移动平台Android操作系统虚拟化技术的实现

刘博文[1,2,3],顾乃杰[1,2,3],谷德贺[1,2,3],苏俊杰[1,2,3]
LIU Bowen[1,2,3], GU Naijie[1,2,3], GU Dehe[1,2,3], SU Junjie[1,2,3]

1.中国科学技术大学 计算机科学与技术学院,合肥 230027
2.中国科学技术大学 安徽省计算与通信软件重点实验室,合肥 230027
3.中国科学技术大学 先进技术研究院,合肥 230027
1.School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China
2.Anhui Province Key Laboratory of Computing and Communication Software, University of Science and Technology of China, Hefei 230027, China
3.Institute of Advanced Technology, University of Science and Technology of China, Hefei 230027, China

LIU Bowen, GU Naijie, GU Dehe, et al. Implementation of OS-level virtualization technology for Android on mobile platform. Computer Engineering and Applications, 2017, 53(14):32-38.

Abstract: The virtualization technology research is gradually moving from the server area to the field of mobile intelligent devices. The existing virtualization architecture requires a large number of instruction translations between the physical hardware layer and the virtual machine, which is of high cost and low efficiency. In order to solve this problem, it proposes a lightweight mobile operating system level virtualization architecture. It is based on the Linux kernel namespace mechanism to expand the driver namespace framework to achieve multiple virtual Android systems running simultaneously. In addition, the universal active-inactive model is designed to ensure the isolation and multiplexing of the hardware devices among virtual systems in order to solve the conflict produced when multiple virtual Android systems have access to a set of hardware devices simultaneously. Experimental results show that the virtual Android systems do not increase the overhead in the CPU utilization while the memory usage decreased by 6.7%, which proves that this virtualization architecture has a high versatility and practicality.

Key words: Operating System(OS)-level virtualization; Android system; namespace mechanism; hardware isolation and multiplexing

摘 要:虚拟化技术的研究正逐渐从服务器端转向移动智能设备领域。现有的虚拟化架构需要在物理硬件层和虚拟系统间进行大量的指令翻译,开销大,效率低。针对这一问题,提出了一种轻量级的移动操作系统虚拟化架构。通过在Linux内核命名空间机制的基础上扩展Driver命名空间框架,实现了多个虚拟Android系统的同时运行。此外,针对多个虚拟系统同时访问一套硬件设备发生冲突的问题,设计了通用的active-inactive模型来保证虚拟系统间对硬件设备的隔离复用。实验结果表明,虚拟后的Android系统在CPU使用率上并没有增加额外的开销,在内存使用量上减少了6.7%,此虚拟化架构具有很好的通用性与实用性。

关键词:操作系统虚拟化;Android系统;命名空间机制;硬件隔离复用
文献标志码:A **中图分类号:**TP316.89 doi:10.3778/j.issn.1002-8331.1703-0314

1 引言

随着移动设备的硬件配置不断提高、用户使用场景的多样性与日俱增、安全与隐私问题日益凸显,移动平台上对于虚拟化技术的需求也愈演愈烈[1]。虚拟化技术

基金项目:安徽省自然科学基金(No.1408085MKL06);高等学校学科创新引智计划项目(No.B07033)。
作者简介:刘博文(1991—),男,硕士研究生,研究领域为操作系统虚拟化技术,E-mail:bwliu@mail.ustc.edu.cn;顾乃杰(1961—),男,博士,教授,研究领域为并行分布式算法,并行体系结构;谷德贺(1992—),男,硕士研究生,研究领域为操作系统虚拟化;苏俊杰(1991—),男,博士研究生,研究领域为虚拟化技术,操作系统。
收稿日期:2017-03-20 **修回日期:**2017-05-09 **文章编号:**1002-8331(2017)14-0032-07

通过将硬件资源进行隔离并加以管理，可以显著提高资源的利用率，已广泛应用在PC端和高性能服务器领域，而目前针对移动平台的研究成果却十分匮乏[2]。传统的Hypervisor虚拟化架构[3]虽然具有良好的数据隔离性，但需要在物理硬件层和虚拟机间进行繁复的指令翻译[4]，为系统带来大量的性能损耗。移动终端中对各类硬件设备进行了高度封装与集成，对性能要求严苛，因此需要设计一种轻量级且通用的虚拟化架构来满足移动平台对于虚拟化的需求。

