ARM-MuxOS：一台手机，多个世界

余宽隆 陈 渝 茅俊杰 张 磊
（清华大学计算机科学与技术系 北京 100088）

摘 要 在移动设备上并发运行多个操作系统，可拓宽和多样化其使用模式，但目前采用的移动虚拟化管理系统技术会带来性能开销和多余的内存消耗。通过分析在单一移动设备上支持多个操作系统所带来的多OS内存管理和外设分配等方面挑战，研究并设计了物理内存在线分配和分时复用外设等新技术，本设计在Galaxy Nexus智能手机上最终实现了ARM-MuxOS原型系统。这一系统不仅可在单一移动设备上支持多个操作系统，而且可在内存较少的环境下管理多个OS的内存分配，避免了传统虚拟化技术的性能开销与工程量。实验结果表明，ARM-MuxOS原型系统不仅能支持Android与FireFox OS的快速并发执行，而且其性能和内存消耗优于现有的移动虚拟化管理系统。

关键词 操作系统，移动设备，内存管理，虚拟化
中图法分类号 TP316 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.10.002

ARM-MuxOS: A System Architecture to Support Multiple Operating Systems on Single Mobile Device

YU Kuan-long CHEN Yu MAO Jun-jie ZHANG Lei
(Department of Computer Science and Technology, Tsinghua University, Beijing 100088, China)

Abstract Enabling concurrent execution of multiple operating systems on mobile devices greatly extends their usage model. Mobile virtualization provides such functionality, but has poor performance. We first analyzed the challenges of allocating physical memory and sharing hardware devices among multiple general purpose operating systems on a single mobile device, designed new methods to answer these problems, and implemented a prototype of ARM-MuxOS on a Galaxy Nexus smartphone, which can support multiple operating systems running concurrently on it, cleverly manage its limited memory across many operating systems, and avoid the performance overhead of mobile virtualization and its required high engineering effort. Our test results show that ARM-MuxOS supports Android and Firefox OS and with an almost native performance, and it is better than current paravirtualization-based methods.

Keywords Operating systems, Mobile device, Memory management, Virtualization

1 引言

购买移动设备时，消费者通常需要在几个移动OS之间做出选择（Android、IOS、Windows Phone、WebOS、Ubuntu Phone、Tizen、Firefox OS等），因此用户被限于单个OS及其应用程序生态系统。为了减少限制，我们提出在一台移动设备上运行多个不同的OS，使个人可同时访问几个应用程序生态系统。这种方案不仅可以方便开发人员在不同的OS版本中快速检测他们的应用程序，还可进一步实现快速部署、系统迁移等不同功能。移动设备的特点是硬件设备和OS的多样化。为了支持在一个移动设备上运行多个OS，当前最主要的方法是移动虚拟化。针对主流ARM移动设备的移动虚拟化相关技术是一个非常热门的研究领域[1-4,6,7]。

现在可用的ARM移动设备还没有虚拟化硬件支持，这促进了半虚拟化管理程序（Virtual Machine Monitor，VMM）的发展[1,3,4,7]。但为使移动设备在半虚拟化环境下运行，需要做出大量软件修改工作。而且根据虚拟化的效率与应用程序的性质（如系统调用的频率），半虚拟化会导致操作系统与应用程序的执行显著变慢。这是因为虚拟化会造成内存访问、I/O性能、缓存效率等的软件开销[5]。该开销浪费了操作系统尤其是后台闲置操作系统的处理器执行开销，导致功耗的增加。

传统操作系统需要管理整个硬件设备上的物理资源（如处理器、定时器等）。如果在一个硬件设备上支持几个OS并发执行，则存在硬件设备被几个操作系统访问的情况。传统操作系统独占这些硬件资源，没有考虑共享问题，所以无法并发运行。多个操作系统为了复用到硬件物理资源，需要通过一个虚拟层导出虚拟设备接口给客户OS，才能捕获并管理I/O访问。现有的方法是让硬件设备的实际控制由某一个特权操作系统[2,7]或虚拟层中用以支持虚拟化的驱动[7]来负责，但这会导致在一个新平台实现虚拟化需要付出大量开发工作来支持所有的硬件设备，使得实现难度较高且工作量较大。

在不基于虚拟化的复用硬件设备方法的情况下，仍可通

到稿日期：2013-06-20 返修日期：2013-08-25 本文受国家自然科学基金项目（61170050），核高基项目（2012ZX01039-004）资助。
余宽隆（1988－），男，硕士，主要研究方向为嵌入式系统，E-mail：alexis.yuhe@gmail.com；陈 渝（1972－），男，博士，副教授，主要研究方向为操作系统；茅俊杰（1989－），男，博士生，主要研究方向为操作系统、虚拟化；张 磊（1978－），男，博士，工程师，主要研究方向为普适计算、虚拟化。

