

MobiVMM: a Virtual Machine Monitor for Mobile Phones

Seehwan Yoo
Korea University, South Korea
shyoo@os.korea.ac.kr

Yunxin Liu
Shanghai Jiaotong University
Microsoft Research Asia,
China
Yunxin.Liu@microsoft.com

Cheol-Ho Hong
Korea University, South Korea
chhong@os.korea.ac.kr

Chuck Yoo
Korea University, South Korea
hxy@os.korea.ac.kr

Yongguang Zhang
Microsoft Research Asia,
China
ygz@microsoft.com

ABSTRACT

Mobile phones have evolved into complex systems as they have more and more new applications built-in. As a result, they are less reliable and less secure than before. Virtual Machine Monitors (VMM) or hypervisors have been introduced to help the reliability and security of mobile phones but the existing research does not completely address three issues critical to mobile phones: real-time support, resource limitation, and power efficiency. In this paper we propose building a new VMM called MobiVMM for mobile phones to deal with these issues. MobiVMM enables real-time support using priority based scheduling and a pseudo-polling mechanism. Resource and power efficiency is achieved through light-weight design and implementation, highly customized guest operating systems, and a virtual hardware abstraction layer. We present our design considerations and report some preliminary experimental results based on the OMAP 2430 development platform.

Categories and Subject Descriptors

D.3.3 [Special-Purpose and Application-Based Systems]: Real-time and embedded systems

General Terms

Management, Performance, Design, Reliability, Experimentation

Keywords

Virtual machine monitor, mobile device, embedded operating systems

1. INTRODUCTION

With advances in hardware and software technologies, mobile phones are becoming more and more powerful. Today,

high-end mobile phones are equipped with CPUs up to 1 GHz, exceeding 100 Mega bytes of RAM, several Giga bytes of storage, and rich peripherals such as cameras, GPS receivers, fingerprint readers, Bluetooth technology, Wi-Fi, and various sensors. Many run a general-purpose OS like Windows Mobile, Linux, or Symbian, and allow user installation of a large set of applications. As mobile phones become increasingly complex, the software becomes less reliable and less secure. The strong interdependencies among OS components (e.g., device drivers) and among applications (e.g., through shared data) are often cited as the root cause for system instability or crashes. Malwares also exploit such dependencies to affect the whole system and hence other applications. In fact, the amount of malware and the number of viruses targeting mobile phones is increasing dramatically [3, 16]. System virtualization, which can provide a new layer of isolation to separate groups of applications or features, are widely regarded as a promising way to address the above reliability and security problems [4, 5, 6, 9, 11].

In this research, we investigate techniques to incorporate a Virtual Machine Monitor (VMM) into the software architecture of mobile phones. We have designed a virtualization layer called MobiVMM (Figure 1), to enable some useful user scenarios. Firstly, we observe that usage models of a mobile phone are rather different from that of desktops or laptops. For example, people are more attached to them in constant use. They are often used for multiple purposes and have vastly different applications running, as people tend not to carry multiple devices. Through MobiVMM, we can categorize mobile applications into several groups and isolate the groups using different VMs. For example, a phone can have three VMs: one real-time VM for basic phone functions like making a call and sending short messages; one that runs critical applications such as a digital payment or online banking; and one "best-effort" VM that adopts general applications including mobile games, a media player or a web browser. Thus, MobiVMM provides a convenient way for application integration and isolation.

Secondly, MobiVMM offers greater portability of data and applications than other software architectures in mobile phones. By suspending a VM, people are able to easily move their data and applications around. When people replace their old mobile phone, a lot of effort is required to copy old data, install favorite applications, and configure detailed settings. With MobiVMM, any user would be able to carry the entire VM to a new mobile phone and access their data and ap-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MobiVirt 2008 Breckenridge, CO, USA

Copyright 2008 ACM 978-1-60558-328-0/08/06 ...\$5.00.

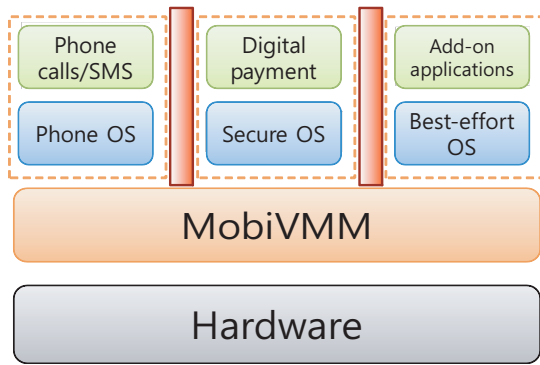


Figure 1: VMM for Mobile Phones

plications continuously. People could also backup VMs to a persistent storage so that if a mobile phone is lost, its data and applications can be restored from the backup storage.

To build a proper VMM for mobile phones, several critical issues must be addressed: real-time support, resource limitation, and power efficiency. In addition to the requirements of realtime voice communication, mobile phone differs from PCs in having unique restrictions such as limited resources in memory size, processing power and energy consumption. However, existing VMMs do not address those issues completely. On the other hand, MobiVMM deals with those issues as its first consideration.

In addition, VMMs for PCs often treat all guest OSs equally and support almost identical services for each. Instead, we are building MobiVMM with the assumption that each guest OS can behave differently and has different requirements. For example, as shown in Figure 1, the phone OS requires real-time support, the secure OS emphasizes security and the best-effort OS just asks for a best-effort service. MobiVMM treats them differently, i.e. applying different scheduling or security policies to them. In this paper we report our preliminary design and evaluation.