针对当前移动平台中虚拟化架构效率不高的问题，浙江大学陈文智团队通过将LXC工具移植到移动ARM平台，使用LXC工具创建出多个容器，并将Android系统运行于每个容器中，实现了一种基于容器的虚拟化解决方案[5-6]。但是该方案需要修改每个虚拟Android系统的源代码，不具有良好的通用性和可移植性。哥伦比亚大学的Andrus和Nieh等人摒弃了传统的Hypervisor虚拟化设计架构，通过初始化一个被称为CellD的Android操作系统实例，用来创建出多个独立的虚拟Android系统[7-8]。CellD的功能类似于宿主系统，用来管理各个虚拟系统实例。此方案虽然实现了Android操作系统级的虚拟化，但是构造出的CellD实例却引入了一部分额外开销。

为了解决上述方案存在的问题，达到轻量级、高通用性的目的，本文将Android系统开源性、应用软件丰富性、硬件设备多样性等优点与虚拟化技术相结合，基于移动ARM平台，提出了一种轻量级的移动操作系统虚拟化架构。通过在Linux内核现有的命名空间资源隔离机制中扩展Driver命名空间框架，用来支持多个虚拟Android系统实例同时运行在一套硬件设备上。并在此基础上提出active-inactive模型，实现了多个虚拟系统隔离复用硬件设备。此架构是一种将虚拟化技术应用在移动平台上的操作系统虚拟化解决方案，具有良好的通用性与实用性。

2　技术背景

2.1　操作系统级虚拟化

虚拟化技术从实现原理的抽象层次上主要分为原始级虚拟化、硬件级虚拟化和操作系统级虚拟化[9]。前两种方法在实现原理上均采用传统的Hypervisor架构，在宿主系统与虚拟系统之间引入VMM（Virtual Machine Monitor）层来统一管理虚拟系统并完成指令翻译工作。而操作系统级别的虚拟化技术以共享内核的方式，在宿主系统上为应用程序提供多个完整、隔离的运行环境，每个独立的运行环境和宿主系统具有相同的功能与特性，相当于内核的一个新特性。操作系统级虚拟化避免了硬件层和虚拟实例间大量的指令翻译工作，是一种轻量级的虚拟化技术[10]。

容器（LXC）技术是操作系统级虚拟化的典型代表，通过Linux内核提供的Namespace与Cgroup特性[11]，不需要虚拟任何硬件设备即可实现在唯一系统实例上以共享内核、共享文件系统的方式同时运行多个独立的虚拟操作系统。图1显示了一种基于容器的操作系统虚拟化架构，该架构在宿主Android系统完整启动的基础上，利用LXC工具创建出多个容器并将完整的Android系统运行于每个容器中，从而构造出多个虚拟实例。宿主系统与多个容器中的虚拟实例共享同一个Linux内核，相比于传统的Hypervisor架构，性能得到了大幅度提高。

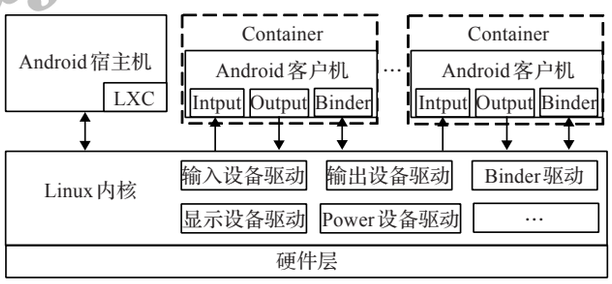

图1　基于容器的操作系统级虚拟化架构

2.2　命名空间机制

Linux从内核版本2.4.19开始，提供了一种轻量级的资源隔离方案，被称为命名空间机制[12]。通过对全局的系统资源进行了封装隔离，使得UTS主机名、IPC进程通信机制、PID进程树以及MNT文件系统挂载点等资源不再具有全局性，而是属于某个特定的命名空间。目前内核中实现了六种不同类型的命名空间（如表1），每一种命名空间是一类全局系统资源的抽象集合，这种抽象使得在进程的各类命名空间中可以看到隔离的全局系统资源，且对于其他命名空间中的资源透明。只需要在clone()系统调用时指定相应的flag即可创建新的命名空间[13]。

表1　内核现有命名空间简介

类型	参数flag	隔离内容
IPC	CLONE_NEWIPC	隔离IPC进程通信、消息队列机制
Network	CLONE_NEWNET	隔离网络设备、端口、防火墙等信息
MNT	CLONE_NEWNS	隔离文件系统挂载点信息
PID	CLONE_NEWPID	隔离进程ID号
USER	CLONE_NEWUSER	隔离User ID以及Group ID
UTS	CLONE_NEWUTS	隔离Hostname以及NIS