过其他方法使设备被多个 OS 访问，例如，在服务器和桌面领域所采用的特殊硬件设备的多路设计，或者利用特殊硬件设备（如 DMA，中断控制器）来专门调解 I/O 访问[9]。但鉴于移动设备的硬件情况，前一个方法并不适用；而第二个方法却需要针对不同问题提供不同的解决方法。通过对移动设备的使用模式进行分析，我们发现移动设备的使用模式与桌面服务器的使用模式不尽相同。在移动设备上，任何时间，用户一般只能操作单个操作系统的少量应用，甚至是一个应用。因此，在绝大多数情况下，多个操作系统无需并行执行，而是通过并发执行来合作以分时共享移动设备上的各种物理资源。我们认为，为解决在单个移动设备上支持多个操作系统这一问题，可通过修改操作系统本身来避免虚拟化的复杂性。本文针对在单个移动设备上支持多个操作系统这一问题提出了我们的设计思路，即调整 OS 的物理内存管理、调度和外设管理，允许操作系统共享硬件物理资源，从而实现 OS 自身的切换，完成 ARM-MuxOS 框架原型系统，验证设计思路的可行性。此外，我们还实现了 OS 内存管理和分配的动态优化来提高用户交互的体验。我们选择 ARM 智能手机 Galaxy Nexus 作为硬件平台实现了上述技术并完成了 ARM-MuxOS 原型，这一原型可支持 Android 和 Firefox OS 并发执行与快速切换。与移动虚拟化相比，我们的设计选择不仅不会降低性能，还可以保持设备的兼容性，并且其代码量仅占移动虚拟化所需代码量的 1/5。

2 ARM-MuxOS 框架的设计与实现

2.1 操作系统并发执行原理设计

传统移动设备的设计与使用方式没有考虑多 OS 同时交互。其实，当几个 OS 同时运行时，用户只集中于屏幕上显示的 OS（前台 OS），而其他 OS（后台 OS）在绝大多数情况下可以暂停。用户若需要暂时运行另外一个后台 OS，首先要暂停执行前台 OS，而后恢复执行此后台 OS。这种运行方式即 OS 并发执行方式，适合移动设备的通常使用模式。为避免此前方案所遇困难，我们基于 OS 并发执行方式设计了一个能支持多个 OS 的系统架构：ARM-MuxOS 可协调多个 OS 的相互作用，允许多个 OS 共享同一平台（见图 1）。

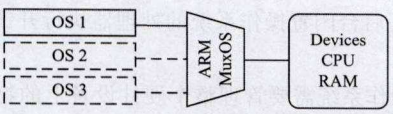

图 1 ARM-MuxOS 的概念：OS1 作为前台 OS，OS2 与 OS3 作为后台 OS

处理器管理 前台 OS 控制所有的处理器核。要把执行时间分配给多个 OS，首先需要保存前台 OS 的处理器上下文，而后加载另外一个后台 OS 的处理器上下文。这个过程类似处理器上下文切换。处理上下文的保存以及加载，与 OS 细节无关，只与硬件有关。我们把负责这个任务的功能模块称为 monitor，它既可以嵌入到一个操作系统的内核里，也可以实现为一个单独的软件（如 bootloader[8]）。

硬件设备管理 OS 并发执行模式只允许前台 OS 访问硬件设备。因此，OS 与硬件之间不需要任何软件层：每个 OS 使用本身自带的设备驱动来确保本机的性能与设备的兼容性。但为确保成功地切换系统，并发执行需满足以下两个条件：

1）对于每个硬件设备，每个 OS 带有自己的设备配置。当一个后台 OS 接管了移动设备的控制，它一定会修改设备状态，覆盖任何先前未保存过的配置。因此，切换时每个 OS 需要完整地保存自己的设备状态，否则恢复时，硬件设备会因为配置数据不足或不正确而出现无序状态。

2）前台 OS 暂停时把硬件设备设置到某个暂停状态，而后后台 OS 恢复时必须将此设备暂停状态恢复。如处理不当（如某些设备在没预料的状态），会导致硬件设备的控制转移失败，平台可能出现死机或异常状况。因此，所有参与 OS 必须有相同的设备暂停状态。

现有的移动软件环境适合 OS 并发执行模式。移动应用程序的使用模型规定，这些软件可以随时被打断（如来电话时）。因此，前台 OS 可以随时安全转移到后台，应用程序也会随之自动暂停。与此同时，处于暂停状态的移动设备会通过进入休眠方式来节省电量。这是由于在暂停状态下，前台 OS 会首先将处理器上下文与大多数硬件设备的状态保存到物理内存里，而后关闭硬件设备的电源。恢复时，OS 会从内存加载之前所保存过的状态与上下文。这是一种常见的电源管理功能，它在 Linux 术语中叫待机机制，即 suspend-to-RAM（STR）。而其他通用 OS 也具有相同功能。

