计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(12):58−63 [doi: 10.15888/j.cnki.csa.006110]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 LXC 的 Android 系统虚拟化技术①

谷德贺[1,2,3], 顾乃杰[1,2,3], 刘博文[1,2,3], 苏俊杰[1,2,3], 贺爱香[1,4]

[1](中国科学技术大学 计算机科学技术学院, 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)
[4](安徽新华学院 信息工程学院, 合肥 230088)

摘　要: 虚拟化技术的研究正逐渐从高性能服务器端转向移动智能设备领域. 现有的虚拟化方案多是采用多内核方案, 系统负载高, 效率低. 针对车载系统等平台多屏显示以及资源受限等问题, 本文提出一种基于容器技术的 Android 轻量级虚拟化方案. 该方案通过利用 Namespace 资源隔离机制和 Cgroup 资源分配机制, 使得 ARM 平台在资源使用较少的同时, 能够同时启动多个 Android 虚拟机, 并且各虚拟机上的屏幕显示相互独立. 测试结果表明, 该方案的内存占用率较双系统方案降低了 7%, 而平均 CPU 使用率较原生 Android 系统仅增加了 1%.

关键词: 虚拟化技术; 资源隔离; 资源分配; ARM 平台; 虚拟机

引用格式: 谷德贺,顾乃杰,刘博文,苏俊杰,贺爱香.基于 LXC 的 Android 系统虚拟化技术.计算机系统应用,2017,26(12):58–63. http://www.c-s-a.org.cn/1003-3254/6110.html

Virtualization Technology of Android System Based on LXC

GU De-He[1,2,3], GU Nai-Jie[1,2,3], LIU Bo-Wen[1,2,3], SU Jun-Jie[1,2,3], HE Ai-Xiang[1,4]

[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)
[4](Institute of Information Engineering, Anhui XinHua University, Hefei 230088, China)

Abstract: The virtualization technology research is gradually moving from high-performance server to mobile intelligent devices. Existing virtualization solutions are mostly multi-core solution with high system overhead and low efficiency. This paper presents a lightweight virtualization scheme based on Linux container for multi-screen display and resource limitation of vehicle system. Our program, through the use of Namespace resource isolation mechanism and Cgroup resource control mechanism, can start several Android virtual machines on the ARM platform at the same time, while displaying on different screens and running independently. The performance test results show that on the ARM platform, the program uses less than 7% of the memory of the two systems, and the average CPU usage after virtualization is only 1% higher than the native Android system.

Key words: virtualization technology; resource isolation; resource control; ARM platform; virtual machine

近年来云计算[1]的快速发展, 虚拟化技术[2]被广泛应用于高性能服务器, 以提高系统资源的利用率. 同时, 随着智能手机等移动终端的普及[3], 智能终端扮演着越来越重要的角色, 用户对视频、微信、新闻浏览等功能的使用开始从 PC 转移至移动终端. 由于生活和工作等场景的多样化, 用户不得不携带多个终端设备以满

① 基金项目: 安徽省自然科学基金 (1408085MKL06); 高等学校学科创新引智计划项目 (B07033)
收稿时间: 2017-03-15; 修改时间: 2017-03-31; 采用时间: 2017-04-07

足不同应用场景的需要, 例如车载系统上司机和乘客同时对导航以及娱乐系统的需求. 与此同时, 面对恶意广告、病毒以及隐私信息泄露[4-6]等问题, 更多的用户开始关注移动平台[7]的隐私安全性问题.

通过虚拟机机制[8], 将多个独立且隔离的智能手机软件实例运行在同一个 ARM 硬件上, 可以有效解决 Android 设备的安全风险. 现有的虚拟化方法都对用户层和内核层做了大量地修改, 如哥伦比亚大学提出的 Cells 模型[9]; 或利用 ARM 平台的 Hypervisor 模式[10], 如 KVM/ARM 架构[11]; 或设计完整的微内核模式, 如 OKL4 的微内核模型[12], 将宿主系统和客户机系统隔离运行. 这些方案对于高性能服务器可能有效, 但应用于智能手机主要有两个问题. 其一, 智能手机资源的受限, 运行整个额外的操作系统以及用户空间环境, 将带来很大的系统开销, 导致系统响应速度过慢; 其二, 部分方法对内核层进行大量地修改, 导致宿主系统和客户机系统很难升级和扩展, 对于频繁的 Android 更新, 这将带来很大的工作量.