与操作系统级虚拟化密切相关的是PID命名空间与MNT命名空间。PID命名空间为进程提供了一个隔离的进程视图，同一个进程可同时处于不同的PID命名空间中，拥有不同的进程ID。所有子PID命名空间相互透明，具有嵌套式的父子关系，通过init进程所在的根PID命名空间进行管理。MNT命名空间则隔离了文件系统挂载点信息，不同的MNT命名空间拥有独立的挂

载点信息,内部进程组看到的文件系统是不一样的,相互透明。

3　架构设计

Android系统采用Linux内核,并在此基础上增加了丰富的应用软件与多样的硬件设备,因此对各种硬件设备的虚拟化实现是本架构设计的核心。此架构的设计主要包括3个方面:(1)总体架构设计;(2)Driver命名空间框架设计,该框架用来隔离各虚拟系统对设备的访问;(3)active-inactive模型设计,该模型保证了各虚拟系统对硬件设备的隔离复用。

3.1　总体架构设计

此方案的总体架构如图2所示。

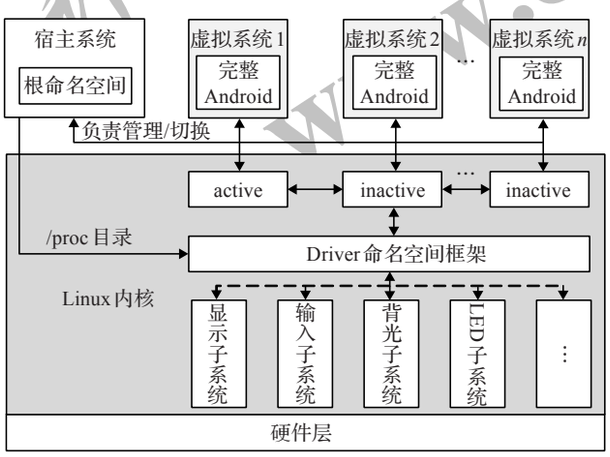

图2　总体框架图

整个系统启动时会先启动一个最基本的初始化环境,负责管理和切换多个虚拟客户系统。该环境是一个精简的Android系统,由init进程构建了Android的基本服务、文件系统以及各类命名空间。

宿主系统的根命名空间成功启动后,为宿主系统的init进程创建各类子命名空间,MNT命名空间为虚拟客户系统提供隔离的文件系统视图;PID命名空间提供独立的进程树视图;Driver命名空间则提供隔离的硬件访问视图。通过各类子命名空间提供隔离的全局系统资源集合为每个虚拟客户系统创建出一个完整而独立的Android系统运行环境。

通过在内核中扩展Driver命名空间框架,使得多个虚拟客户系统可同时运行在一套硬件平台上,并且隔离地访问硬件设备。此外,active-inactive模型的设计将多个虚拟客户系统标记为一个active状态和若干个inactive状态。由于硬件设备可同时被多个虚拟客户系统中的进程访问,该模型确保了只有active状态的虚拟客户系统中的进程对硬件设备的访问是有效的,实现了对硬件设备的隔离复用机制。本架构是一个针对Android系统中各类硬件的通用框架设计,可以实现包括显示设备、输入设备、Backlight背光设备以及LEDs指示灯在内的Android最核心的硬件设备的隔离复用。内核层面上利用PID和MNT命名空间机制为每个虚拟客户系统构造出隔离的运行环境;硬件设备层面利用Driver命名空间框架实现了对硬件资源的隔离复用。每个虚拟客户系统都有私有的空间,相互独立隔离,且不可以篡改、访问其他虚拟客户系统。宿主系统通过对/proc目录的读写来管理与切换每个虚拟客户系统。

3.2　Driver命名空间框架设计

作为对Linux内核现有命名空间机制的扩展,Driver命名空间为每个虚拟客户系统提供了隔离的硬件访问环境。该框架由3部分组成:内核数据结构设计、通用接口设计、硬件设备回调函数设计。

在内核中设计driver_namespace结构体(图3)用来表示一个Driver命名空间。成员active标识是否为激活状态,用于active-inactive模型识别状态;Driver命名空间与PID命名空间具有一对一的关系,通过获取init进程所属的PID命名空间并共享给所属的Driver命名空间;成员driver_info代表在Driver命名空间中注册的某个硬件设备;notifiers则把driver_info结构里的成员notifier串连起来,用于状态切换时通知注册在该Driver命名空间中的每个硬件设备响应回调处理函数。

```
struct driver_namespace {
        bool active;
        struct pid_namespace *pid_ns;
        struct driver_info *driver_info[DRIVER_MAX];
        struct blocking_notifier_head notifiers;
};
```