我们通过扩展 STR 框架来实现 ARM-MuxOS。首先，如图 2 与图 3 所示，前台 OS 快速保存设备状态与处理器上下文到某一个固定的物理内存空间（Suspend-And-Resume，RAM，待机内存），进入暂停状态（(a)）。由于后台 OS 恢复时也是从待机内存加载上下文与设备状态，因此 ARM-MuxOS 首先修改待机内存的内容，用后台 OS 状态代替前台 OS 状态（(b)）。而后，后台 OS 进行正常的恢复过程（(c)），最终变成前台 OS。ARM-MuxOS 会保存处理器上下文与设备状态到一个所有 OS 能共用的内存区，称为 monitor RAM。

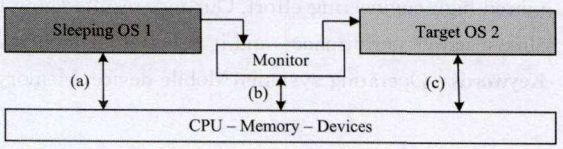

图 2 系统切换的概念

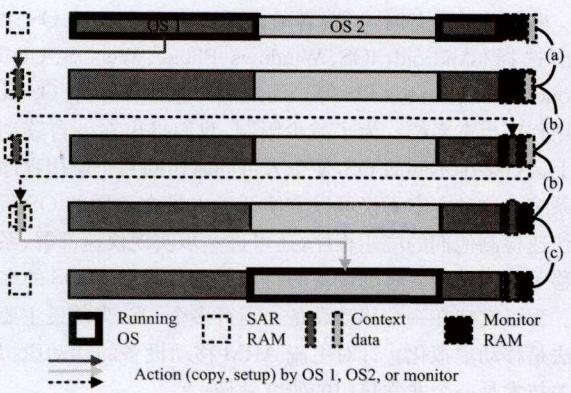

图 3 系统切换内存状态

通常情况下，驱动会保存完整的设备状态并关闭硬件设备的电源。因此，为避免修改驱动，所涉及的 OS 一般都要有一个相同的暂停状态。如果两个 OS 之间的设备暂停状态有差别，那必须修改驱动或者切换时让 monitor 来修改差别（(b)）。

2.2 物理内存在线分配优化

在单一移动设备上运行多个 OS，引发了两个关于物理内

存的问题：如何在多个 OS 之间组织与分配内存以提高前台 OS 的执行效率，以及如何防止外部对 OS 内存的非法访问以确保安全性。

首先，每个 OS 在其存在周期中都在物理内存中存在，并且必须通过明确的内存边界来避免与其他 OS 重叠。一种普通的方法是静态地把物理内存分块管理，而后根据需求把空闲物理内存块分配给每个 OS[8-10]。此方法虽可用，但效果较差。其原因如下：

1)OS 的数量不能超过内存块的数量；

2)用户不会随时需要几个 OS，因此没用的 OS 会浪费内存；

3)一个 OS 内存不够时无法使用其他 OS 的内存。

物理内存分配管理　由于移动设备的内存资源有限，上述方法会导致前台 OS 无法充分使用物理内存，从而影响人机交互体验。因此，更好的方法是在启动一个新 OS 时才分配内存、随时控制内存的大小，且一旦停止使用一个 OS 就马上回收它所占用的内存。

操作系统内核中内存热插拔(Memory Hotplug)功能，允许 OS 在运行时为了扩大本机内存或更换坏掉的物理内存条而随时添加和删除内存。以 Linux 内核为例，其内存热插拔功能模块采用稀疏内存模型(Sparse Memory Model)，此模型把内存划分为固定大小的内存块，每个内存块在运行时可以单独启用或禁用。当一块内存被禁用时，其内存中的内容通过页迁移机制迁移到另外一个可用的内存块。Linux 内核中的关键内存(如包含内核代码或被锁住的内存)是不可迁移的，此内存只属于一个 OS。而其他物理内存属于可迁移的内存，基本上都可迁移。我们通过扩展内存热插拔功能，使得 ARM-MuxOS 可动态控制不同 OS 所需内存。ARM-MuxOS 设置了可在不同 OS 中分时复用的可迁移内存块，这些内存块可根据应用需求随时从一个 OS 转移到另外一个 OS。

为了避免物理内存布局的重叠，ARM-MuxOS 设置了物理内存布局表来注明内存块和其所属 OS 的关系(如图 4 所示)。修改任何 OS 的内存布局都需要首先检查内存表，而 monitor 可以负责此任务。唯一允许几个 OS 共用一个内存块的情况是驱动要求固定内存地址的读写权利。此种情况并不会出现访问异常，因为切换时驱动可以把内存块的内容保存好(如待机内存)，内存块的内容也可以被安全地重写(如 framebuffer)。为实现更安全的物理内存隔离，ARM-MuxOS 在接下来的工作是在支持 Trustzone 或虚拟化硬件扩展机制的智能手机中实现物理内存安全隔离。