LXC[13]即 Linux Container, 是利用 Linux 内核容器特性为用户提供空间接口的开源工具, 其通过强大的 API 和简单的工具, 可以让用户轻松创建和管理系统或者应用程序容器. LXC 利用内核支持的资源隔离以及控制机制, 通过对容器的配置, 使用 lxc-start 等工具对容器进行控制, 可以快速部署, 且具有更小的虚拟化开销等优点.

针对移动设备资源的受限, 以及车载系统对于多屏显示的需求, 本文利用 LXC 开源工具提出一种 Android 系统轻量级虚拟化多屏显示方案. 本方案是一种系统级虚拟化方案, 通过利用 Linux 内核支持的虚拟化特性, 在对内核尽可能少的修改前提下, 将多个系统独立的运行在同一个 ARM 平台上且拥有不同的显示屏幕. 本方案中多个系统共用同一个内核, 其所带来的系统负载小, 即使在性能较差的平台也不会影响用户体验, 且具有容易部署和可扩展性强等优势.

1 知识背景与相关工作

现有的移动虚拟化技术主要从以下几个方面来隔离宿主系统, 分别为操作系统级虚拟化方案、基于 Hypervisor 监管程序方案[14]和基于微内核方案[14].

1.1 基于操作系统级方案

操作系统级虚拟化将系统的命名空间划分为不同的虚拟机. 虚拟机之间共享唯一的系统内核以及主机环境, 每个虚拟机只保存本地环境的状态, 本文也是采用此解决方案. Cells[9]是哥伦比亚大学提出的基于操作系统级虚拟化的解决方案. Cells 引入了一种前端虚拟手机 (VP) 和多个后台虚拟手机的使用模型, 使多个 VP 可以互不干扰、独立运行; 且基于 Namespace 思路提出一种新的 Device 命名空间机制, 使不同设备具有不同的访问权限, 例如不可访问、共享访问以及互斥访问, 使手机硬件得到很好的复用和隔离. 但 Cells 对 Linux 内核层做了大量的修改, 使得其不可能并入 Linux 主线中. 不同的是, 本文在未对内核层做大量地修改的情况下, 同样完成了系统需求.

1.2 基于 Hypervisor 方案

KVM/ARM[11]是一个基于 ARM 平台 Hypervisor 模型的完整系统虚拟化解决方案. KVM/ARM 与其他虚拟化方案最大的区别在于大多数 VMM 都实现主要的服务, 例如调度器、内存管理以及时钟等. KVM 利用现有 Linux Kernel 的功能, 以及硬件支持, 使得虚拟技术的系统性能负载很小.

KVM/ARM 引入一种技术, 使其可以在 CPU 不同特权模式下运行. 通过该技术利用 ARM 硬件虚拟化支持进入监管模式, 在统一时间内利用内核模式运行现有的 Linux 内核服务. 该方案中不同系统使用各自的内核, 且上下文切换带来的系统消耗较多, 不适用资源受限的移动设备.

1.3 基于微内核方案

微内核[15]采用最小化原则, 在保证能够实现全部功能的前提下使内核尽量的小. OKL4[12]是从 Open Kernel Labs 出现的一种基于微内核的虚拟化技术. OKL4 通过构建一种简单的内核, 提供比 Hypervisor 监管程序的更高效率, 而且设计的微内核具有一定的通用性, 可以在多平台以及多系统上构建. 通过 vCPUs、vMMU、vIRQs 以及 TLB 相应的硬件支持, 微内核足够小, 并可以正确的支持所有功能. 使得通过 OKL4 的系统负载足够小. 但因为其需要设备支持以及仿真支持, 对于目前日益多元化的移动设备来说, 是一个繁重的硬件设备要求, 也增加了硬件的成本.