图3　Driver命名空间结构体设计

设计全局结构体数组driver_global将硬件设备表示在Driver命名空间中,每个元素表示一个注册在Driver命名空间框架中的硬件设备。成员name表示设备名称;ops结构提供create()接口,具体的实现则在各个硬件设备中定义,当内核在初始化硬件设备时被调用;成员head节点将不同Driver命名空间中的同一种硬件设备串联起来。当Driver命名空间中的进程首次使用某硬件设备A时会创建一个A_driver_ns结构。该结构中有指向driver_info结构的指针,因此它是Driver命名空间框架与具体硬件设备连接的纽带。该结构创建成功后,每次访问硬件设备A便会创建一个A_client对象。所有A设备的A_client对象连接在一起且同一个Driver命名空间下的A_client被串在该Driver命名空间中的A_driver_ns结构中。综上所述,Driver命名空间框架在内核中的数据结构设计如图4所示。

在表示进程信息的task_struct结构中存在nsproxy成员指向该进程所在的各类命名空间[14]。通过在nsproxy结构中添加指向driver_namespace结构的指针将Driver命名空间框架引入内核。

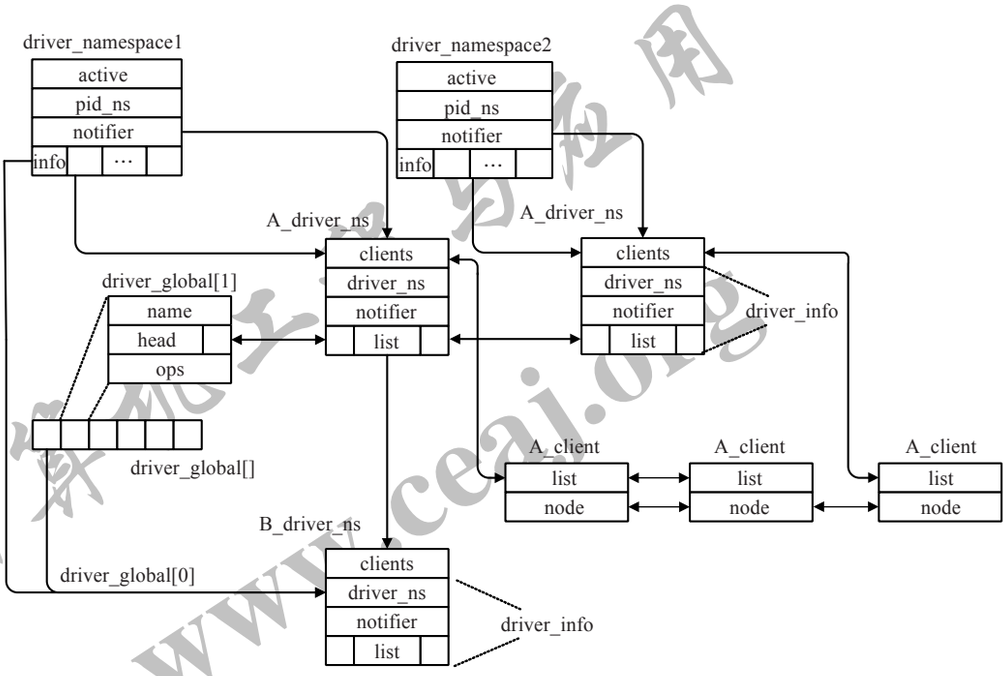

图4　内核数据结构设计图

数据结构的设计将硬件设备与Driver命名空间关联在一起,在硬件设备初始化并在driver_global结构中注册成功后,当某进程实际访问硬件设备A时需要在Driver命名空间框架中定义通用的接口函数来创建driver_info结构,这极大简化了Android系统中各类硬件设备与该框架的关联;硬件设备则需要结合自身特点定义回调函数用于创建A_driver_ns结构。Driver命名空间框架的工作原理如图5所示。