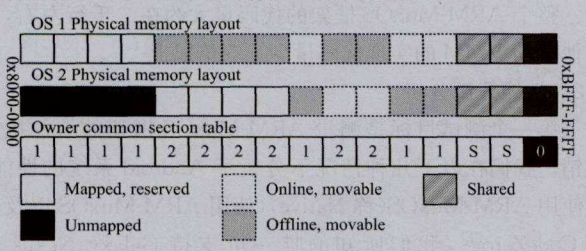

图 4　内存块分配与内存图表

2.3　系统运行流程

如图 5 所示，我们选择了把 monitor 机制实现在 Linux 内核中：每个 OS 自身带一个 monitor，按照 OS 的自身需求保存处理器上下文与设备状态，而通过 ioremap() 访问 monitor RAM 来保存各种信息。同时，每个 monitor 也导出一个 sysfs 接口，方便用户通过普通 shell 命令来进行操作：我们的原型利用手机按钮来进行一系列的 shell 脚本。

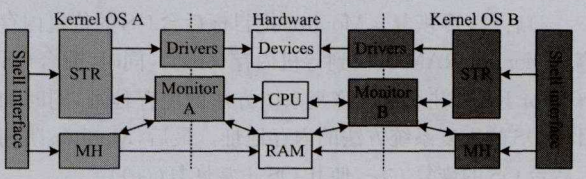

图 5　ARM-MuxOS 总体架构(例子中 OS 为两个)

如下是我们原型所提供的方法的流程描述。

2.3.1　系统加载与启动

流程主要有 3 个阶段，如图 6 与图 7 所示。

1)准备内存。在 ARM 平台上启动 Linux 内核，不仅需要在编译内核时决定内核代码入点地址，也需要用户在启动前决定内存布局。为将其内存块分配给目标 OS，前台 OS 首先要禁用这些内存块，然后将目标 OS 的内核映像(kernel image)、initrd 等加载到这些内存块。随后前台 OS 可进入待机状态。

2)准备硬件。前台 OS 在进行暂停过程中，硬件设备处于暂停状态，但其状态不适合启动目标 OS。因此，monitor 执行 CPU 的热重启，使设备进入启动过程。Bootloader 会初始化硬件设备，使其状态符合目标 OS 的启动需求。

3)启动目标 OS。Bootloader 在启动的过程中，在之前的前台 OS 的内存中加载初步 OS 的内核映像((d))。我们提前禁止了初步 OS 使用被覆盖的内存地址，因此避免了内存异常访问((a))。加载内核映像后，bootloader 进入映像的入点。我们提前修改映像头代码，让处理器从 monitor RAM 读取目标 OS 的内核映像地址，而后跳到此地址。目标 OS 启动后((e))，将按照物理内存布局表回收释放过的内存块，最终完成启动过程。

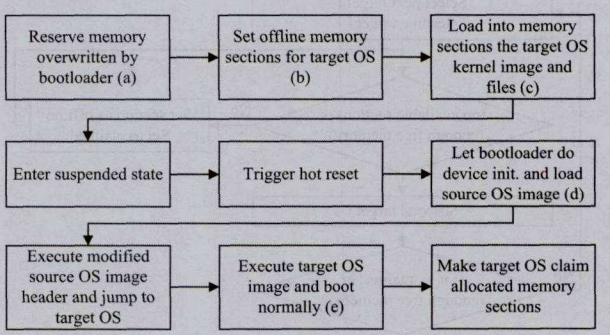

图 6　系统加载与启动流程

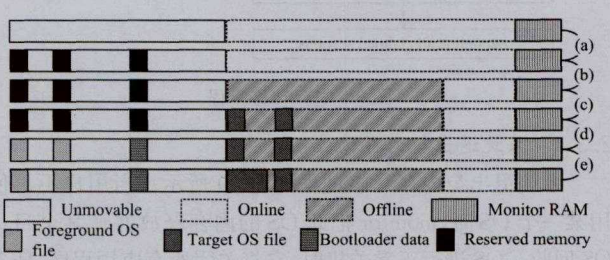

图 7　系统加载与启动内存状态

2.3.2　系统切换

系统切换流程主要有两个阶段，如图 8 与图 3 所示。

1)前台 OS 休眠。前台 OS 在进行暂停过程中((a))，硬

件设备会先将上下文保存到待机内存，随后进入暂停状态，进而 monitor 调整某些设备的状态(如取消 CPU 的等待待机状态)，最后 monitor 把待机内存内容保存到 monitor RAM。

2)后台 OS 恢复。Monitor 把目标后台 OS 的待机内存内容从 monitor RAM 载回到待机内存((b))。同时，前台 OS 在 monitor RAM 注册自己恢复系统方法的内存地址，同时加载目标 OS 的恢复系统方法的内存地址。最后，Monitor 跳到目标后台 OS 的恢复方法，使其 OS 正常恢复((c))。