2 ARM 平台移动虚拟化的设计方案

针对车载系统等平台多屏显示以及资源受限等问

题, 本文提出一种轻量级虚拟化方案. 该方案主要包括系统的总体架构以及多屏显示方案.

2.1 系统架构

利用 Linux 内核中 Namespace 的资源隔离机制和 Cgroup[16]的资源控制机制, 以及轻量级容器工具 LXC 的特性, 本文采用操作系统级虚拟化解决方案. 通过 LXC 工具对容器系统进行管理以及操作, 利用内核机制的支持, LXC 工具小且功能完善等优势, 完成多系统虚拟化的需求.

命名空间将 LXC 启动的进程隔离放入独立的命名空间中, 通过 Android init 进程初始化 Android 系统, 形成独立的一组进程视图. 同时 Cgroup 机制负责不同系统的资源的配置, 分配 CPU 节点和 Memory 节点, 以及监控容器空间内系统的运行状况. 这样在容器空间内的进程只能感受到当前容器的状况.

Cgroups 可以为不同用户层面的资源管理, 提供一个统一化的接口. 从单个进程的资源控制到操作系统层面, Cgroup 主要提供三个主要功能: 1) 资源限制: Cgroup 可以对进程组使用的资源总额进行限制; 2) 优先级分配: 通过分配 CPU 时间片数量及硬盘 IO 带宽大小, 实际上就相当于控制进程的优先级; 3) 资源统计: Cgroup 可以统计系统的资源使用量, 如 CPU 使用时长、内存使用量等. 本文的架构如图 1 所示.

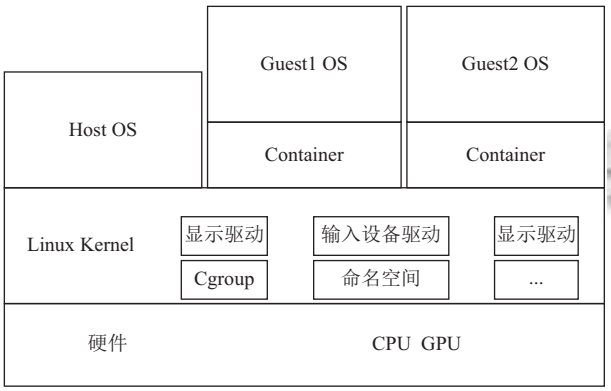

图 1 系统架构图

基于宿主机以及 Android 系统的特性, 底层硬件共享同一个 CPU、Memory、GPU 等设备, 但不同系统拥有不同的显示设备以及输入输出设备, 多个系统间共享同一个 Linux 内核, 需要针对不同的显示以及输入输出设备作设备, 设计隔离以及复用方案, 而 GPU 与 CPU 不同, 其本身具有隔离性, 不同应用程序均持有独立的显示上下文信息, 不需要进行单独的处理. 在宿主机中配置 LXC 工具, 通过 LXC 工具就可以同时启动不同的系统.

本方案采用 LXC 作为容器虚拟化工具, 通过利用内核中命名空间的隔离机制和进程资源控制组对资源进行划分, 使得宿主系统与不同客户机系统能够隔离, 相互不影响, 并且流畅地运行. 该方案使得单个硬件设备即可运行多个虚拟系统, 每个系统运行标准的 Andoird 环境, 每个 Android 系统运行着未修改的 Android 应用程序. 每个虚拟系统完全隔离, 不能检查, 篡改或者访问任何其他的虚拟机.

每个虚拟系统可以被创建和配置通过 PC 机下载到开发板. 虚拟机系统可以被用户删除, 但是其配置权限是通过密码保护的, 只能拥有权限的用户才能在 PC 机上改变. 例如, 对于管理员可以创建虚拟机, 下载或使其从设备上移除, 但普通用户不能重新配置, 这对于虚拟机系统来说有很好的安全保障.