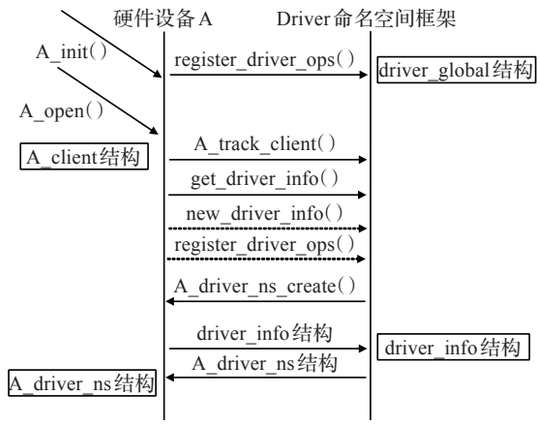

图5　Driver命名空间框架工作原理

图5中右向箭头表示定义在Driver命名空间框架中的通用接口函数,当硬件设备A被访问时会通过上述调用关系触发定义在各个硬件设备中的回调函数来创建A_driver_ns结构体;虚线则表示当进程首次访问硬件设备A时的创建流程;而左向箭头表示定义在各个硬件设备子系统中的回调函数。通过图5中的函数调用关系,最终将创建的A_driver_ns保存在A子系统中,driver_info结构保存在Driver命名空间中,完成了硬件设备与Driver命名空间框架的关联。

3.3　active-inactive模型设计

不同Driver命名空间框架中允许个性化定制注册不同的硬件设备,为每个虚拟客户系统提供隔离的硬件访问环境,active-inactive模型则保证了多个虚拟系统对硬件的隔离复用。该模型允许系统中只存在一个active状态的虚拟客户系统和若干个inactive状态的虚拟客户系统,每个虚拟客户系统均具有访问硬件设备的权限,但仅处于active状态中的进程对硬件设备的访问是有效的。由于每个虚拟客户系统由宿主系统构造,且各类根命名空间可以通过/proc目录来管理子命名空间,因此宿主系统通过/proc目录来统一管理和切换多个虚拟客户系统。

当虚拟客户系统的状态发生切换时,该模型首先通过捕获内核proc写事件[15]来触发Driver命名空间框架中的set_state_switch()接口,该接口封装了Linux内核通知链(Notification Mechanism)机制中的事件通知函数。依据定义在内核中的ACTIVE与INACTIVE宏,通知发生状态切换的Driver命名空间中注册的所有硬件设备,触发相应的切换处理回调函数,而每个硬件设备的切换处理回调函数已经在创建A_driver_ns时注册在了Driver命名空间框架中,具体实现则在各个硬件设备中定义。

4　架构实现

4.1　系统启动流程

Android系统的内核默认是没有开启与命名空间相关的编译选项的,通过在.config文件中打开相关配置选项来定制开启命名空间机制的内核。同时需要对内核

的根Makefile文件作出相应的修改,使得Driver命名空间框架会被编译进内核中。

内核启动成功后会在用户空间创建init进程,Android系统的init进程在启动时会根据init.rc文件来创建文件系统、关键服务等[16]。通过为宿主系统的根init进程创建各类命名空间,这些命名空间的组合为子init进程提供了各类系统资源隔离的视图,子init进程完成和根init进程同样的功能,从而构造出多个完整而独立的虚拟Android客户系统。步骤如下:

(1)编写创建虚拟客户系统的脚本。该脚本通过在系统调用clone中指定堆栈空间、指定子进程需要继承的flag标记等参数为根init进程构造出各类命名空间集合。

(2)修改宿主系统的init.rc文件。将步骤(1)的脚本文件添加进init.rc中的启动过程并运行多次,创建出多个虚拟Android客户系统运行环境。首个创建出的虚拟客户系统设置为active状态。

4.2　硬件设备隔离与复用

各类命名空间的集合初步构建了一个独立的Android系统运行环境,但由于移动设备高度集成了大量硬件设备,因此该运行环境还不足以支持一个完整的移动设备使用环境。Driver命名空间框架中设计的数据结构与接口函数屏蔽了硬件差异,是一种通用的隔离框架,可以实现移动平台Android系统大多数硬件设备的隔离复用。由于篇幅所限,本文选取最核心的显示设备子系统和输入设备子系统进行介绍。

(1)显示设备子系统隔离复用

Android驱动层通过提供一个标准封装好的FrameBuffer(下文称为FB)显示驱动程序为物理硬件提供了抽象显示,应用程序通过mmap系统调用将一个打开的FB字符设备文件映射到进程的虚拟地址空间中,允许进程以读写这段虚拟地址的方式直接操作物理显存地址[17],从而实现了对ARM平台的LCD显示屏操作。