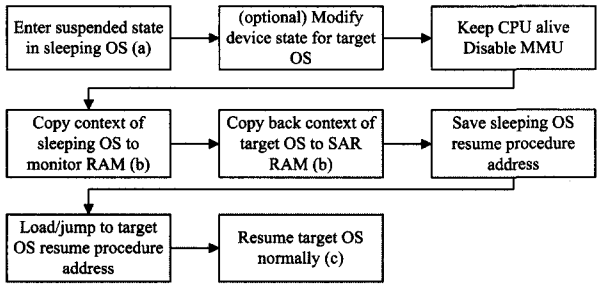

图 8　系统切换流程

2.3.3　内存转移

内存移转流程如图 9 所示。前台 OS 向 monitor 申请一个或多个内存块，进而进行暂停过程。通过访问物理内存布局表后，monitor 可找到所有后台 OS 可释放的内存块，并选择其中一个。此后台 OS 恢复内核(但不恢复应用程序)，释放内存块，随后立刻进入暂停状态。如果此后台 OS 无法释放足够的内存块，monitor 再寻找另外一个后台 OS 进行上述过程，直到释放出足够内存。最终，monitor 跳到前台 OS 的恢复处理过程，前台 OS 即可回收后台 OS 刚释放的内存。

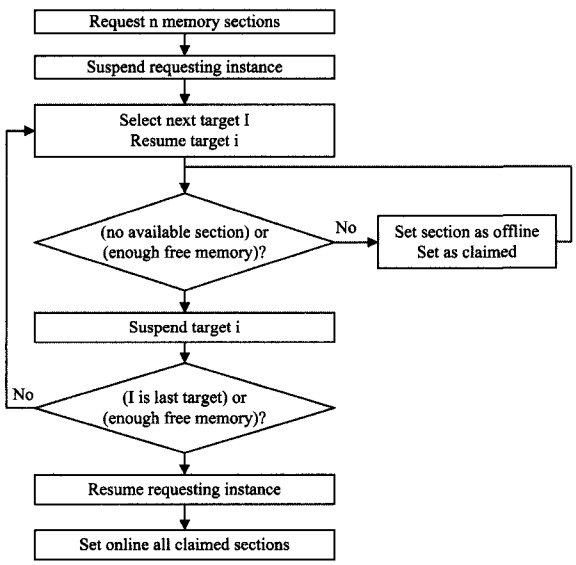

图 9　内存转移流程

2.3.4　系统退出与内存释放

系统退出与内存释放过程如图 10 所示。当用户不再使用某一个 OS 时，monitor 把此 OS 的内存释放并让某个目标 OS 回收。OS 进行正常关机时，它将关闭所有应用程序、任务与文件系统。此时 OS 进行暂停过程，随后硬件设备处于暂停状态。Monitor 记录 OS 所用的内存块并在物理内存布局表中将它们标记为可用。Monitor 跳到后台目标 OS 的恢复方法：硬件设备状态符合目标 OS 所需状态，因此目标 OS 可以正常恢复。目标 OS 最终回收前一个 OS 的内存块并照常运行。

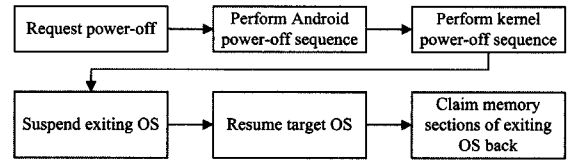

图 10　系统退出与内存释放

3　可行性与性能分析

3.1　可行性分析

我们在 Galaxy Nexus 手机硬件平台上的 Android(4.1)与 Firefox OS 上实现了 ARM-MuxOS 框架，此二者的 OS 都基于 Linux 3.0.x 内核，如图 11 所示。在可用内存为 700Mb 的情况下，框架允许动态启动任何 OS，并控制整个内存与硬件，进而可以加载与启动其它 OS，管理内存的程序成功分配、转换、释放与内存的回收。Android 与 Firefox OS 可以运行自己应用程序的生态系统(Android 与 HTML5 应用程序)。尽管目前 Wi-Fi 以及照相机功能在某些情况下有异常情况出现，但是基本不影响使用。完成不同操作系统切换的时间开销可控制在 3 秒之内，而平均的操作系统切换开销为 1.7 秒，这几乎不影响用户的人机交互体验。