2.2 Android 显示框架设计

由于本文中所使用的硬件平台是共享单个 CPU、GPU 以及内存设备, 但拥有两块不同的显示屏幕, 因此对多个系统同时显示在不同屏幕上单独设计方案.

Android 采用分层的架构设计, 将系统分为四层: 应用层, 应用程序框架层, 运行库存, Linux 内核层. Android 对 Linux 内核驱动程序进行封装, 提出硬件抽象层, Android 的显示系统硬件抽象层提供的是 Gralloc, 分别负责对 framebuffer 以及帧缓冲区的分配和释放. Linux 内核驱动层提供了 FrameBuffer 显示驱动程序, 其设备驱动程序的设备文件通常放在/dev/graphics/fb0 或者是/dev/fb0.

利用不同显示设备在系统中存在不同的设备节点, 以及宿主 Linux 和客户机 Android 系统的特性, 提出双屏显示方案. 其架构如图 2. 对于宿主 Linux 系统使用 QT 交互界面默认使用/dev/fb0 对应的屏幕设备, Android 客户机系统通过 LXC 容器启动, 将 Android 的显示画面输出到 fb1 对应的屏幕中, 从而实现双屏同时显示 Qt Linux 系统与 Android 系统画面.

在 Linux 中, FB 的使用模式是通过两种的访问类型: mmap 和标准的 ioctl 操作. 不同虚拟机对底层不同的 FrameBuffer 驱动进行映射, 同时底层 FrameBuffer 的驱动在内存分配上也是相互隔离的. 这样就使得不同系统的访问流相互隔离, 从而达到显示隔离的效果.

最后 GPU 将渲染的数据直接写入对应的 buffer 中, 与 CPU 处理的内存并不同, GPU 是直接对 FrameBuffer 进行操作, 与 CPU 处理的数据是相互独立的. 相较于在 Android 的服务层做修改, 本方案不会改变 Android 的通用接口, 使得实现方案更加具有通用性. 同时本方案将 Android 系统应用程序的数据直接显示在 fb1 上, 使得多系统显示数据得到很好的隔离. 并且不同系统均可以正常显示, 相互之间没有影响.

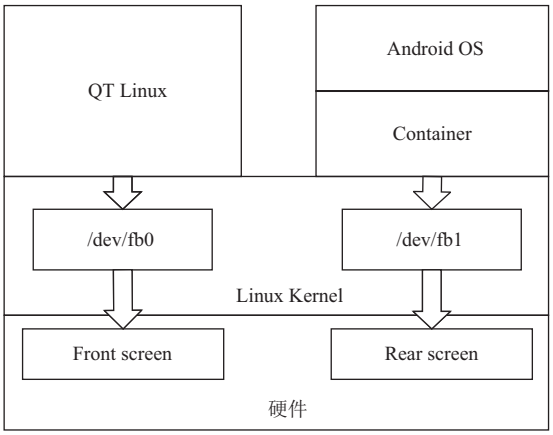

图 2 双屏显示方案

3 实现细节

为了实现当前的设计方案, 需要完成以下工作:

(1) 对 Linux kernel 进行定制, 开启 Namespace 机制和 Cgroup 机制在内核中的选项. 使用指定的交叉编译工具, 移植 LXC 工具到宿主系统中.

(2) 制作 Android 容器文件系统, 使得 LXC 可以启动客户机系统.

(3) 完成 Android 系统对显示的需要, 对不同系统显示进行隔离.

3.1 LXC 移植

使用 LXC 的工具的前提是 Linux 内核支持 Namespace 机制和 Cgroup 机制, 虽然现在 Namespace 和 Cgroup 加入到内核中, 但是其默认功能是没有开启的. 需要手动打开 Kernel 对应的选项, 并重新编译内核, 目前使用的 Kernel 版本是 3.4.35.