为每个虚拟客户系统创建一个虚拟FB结构,active虚拟客户系统中的进程允许将虚拟FB结构直接映射到真实的FB结构中;而inactive虚拟客户系统中的进程仅能映射虚拟FB结构,不做实际的显示操作。当active与inactive状态的虚拟客户系统发生切换时,FB显示设备子系统根据切换处理回调函数做出响应,该响应过程由3个步骤完成:FB显存缓冲区地址的重映射、FB显存缓冲区内容的深拷贝以及硬件状态同步。

(2)输入设备子系统的隔离复用

Android系统输入设备子系统主要由硬件驱动层、输入核心层以及输入事件处理程序三部分组成。输入事件在默认的情况下会被发送到所有监听该输入事件的进程中,但这个机制并不支持为多个虚拟客户系统提供所需的隔离[18]。为了实现输入设备的隔离复用,只需要修改内核中默认的输入事件处理程序evdev即可。对于每个进程监听的输入事件,输入事件处理程序首先检查是否在active状态的虚拟客户系统中。如果是,则响应该输入事件;如果不是,则不唤醒特定的处理流程。因此不需要修改驱动层以及输入核心层。

输入设备的隔离复用原理如图6所示。首先,当输入设备子系统在初始化的时候,通过调用Driver命名空间框架中的注册接口函数将输入设备子系统注册在全局数组driver_global中。然后每当进程访问输入设备中某个设备时,evdev_clients被创建。通过Driver命名空间框架中的相关接口创建一个关联该框架与输入设备子系统的evdev_driver_ns结构,同时将代表输入设备的driver_info结构保存在Driver命名空间框架中。最后,当发生状态切换时触发set_state_switch接口来回调定义在输入设备中的切换处理回调函数,从而实现多个虚拟客户系统对输入设备子系统的隔离复用。

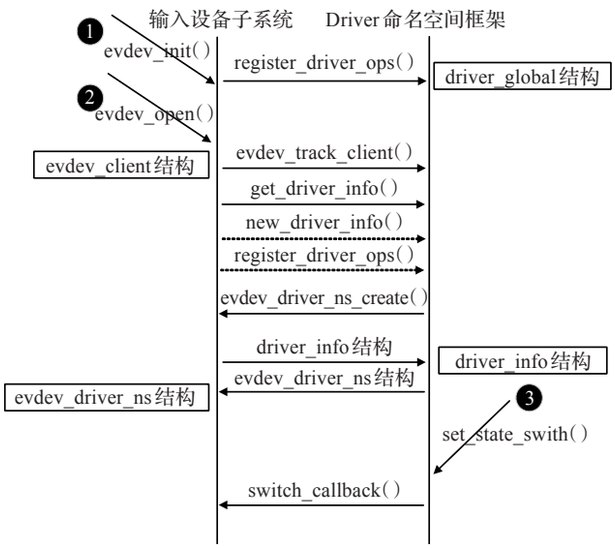

图6　输入设备隔离复用原理

4.3　架构内存优化

Android系统中允许杀死一些对系统没有影响的进程,不会损失任何功能;而核心进程即使被杀死也会在后台重新启动。由于宿主系统与多个虚拟客户系统共用内存,通过修改Android系统中的LMK(Low Memory Killer)机制来保证同时启动多个虚拟客户系统时内存的正常使用。

每个进程的/proc/PID/oom_adj文件中保存了一个临界值[19],取值范围从-17到15。当系统内存不足时,临界值越大的进程越容易被杀死,而内存的临界值则通过在系统的lowmem_minfree数组中指定。利用LMK机制的实现原理,通过在每个虚拟客户系统中为核心服务与非核心服务设定不同的临界值,达到杀死inactve状态的虚拟客户系统中大量消耗内存的进程,实现对内存的简单优化。

5 实验与测试

5.1 测试环境与方法

根据上文设计的操作系统级虚拟化架构,基于移动ARM平台,选取通用的Cortex A7开发板,在不失一般性的基础上结合该平台的参数配置,利用宿主系统创建出2个虚拟Android客户系统,并从功能测试与性能测试两方面来进行评估分析,测试平台参数如表2所示。

表2 测试平台参数

内容	类别	参数
硬件	CPU	ARM Cortex A7 dual-core
	内存	DRAM 1 GB
	存储	eMMC 8 GB
操作系统	内核版本	Linux kernel 3.4
	Android	AOSP Android 4.2.2

