有改变之

处,综上所述,本说明书内容不应理解为对本申请的限制。

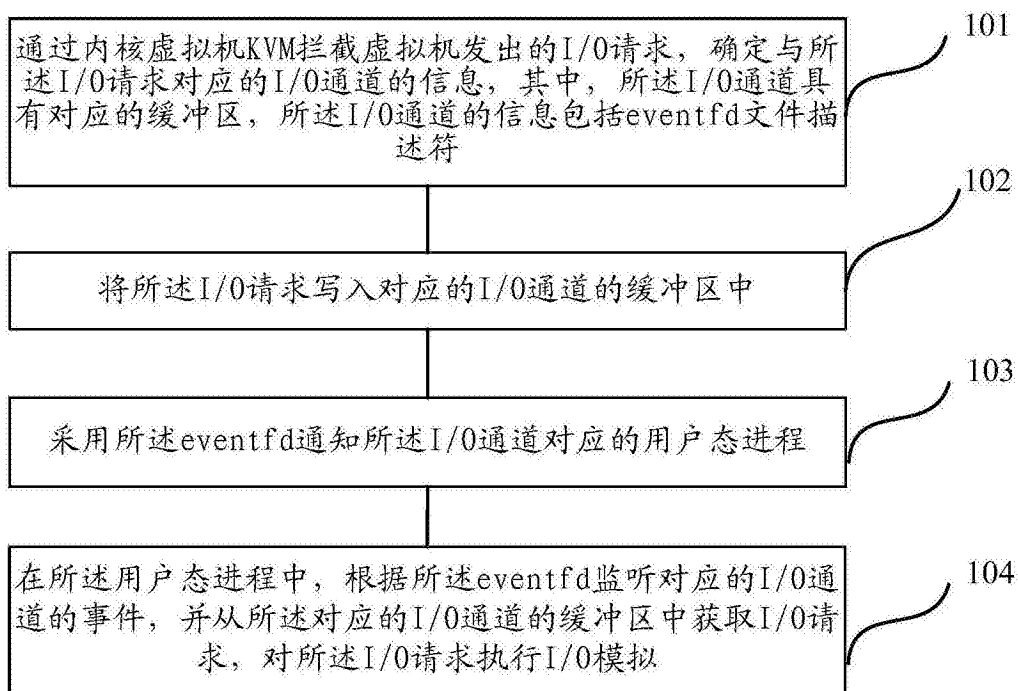


图1

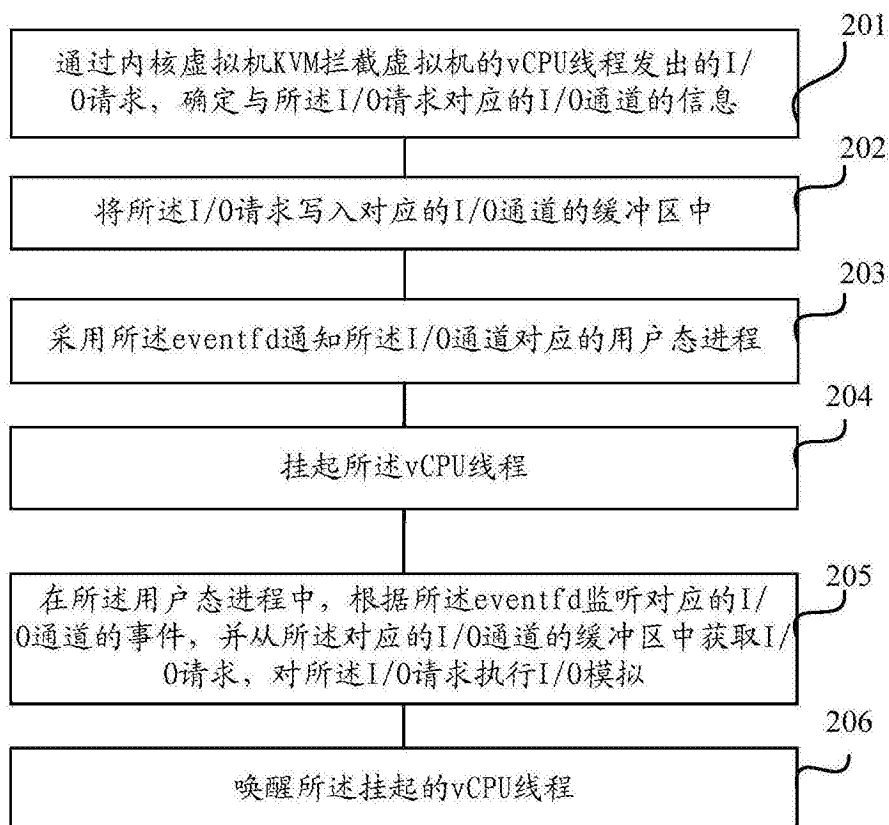