由于使用定制的 ARM 平台交叉工具, 所以编译过程会和普通 GCC 编译不同, 需要的依赖项也不同. 为了提高通用性, 在脚本中加入对各个平台依赖的判断. 这里使用的编译工具是 armv6-gcc, 为了使 LXC 工具正常编译, 系统需要安装 automake 等工具, 提前编译 libcap 库. 但为了并对 LXC 源码进行了修改, 其主要修改的内容有:

(1) 与 Android 系统不兼容部分, 例如 setenv, share, tmpfile 等函数, 在 Linux 平台与 Android 平台中, 其基础函数库参数以及实现方式不同, 从而影响 LXC 正常执行.

(2) 无法正常编译的问题, 例如部分宏出现未定义问题, 需要增加相应的头文件.

(3) 相关逻辑的修改, 使得可以启动 Android 容器, 且不需要修改通用的 API 接口.

执行 make 以及 make install, 将交叉编译好的 LXC 工具移植到 ARM 平台的机器中, 通过使用 LXC 工具集中的 lxc-checkconfig 来检测 kernel 选项是否打开以及其他功能是否运行正常. 并且可以通过尝试创建一些模板容器, 检查 LXC 是否可以正常运行.

3.2 容器文件系统制作

根据 LXC 容器创建过程以及 Android 系统启动分析, 由于存储容量有限, 且原生的 Anroid 文件系统需相应的修改, 才可以在 Linux Container 中正常的运行.

宿主机系统采用普通的嵌入式 Linux 系统, 则为了启动 Android 客户机系统, 使用 LXC 启动命令 (lxc-start) 将客户机系统顺利启动, 需要在 Linux 系统中创建容器系统. 在 LXC 工具指定的路径建立容器的名称, 不同的客户机的文件目录是相互隔离的. 同时也可以通过设置共享相应的客户机的库文件等. 不同的目录对应不同的客户机系统, 这样运行时不同的客户机系统之间无法感知其他系统的文件系统.

主要的文件目录如图 3, 对于宿主机系统都有其容器名, 其中 rootfs 为其 Android 客户机系统的目录文件. 其目录下包含 Android 系统启动必须的文件, 修改原 Android 系统的 init.rc 文件, 关闭不必要的服务, 修改其创建的设备节点用来与宿主系统隔离和复用, 修改挂载的文件系统, 复用宿主系统的文件, 提升低端设备的扩展性, 以及隔离需要隔离的文件目录. 通过自动化脚本配置容器信息, 提升容器配置的通用性.

3.3 双屏显示

由于本文中 ARM 平台的硬件设备是一个 CPU, Memory 但是拥有两个屏幕硬件设备, 在 Linux 中每个应用程序通过 mmap 系统调用, 将显存映射到应用程序对应进程的虚拟地址空间中, 可以直接将所需要显示的内容直接写入内存空间, 通过实现一系列文件操

作接口, 使得应用程序可以直接操作字符设备, 最终 LCD 控制器自动将显存中的内容显示在屏幕设备中.

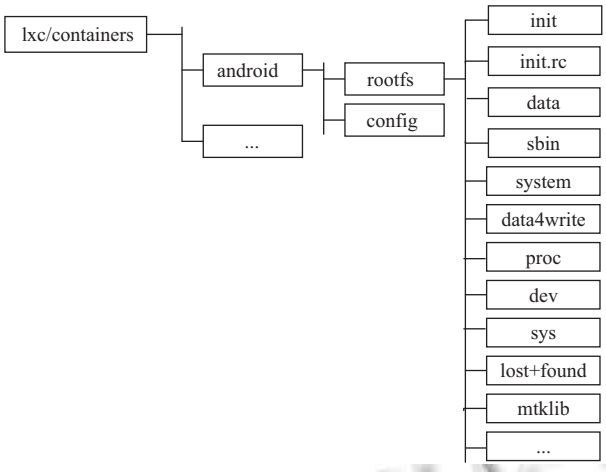

图 3 Android 容器目录文件结构

Android 系统在设计之初只考虑单屏的存在, 因此对于移动设备来说, 因为只存在一个屏幕, 在字符设备节点中默认只有/dev/fb0 或者/dev/graphics/fb0 存在. 对于 Android 应用程序, 通过 SurfaceFlinger 服务, 调用 libgl 库进行图像处理, 将显示数据交给硬件逻辑层 Gralloc, 最终底层 FrameBuffer 驱动以及 GPU 驱动将需要显示的内容刷新在显示设备中. Gralloc 对于复杂的显示驱动, 保证 Android 上层应用显示接口不变. 因此选择对 Android 硬件抽象层 Gralloc 中作出相应的修改, 实现 Android 客户机系统可以正常显示在第二块屏幕上.