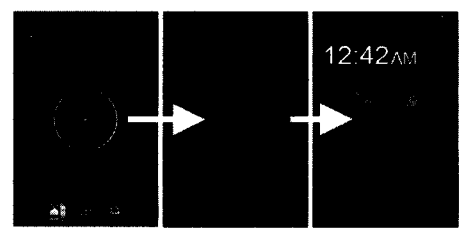

图 11　系统切换从 Android 到 Firefox OS 时的用户体验

我们扩展了 Linux 内核的 STR 模块和内存热插拔模块(自 Linux 2.6.23 就已存在)来实现 monitor；大部分代码已经属于与硬件无关的代码，可以用于其他硬件平台，包括 x86。与硬件有关的一小部分代码(包括 STR 模块最底层的代码)需要添加一些补充。一般的移动设备驱动支持待机状态，所以我们无需再次修改或补充这些驱动。我们也同样使用了 ARM 虚拟地址与物理地址转换模块(自 Linux 2.6.39 就已存在)。因此，ARM-MuxOS 不仅可以在其他 Linux 版本实现，也可以被快速移植到其他硬件平台。

整个 ARM-MuxOS 框架的代码量大约在二千行左右，比其他移动 VMM 的实现代码量少了 5 倍以上[1,7]。

3.2　性能开销分析

第一个测试目标是测量 ARM-MuxOS 框架的软件执行开销。我们在以下几种情况下分析了 Android 系统的性能：不使用 ARM-MuxOS(称 Native)，使用 ARM-MuxOS 以及使用 CodeZero[1](我们所知的唯一可支持 Galaxy Nexus 的 VMM)。我们使用了几个通用基准测试来评论 CPU、内存、GPU 与 I/O 的性能开销，并使用了 PowerTutor 来测量功耗。如图 12 所示，实验数据表明 Native 执行性能与加载了 ARM-MuxOS 框架的执行性能的差别不超过 1%，几乎可忽略不计。影响性能的原因是稀疏内存模型稍微拖长了内存访问。

但加载了 CodeZero 的 Android 由于更多的 VMM 软件执行开销导致其整体性能平均降低 20%。

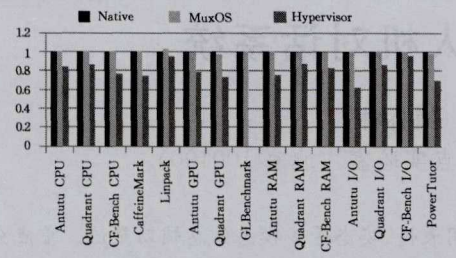

图 12　ARM-MuxOS 与 VMM 性能对比

第二个测试目标为测量支持多个 OS 时，尤其是当减少了一半内存后，是否会影响各个 OS 的性能。我们在不使用 ARM-MuxOS(整内存、半内存)和使用 ARM-MuxOS(OS1、OS2)情况下基准了 Android 的性能。如图 13 所示，按实验数据，对比不使用 ARM-MuxOS 的两个情况，可发现 RAM 减少一半时基本不会影响基准的结果(少于 0.5%)。对比不使用 ARM-MuxOS 与使用 ARM-MuxOS 两个情况，可发现性能降低了 1%，此结果与前一个测试结果相似。对比两个使用 ARM-MuxOS 的 OS，第二个 OS 性能降低了 4%以下。由此得出，两个 OS 内存布局的差别会影响性能。测试数据也表明 ARM-MuxOS 不会影响性能。

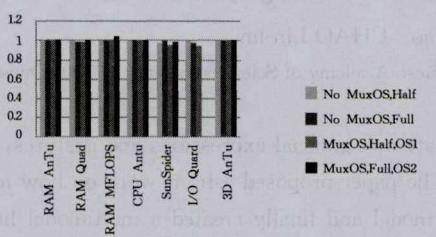

图 13　ARM-MuxOS 性能测试结果

虽然 ARM-MuxOS 本身不影响性能，但在同时运行多个 OS 过程中，移动设备的物理内存被划分，这使得每个 OS 的内存比单个 OS 的内存少了至少一半。因此，若想保持应用程序的性能，必须保证每一个 OS 有足够的 RAM(例如 Android 4.0 的要求是 300Mb 以上)，否则 OS 将会不稳定。

3.3　系统切换时间分析

切换时间是用户体验的一个重要元素。我们测量了切换过程中的每一个阶段，从初步 OS 切换到第二个 OS 的时间，以避免两个不同 OS 时钟的时差。测量情况分为两种：闲置时与运行游戏时。

如图 14 所示，整个过程包括：程序暂停与恢复(delay、suspro、respro)和硬件暂停与恢复(early、susdev、cpuoff、limbo、cpuon、resdev)。结果分为 3 种情况：

1)通常情况下无问题出现，平均时间为 1.74 秒(程序0.7 秒，硬件 1 秒)。

2)当一些任务与程序没有准时释放自己的禁止休眠锁(wakelock)时，会使暂停过程推迟 0.5 秒到几秒。平均时间在 2.5 秒(程序 1.5 秒，硬件 1 秒)以上。