5.2 功能测试

在功能测试上从宿主系统启动测试、虚拟客户系统启动测试、切换测试以及隔离性测试四个方面进行了完整的测试工作。

通过查询文件系统、核心服务是否启动成功并根据readlink工具查看各个根命名空间是否创建成功来测试宿主系统是否启动成功;利用pstree工具查询两个虚拟Android客户系统的进程树,可判断虚拟客户系统是否启动成功;系统切换测试则在每个虚拟客户系统中安装实现切换的应用程序,通过多次切换操作来完成;隔离性测试则通过在两个虚拟客户系统中分别进行读写操作、安装应用程序来验证隔离性。

5.3 性能测试

性能测试上从CPU使用率以及内存使用量两个角度来衡量虚拟化的引入所带来的性能开销。

(1)CPU使用率

分别在原生Android和虚拟化的系统中安装“videos”应用软件,从启动初期到完全启动过程中每隔1 s统计CPU使用率,结果如图7所示。

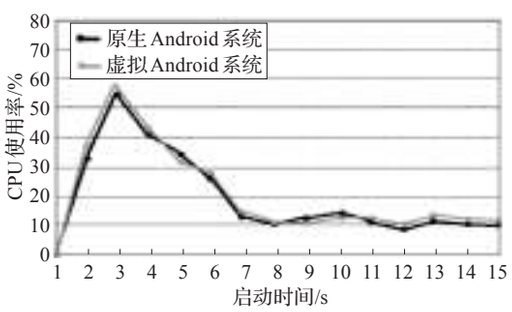

图7 videos启动过程CPU使用率测试图

从统计图中可以看出,对于videos应用程序从启动初期到完全启动状态,虚拟化后的Android系统和原生Android系统并无明显差别,平均有0.8%的性能提升。按照同样的方法,分别对camera、photos、clock、calculator等系统软件的CPU使用率进行测试,测试结果如表3所示。

表3 不同应用软件CPU优化比

应用软件	完全启动耗时/s	CPU使用率优化比/%
Videos	15	0.8
Camera	18	0.5
Photos	13	0.7
Clock	8	0.9
Calculator	9	0.8

表3的测试结果表明,此虚拟化架构对于Android系统软件启动过程的CPU使用率有近1%的优化效果,说明虚拟化的引入并没有带来额外的CPU开销。

(2)内存使用量

通过统计只启动宿主系统、启动宿主系统和一个虚拟客户系统、启动宿主系统和两个虚拟客户系统三种情况下的内存使用量,并从系统启动初期、启动15 min、启动30 min、启动45 min分别对上述三种情况进行了分析,如图8所示。

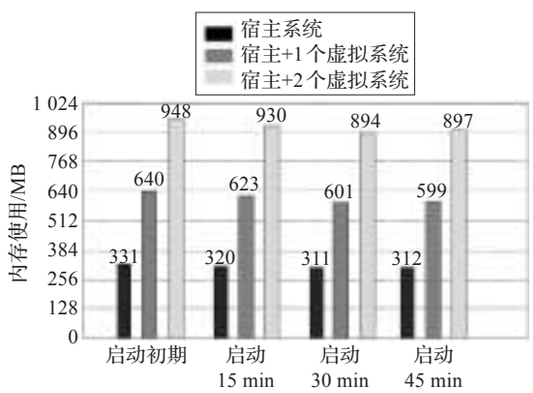

图8 内存使用量测试结果

统计结果表明,随着启动时间的持续,由于LMK内存优化的原因,虚拟客户系统所带来的内存开销逐渐减少。从启动30 min后的稳定状态得出,由于共享内核的缘故,每个虚拟客户系统所带来的内存开销比原生系统约减少了6.7%。

6 结束语

本文针对移动平台中现有虚拟化技术解决方案开销大、效率低的问题设计了一种轻量级的操作系统虚拟化架构,阐述了该架构的原理与实现,并通过实验测试验证了此架构功能的正确性,同时从CPU使用率和内存使用量两个角度分析了该虚拟化架构的引入不仅没有带来额外的性能开销,反而有一定的性能优化。此架构通过在内核中引入通用的Driver命名空间框架并设计active-inactive模型,在保证多个虚拟客户系统正常启动的前提下实现了对硬件资源的隔离复用,具有良好的通用性与实用性。下一步工作是期望从内存使用量、IO

读写等性能角度优化此架构,并在硬件配置更优的平台上进行测试与分析。

参考文献:

[1] Xu Lei,Li Guoxi,Li Chuan,et al.Condroid:a container-based virtualization solution adapted for Android devices[C]//3rd IEEE International Conference on Mobile Cloud Computing,Services,and Engineering (MobileCloud 2015),San Francisco,CA,2015:81-88.