通过对 Android 系统的源码分析, 用户空间使用 FrameBuffer 设备, 首先通过 load 加载 Gralloc 模块. 从 Gralloc 层打开 fb 设备的流程可知, 在保证 Android 显示正常, 只需要对 Gralloc 模块加载的 fb 设备作出相应的修改就可以实现双屏显示的方案, 因此在该硬件抽象层加入相应的逻辑修改, 其主要工作为:

(1) 将 Gralloc 模块关于打开 fb0 设备的逻辑做相应的修改, 使得加载 Gralloc 模块时将不是默认打开 fb0 设备, 使得 Android 设备可以在 fb1 对应的显示屏能够正常显示.

(2) 拒绝容器打开的客户机系统对 Linux 中对应 fb0 的字符设备节点打开的权限, 使得从客户机系统角度来看也只有一个显示设备节点存在.

(3) 在 LXC 启动容器的流程中, 加入对 fb1 设备加载的逻辑. 考虑到 fb1 设备一直打开, 损失移动设备的电量, 在 lxc-start 的启动流程中加入对 fb1 设备加载的相应逻辑, 达到节约能耗的目的.

4 系统成果以及功能测试

实验测试环境是在 ARM 平台车载系统开发板上, 使用 Linux kernel 3.4.35 版本, Android 对应 4.2.2 版本, 基于 LXC1.0.7 的虚拟化解决方案, 其硬件实验环境为: 双核 ARM Cortex A7、1GB DRAM、8GB eMMC. 因其平台硬件资源有限, 只启动双系统进行测试.

4.1 功能测试

通过烧制系统集成 Image, 打开电源开关, 宿主 Linux 系统和客户机 Android 系统同时启动, 宿主系统显示其开机 Logo, Android 系统在另一个显示器显示“ANDROID”字样, 代表两个系统通过宿主系统的命令启动成功; Android 系统通过 LXC 工具正常启动, 且通过两个屏幕分别显示不同系统界面, 互相隔离相互不影响, 表明双屏显示方案正常; 且通过 45 项系统以及硬件功能的 Sanity Test 测试.

同时通过多次的开关机重启, 双系统同时画面几乎同时启动且正常显示, 表示其功能正常且稳定. 且鼠标, 键盘输出输出设备不会在 Android 系统中响应, 只能点击和响应 QT 的界面, 同时对于 Android 系统使用自带的触摸屏去触摸点击, 可以正常交互和显示.

4.2 性能测试

通过读取/proc/meminfo 信息统计不同系统情况下, 内存的使用的情况, 在开机 30 分钟稳定后得到的数据情况如图 4.

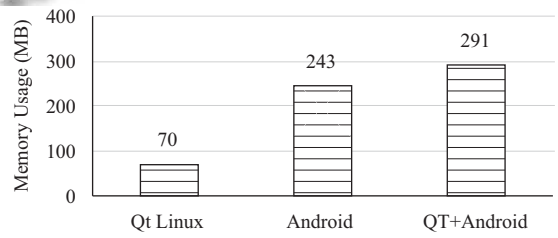

图 4 内存使用情况

我们根据 2016 年 Google 热门 app 榜单, 选取社交类、影音类、游戏类三类, 分别测试微信、音乐播放器、部落冲突, 代表不同的应用场景, 将虚拟化后的 CPU 使用率与原生 Android 的 CPU 使用率进行对比, 得到 10 秒的时间内的增长率, 详情如表 1 所示.