3)暂停过程偶尔失败(正常现象)，使暂停过程重新开始而被推迟 0.5 秒。平均时间为 2.38 秒(程序 1.1 秒，硬件 1.3 秒)。

总之，在 95%的情况下，切换系统可在 3 秒以内完成，在 65%的情况下可在 2 秒以内完成。

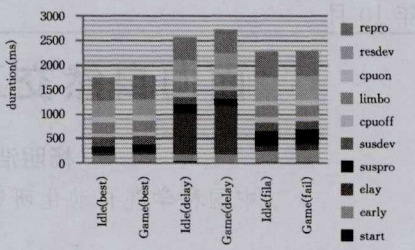

图 14　ARM-MuxOS 切换时间分析

4　相关工作

最近几年，针对 ARM 移动平台的半虚拟化有几个主要的系统管理程序[1,3,4,7]，它们的目标为：在一个平台上同时支持多个操作系统，简化移动设备部署过程，为应用程序提供安全运行沙盒等。然而，在半虚拟化的环境下，支持一个额外的移动设备需要进行大量工作，包括 OS 的移植与虚拟驱动的实现。ARM-MuxOS 是一个在单一移动设备上支持多个操作系统的优良解决方法。尽管 ARM-MuxOS 不允许多个 OS 并发运行，但是与半虚拟化相比，ARM-MuxOS 可以使用户有几乎相同的用户体验。此外，ARM-MuxOS 还可以保持硬件设备的兼容性与硬件性能。使用此功能时，原始设备制造商和爱好者并不需要与第三方(虚拟化管理程序提供者)合作，这是因为 ARM-MuxOS 只依赖于 OS 的自有功能。最后，ARM-MuxOS 的实现与移植较为容易，可以在其他众多移动设备上较快使用。

Cells[2]是一个系统架构，可同时管理在单一 Linux 内核上多个独立的 Android 用户空间。这篇论文介绍了一个硬件设备虚拟化技术，此技术可使多个用户空间能够同时复用硬件设备。此外，本文也介绍了为确保内存隔离与优化且不需要任何其他硬件支持所使用的技术。Cells 的设计只能在同一时间支持一个 OS 内核，而 ARM-MuxOS 的工作原理可同时支持多个 OS 内核，包括非 Linux 的 OS。

与 ARM-MuxOS 相似，为了在单一的无虚拟化平台环境下支持多个操作系统，“OS 切换”[8]也使用了 STR 框架的扩展，但后者是基于 bootloader 来实现 monitor 功能的。现在，智能设备的 bootloader 源代码没有公开，这使得“OS 切换”所使用的方法过时且无法实现。ARM-MuxOS 将 monitor 嵌入到 OS 中，使多个 OS 可相互管理，且不需要借助于第三方程序。此外，此方法也可以在线分配与隔离内存，这一点优于已存在的“OS 切换”技术。

NoHype[9]和 Mint[10]的工作原理为：在一个单一的 x86 机器上将处理器、内存、硬件设备分成不相交的子集，然后在每个子集上并行运行一个 Linux 内核。这两个方法不需要使用虚拟化。前一个系统是通过使用 ARM 平台所没有的特殊硬件来复用设备，而在后者的系统中，每个 OS 使用自有的硬件设备，所以这两者都不适用于 ARM 移动设备。而 ARM-MuxOS 所提出的在线内存分配的方法可改良 NoHype 和 Mint 所使用的技术。

结束语　本文通过介绍在单个 ARM 移动设备上支持多个操作系统的挑战，提出了非虚拟化的解决方案，其达到了更

(下转第 22 页)

法，QNR 的取值达到 0.89，说明本文提出的全色降质图像构造方法较好地满足了高保真比值变换融合所需的两个条件。

结束语 针对现有比值变换融合方法存在光谱和细节失真的问题，本文提出了一种新的全色降质图像构造方法，生成的全色降质图像可较好地满足高保真比值变换融合所需的两个条件。在此基础上，提出了基于比值变换的全色与多光谱图像高保真融合方法。在实验中，本方法与经典的比值变换方法进行了对比，实验结果表明本方法可以较好地避免经典比值变换融合存在的光谱和细节失真现象。同时，本方法融合图像的光谱与细节保真也优于当前应用效果较好的最新融合方法。

参 考 文 献

[1] Aplin P, Atkinson P M, Curran P J. Fine spatial resolution satellite sensors for the next decade[J]. International Journal of Remote Sensing, 1997, 18(18): 3873-3881

[2] Pohl C, van Genderen J L. Multisensor image fusion in remote sensing: concepts, methods, and applications[J]. International Journal of Remote Sensing, 1998, 19(5): 823-854

[3] Zhang J. Multi-source remote sensing data fusion: status and trends[J]. International Journal of Image and Data Fusion, 2010, 1(1): 5-24