图2

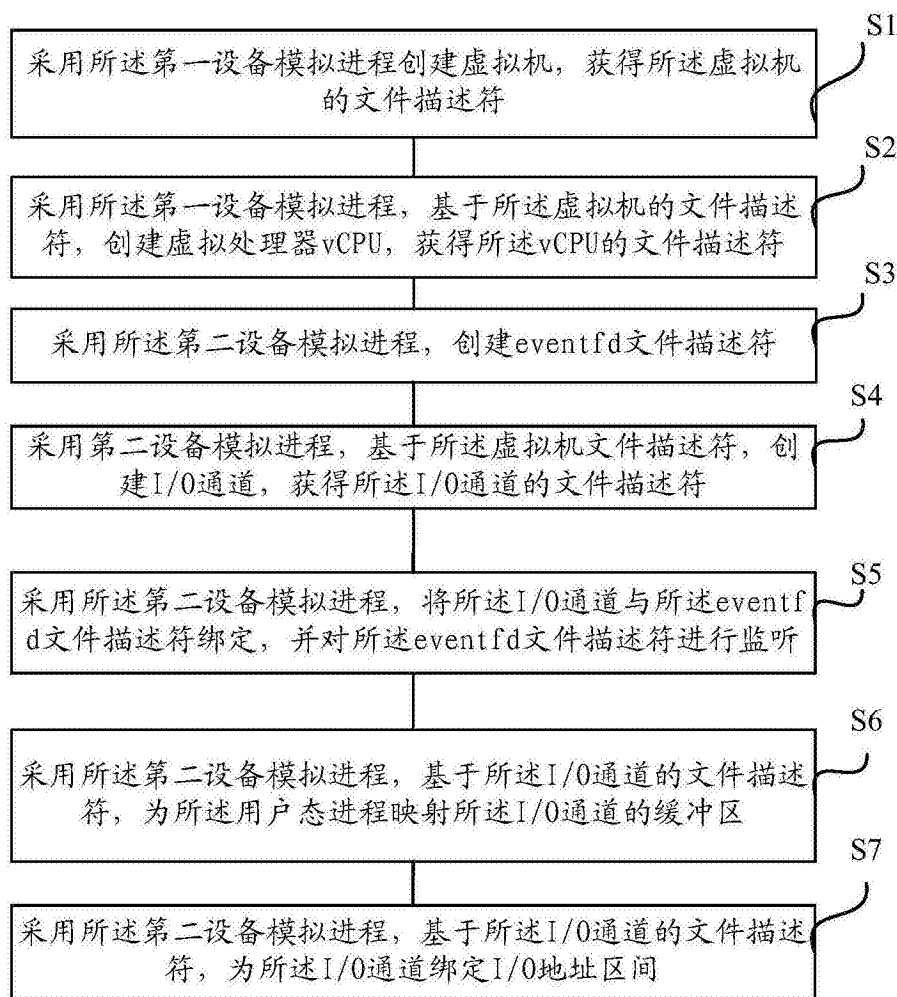


图3

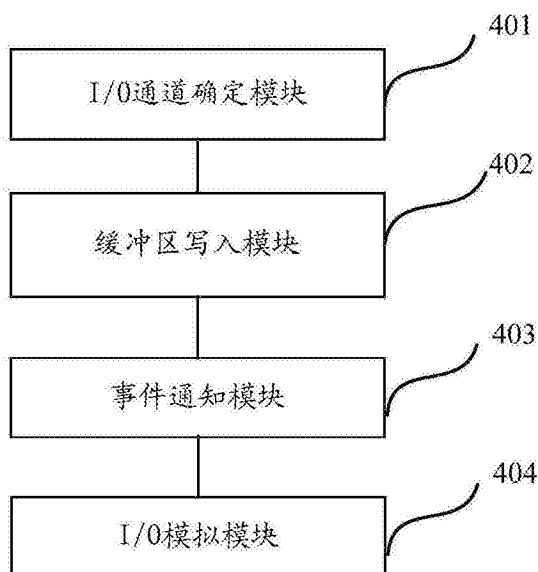


图4