从表中可以看出, 不同种类的应用的 CPU 使用率

的增长情况不同, 但总体情况类似. 相比较来说可以看出原生的 Android 系统在启动 APP 时的速度较快一些, 不过两者的 CPU 使用率相差不大, 在实际使用的场景下基本感受不到两者的差距. 根据三种应用的使用率情况看, 其虚拟化后的 Android 系统的平均 CPU 使用率仅比原生系统高 1%, 性能与原生系统较相近.

表 1 三类 app 的 CPU 利用率变化

	微信app(%)	音乐app(%)	部落冲突(%)
1 s	3.23	1.47	4.25
2 s	2.35	3.01	-3.84
3 s	-2.32	-3.89	1.22
4 s	-0.49	3.03	-2.66
5 s	1.63	1.82	0.41
6 s	2.62	2.41	1.83
7 s	-1.34	1.55	0
8 s	2.71	5.20	-2.15
9 s	-1.26	-2.78	1.52
10 s	2.93	3.21	1, 41

5 结束语

针对车载系统等平台多屏显示的需求, 本文提出一种基于容器技术 Android 轻量级虚拟化方案. 相较于传统多核虚拟化方案, 本方案只需要单个内核即可支持多个 Android 容器, 具有更低的资源开销和更高的性能. 从而实现用户对不同场景以及隐私信息安全的需求. 测试结果表明: 在 ARM 平台上, 该方案的内存占用率较双系统方案降低了 7%, 而平均 CPU 使用率较原生 Android 系统仅增加了 1%. 但是本文所提出的双屏显示方案还不够完善, 对不同输入输出设备的隔离和复用采取不同的策略, 下一步的将围绕隔离以及复用框架的通用性展开进一步工作.

参考文献

1 Armbrust M, Fox A, Griffith R, *et al*. A view of cloud computing. Communications of the ACM, 2010, 53(4): 50–58. [doi: 10.1145/1721654]

2 陈全, 邓倩妮. 云计算及其关键技术. 计算机应用, 2009, 29(9): 2562–2567.

3 TrendForce: 2015 年全球智能手机出货 12.93 亿部, 华为跃升全球第三并突破一亿部. http://press.trendforce.cn/press/20160114-2266.html. [2016-01-14].

4 Enck W, Ongtang M, McDaniel P. Understanding android security. IEEE Security & Privacy, 2009, 7(1): 50–57.

5 杨健, 汪海航, 王剑, 等. 云计算安全问题研究综述. 小型微型计算机系统, 2012, 33(3): 472–479.

6 蒋绍林, 王金双, 张涛, 等. Android 安全研究综述. 计算机应用与软件, 2012, 29(10): 205–210.

7 Android Platform. https://developer.android.com/about/android.html.

8 Aguiar A, Hessel F. Current techniques and future trends in embedded system's virtualization. Software: Practice and Experience, 2012, 42(7): 917–944. [doi: 10.1002/spe.v42.7]

9 Andrus J, Dall C, van't Hof A, *et al*. Cells: A virtual mobile smartphone architecture. Proc. of the Twenty-Third ACM Symposium on Operating Systems Principles. Cascais, Portugal. 2011. 173–187.

10 Uhlig R, Neiger G, Rodgers D, *et al*. Intel virtualization technology. Computer, 2005, 38(5): 48–56. [doi: 10.1109/MC.2005.163]

11 Varanasi P. Implementing Hardware-supported virtualization in OKL4 on ARM[Ph. D. thesis]. New South Wales: University of New South Wales, 2010.

12 Kivity A, Kamay Y, Laor D, *et al*. kvm: the Linux virtual machine monitor. Proc. of the Linux Symposium. 2007. 225–230.

13 LXC-Linux Containers. https://linuxcontainers.org/, 2014.

14 Chen XY. Smartphone virtualization: Status and challenges. 2011 International Conference on Electronics, Communications and Control (ICECC). Ningbo, China. 2011. 2834–2839.

15 毛德操, 胡希明. Linux 内核源代码情景分析 (上册). 杭州: 浙江大学出版社, 2001.

16 Rosen R. Linux containers and the future cloud. Linux Journal, 2014, 2014(240): 3.