[4] Thomas C, Ranchin T, et al. Synthesis of multispectral images to high spatial resolution: A critical review of fusion methods based on remote sensing physics[J]. IEEE Transaction Geoscience and Remote Sensing, 2008, 46(5): 1301-1312

[5] Alparone L, Wald L, et al. Comparison of pansharpening algorithms: Outcome of the 2006 GRS-S data-fusion contest[J]. IEEE Transaction Geoscience and Remote Sensing, 2007, 45(10): 3012-3021

[6] Rahman M M, Csaplovics E. Examination of image fusion using synthetic variable ratio (SVR) technique[J]. International Journal of Remote Sensing, 2007, 28(15): 3413-3424

[7] Gillespie A R, Kahle A B, Walker R E. Color enhancement of highly correlated images—II Channel ratio and 'chromacity' transformation techniques[J]. Remote Sensing of Environment, 1987, 22(3): 343-365

[8] Zhang Y. A new merging method and its spectral and spatial effects[J]. International Journal of Remote Sensing, 1999, 20(10): 2003-2014

[9] Liu J G. Smoothing Filter-based Intensity Modulation: A spectral preserve image fusion technique for improving spatial details[J]. International Journal of Remote Sensing, 2000, 21(18): 3461-3472

[10] Zhang Y. Understanding image fusion[J]. Photogrammetric Engineering & Remote Sensing, 2004, 70(6): 657-661

[11] Wald L, Ranchin T, Mangolini M. Fusion of satellite images of different spatial resolutions: Assessing the quality of resulting images[J]. Photogrammetric Engineering & Remote Sensing, 1997, 63(6): 691-699

[12] Massip P, Blanc P, Wald L. A method to better account for modulation transfer functions in ARSIS-Based pansharpening methods[J]. IEEE Transaction Geoscience and Remote Sensing, 2012, 50(3): 800-808

[13] Alparone L, Aiazzi B, et al. Multispectral and panchromatic data fusion assessment without reference[J]. Photogrammetric Engineering & Remote Sensing, 2008, 74(2): 193-200

[14] Aiazzi B, Baronti S, Selva M. Improving component substitution pansharpening through multivariate regression of MS +Pan data[J]. IEEE Transaction Geoscience and Remote Sensing, 2007, 45(10): 3230-3239

[15] Chien C L, Tsai W H. Image fusion with no gamut problem by improved nonlinear IHS transforms for remote sensing[J]. IEEE Transaction Geoscience and Remote Sensing, 2014, 52(1): 651-663

(上接第 11 页)

有效的结果。我们的原型在同一个移动设备上支持基于 Linux 内核的不同操作系统(Android 与 Firefox OS)的同时执行。由于开源移动操作系统开发的发展，Linux 与 ARM 成为我们实现与研究的基础。但是本文描写的技术与方法亦可以在其他 OS 与硬件平台(甚至 x86)实现。

下一步我们要对 ARM-MuxOS 进行几个方面的改进：一是随着多核移动设备的发展，如何将一个处理器核动态分配给后台的系统从而执行简单的任务。这一问题是为了面对并行执行时共分硬件设备所带来的挑战。二是面对支持非 Linux 的 OS 时，如何处理在两个系统之间不同硬件设备的状态。

参 考 文 献

[1] Labs B. CodeZero hypervisor [CP/OL]. http://dev.b-labs.com/

[2] 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. 2011: 173-187

[3] Barr K, Bungale P, Deasy S, et al. The VMware Mobile Virtualization Platform: is that a hypervisor in your pocket? [J]. ACM SIGOPS Operating Systems Review, 2010, 44(4): 124-135

[4] Dall C, Nieh J. KVM for ARM[C]// Proceedings of the Linux Symposium. 2010: 47-56

[5] Xu Y, Bruns F, Gonzalez E, et al. Performance evaluation of Para-virtualization on modern mobile phone platform[J]. International Conference on Computer, Electrical, and Systems Sciences, and Engineering, 2010(48): 272-279

[6] Heiser G. Virtualizing embedded systems-why bother? [C]// 48th Design Automation Conference. 2011: 901-905

[7] Heiser G, Leslie B. The OKL4 microvisor: convergence point of microkernels and hypervisors[C]// Proceedings of the 1st ACM Asia-Pacific Workshop on Systems. 2010: 19-24

[8] Sun J, Zhou D, Longerbeam S. Supporting multiple OSes with OS switching[C]// Proceedings of the USENIX Annual Technical Conference. 2007: 357-362

[9] Keller E, Szefer J, Rexford J, et al. NoHype: Virtualized cloud infrastructure without the virtualization[C]// 37th Annual International Symposium on Computer Architecture. 2010: 350-361

[10] Nomura Y, Senzaki R, Nakahara D, et al. Mint: Booting multiple Linux kernels on a multicore processor[C]// Broadband and Wireless Computing, Communication and Applications (BWCCA). 2011: 555-560