[2] Paul K,Kundu T K.Android on mobile devices:an energy perspective[C]//2010 IEEE 10th International Conference on Computer and Information Technology (CIT),2010:2421-2426.

[3] 文雨,孟丹,詹剑锋.面向应用服务级目标的虚拟化资源管理[J].软件学报,2013(2):358-377.

[4] Butler M.Android:changing the mobile landscape[J].IEEE Pervasive Computing,2011(1):4-7.

[5] 吴佳杰.基于LXC的Android系统虚拟化的关键技术设计与实现[D].杭州:浙江大学,2014.

[6] Chen Wenzhi,Xu Lei,Li Guoxi,et al.A lightweight virtualization solution for Android devices[J].IEEE Transactions on Computers,2015,64(10):2741-2751.

[7] Andrus J,Dall C,Hof A V,et al.Cells:a virtual mobile smartphone architecture[C]//Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP 2011),Cascais,Portugal,2011:173-187.

[8] Detken K O,Oberle A,Kuntze N,et al.Simulation environment for mobile virtualized security appliances[C]//2012 IEEE 1st International Symposium on Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS 2012),Offenburg,Germany,2012:113-118.

[9] IBM虚拟化与云计算小组.虚拟化与云计算[M].北京:电子工业出版社,2010:26-54.

[10] Dall C,Nieh J.KVM/ARM:the design and implementation of the Linux ARM hypervisor[J].ACM SIGPLAN Notices,2014,49(4):333-347.

[11] 孙鹏.基于虚拟化技术的智能手机移动办公室的研究与应用[D].上海:复旦大学,2010.

[12] Tihfon G M,Kim J,Kim K J.A new virtualized environment for application deployment based on docker and AWS[C]//International Conference on Information Science and Applications (ICISA),2016:1339-1349.

[13] Kerrisk M.clone()-Linux programmer's manual[EB/OL].(2016-12-12).http://man7.org/linux/man-pages/man2/clone.2.html.

[14] 陈晓.基于LinuxContainer的移动终端虚拟化[D].广州:华南理工大学,2013.

[15] Socala A,Cohen M.Automatic profile generation for live Linux memory analysis[J].Digital Investigation,2016(16):11-24.

[16] Shah M A,Kamran M,Khan H,et al.Vdroid:a lightweight virtualization architecture for smartphones[C]//2016 Future Technologies Conference (FTC),San Francisco,USA,2016.

[17] Kim J,Kwon O C,Lee C G.Development of frame buffer structure for automatic dynamic resolution switching on Android mobile platform[J].Journal of KISS:Computing Practices,2010,16(12):1209-1213.

[18] Fang Zhejun,Liu Qixu,Zhang Yuqing,et al.A static technique for detecting input validation vulnerabilities in Android[J].Science China Information Sciences,2017,60(5).

[19] Singh A,Agrawal A V,Kanukotla A.A method to improve application launch performance in Android devices[C]//2016 International Conference on Internet of Things and Applications (IOTA),2016:112-115.

(上接31页)

[14] 黄仁,田丰,田维兴.基于加速度传感器的运动模式识别[J].计算机工程与应用,2015,51(6):235-239.

[15] 肖子明,吴雨川,马双宝.基于三维重力加速器人体动作识别与分类研究[J].武汉大学学报:工学版,2014,47(3):1-4.

[16] Ayachi F S,Nguyen H P,de Brugiere E G,et al.The use of empirical mode decomposition-based algorithm and inertial measurement units to auto-detect daily living activities of healthy adults[J].IEEE Trans on Neural Syst Rehabil Eng,2016,24(10):1060-1070.

[17] 陈鹏展,罗漫,李杰.基于加速度传感器的连续动态手势识别[J].传感器与微系统,2016,34(1):39-42.

[18] 云杨,卢美静,穆天红.基于AdaBoost_SVM的葡萄酒品质分类模型优化设计[J].陕西科技大学学报,2017,35(1):178-182.

[19] Yu Y,Li Y F,Zhou Z H.Diversity regularized machine[C]//International Joint Conference on Artificial Intelligence,2011.

[20] Wu Yuchuan,Zhu Li,Hu Feng.Use of wearable sensors for studying kinematics and dynamics of aged people's gait[C]//International Conference on Mechanics and Mechanical Engineering.Chengdu:World Scientific Publishing Company,2015.

[21] 周志华.机器学习[M].北京:清华大学出版社,2016.