As system virtualization studies for embedded devices like mobile phones are still in their early stages, we believe our work will contribute to the identification of the proper design tradeoffs and choices in building embedded VMMs.

The rest of the paper is organized as follows. We discuss the related work in part 2 and present how to do ARM virtualization in part 3. We describe our design of real-time support in part 4 and demonstrate how to improve resource and power efficiency in part 5. In part 6, we report our preliminary results and conclude the paper in part 7.

2. RELATED WORK

System virtualization has a long history since its first use within IBM VM/370 in the 1960s [8]. Because traditional x86-based IA-32 architecture was not virtualizable according to a previous research [15], many technologies such as binary translation [7] have been extensively studied to execute a VMM for commodity PC hardware. These technologies implement a full virtualization of the underlying hardware and ensure 100% binary compatibility at the expense of performance. In contrast, paravirtualization technologies achieve much higher performance in throughput but require modifications over the source code of existing OSes. Today VMM has been actively used in PC and large-scale servers

for various applications [1, 13, 20].

Only recently it became feasible for embedded devices like mobile phones to run a VMM, and researchers began to consider building embedded VMMs. The recent research paper [5] talked about the opportunities and challenges for pocket size hypervisors. The role and limitations of virtualization in embedded systems were discussed in another research [9]. In a paper [4], the authors proposed to use virtualization to address the security issues in mobile devices.

The efforts to port Xen to ARM processors are drawing many attentions. A paper [6] presents porting efforts of early version of Xen to StrongARM architecture but only the basic functionality to load a testing OS was supported. Recent paper, Xen on ARM [11] presents issues on ARM CPU and memory virtualization. They ported Xen 3.0.2 to an ARM-based mobile phone platform for secure execution.

However, Xen itself is designed for incorporating hundreds of network servers and not optimized to work well for embedded devices that have some unique characteristics compared with PCs or server systems. Interrupt latency in Xen is not stable; thus it is hard to support the real-time requirement of mobile phones. Xen's I/O ring is optimized for throughput, not latency. Recently study on Xen's I/O architecture and scheduler reveals problems that makes hard for Xen to provide a good real-time property [14]. In another research work [12], the author discussed the timing requirements of VMs in real-time embedded systems and the possible approach to provide real-time support by proper scheduling. The work is still in progress and the author plans to do real implementation based on Xen.

Architecturally, microkernel and virtual machine are similar: both provide a thin common hardware abstraction layer. Recent study [10] deals with architectural similarities and differences between virtual machine and microkernel system. The paper gives three major lessons learned in the software architectural perspective. At first, VMM has a good architectural property for avoiding liability inversion because the VM is a stand-alone, isolated execution unit; thus liability inversion problem can be simply avoided. Microkernel seems to be a suitable approach to support a componentized OS service because it defines a flexible inter-communication interface. In common for both, the communication should be light weight enough; otherwise, the performance problem should be considerable. There are also additional studies based on L4 microkernel [2, 18]. Recently L4Linux runs over mobile phone products. However, it supports only limited operating systems because specialized abstraction layers of different native OSs often don't match well with the L4 primitives. For example, DirectX or Quartz in Mac would require amount of effort to be executed over L4.

Trango [17] and VirtualLogix [19] are two commercial VMM implementations for ARM based processors but we are not aware of any technical details on them.

3. ARM ARCHITECTURE VIRTUALIZATION

System virtualization is tightly coupled with the underlying hardware architecture. Unlike PCs where x86 is the dominate architecture, embedded devices including mobile phones are often based on different architectures. We choose ARM as our target platform because it is the most popular architecture for embedded devices. As described below, by

design ARM is not "virtualizable" and proper technologies must be developed to virtualize ARM CPU and memory. Some technologies we used like using ARM domains for virtual modes are similar to the ones presented in Xen on ARM [5].

3.1 CPU Virtualization

ARM architecture is not virtualizable according to the criteria [9] because it has some unprivileged sensitive instructions which silently fail when executed in user mode. Although ARM has 7 execution modes but it has only two privilege levels and the user mode is the only unprivileged mode. As MobiVMM must run in privileged mode to ensure that the whole system is fully under its control, a guest OS has to be deprived to run in the user mode together with its applications. It is challenging to protect the guest OS from its applications.

Every unprivileged sensitive instruction like MSR/MRS instructions (which modify the sensitive Current Program Status Register and may result in nondeterministic system states when silently failed in user mode) are replaced by hypercalls which switch the execution into MobiVMM and validate the operation. MobiVMM maintains shadow copies of the virtual CPU context for each guest and makes them transparent to all the guests.

As illustrated in the memory virtualization part, to protect a guest OS from its application, two virtual execution modes are emulated using ARM's domain protection mechanism: application mode and guest OS mode which both are physically mapped to ARM user mode. MobiVMM controls the transitions among the two virtual modes and other modes through exceptions, hypercalls, and upcalls.

3.2 Memory Virtualization

MobiVMM shares the same address space with guest OSs, and the highest 32MB of the 4GB address space is reserved for MobiVMM. Consequently, we don't need to flush the TLB and cache when switch into or out of MobiVMM, resulting in small overhead.

The memory region of MobiVMM is protected using ARM paging mechanism and cannot be access in user mode. However, as both a guest OS and its applications run in user mode, paging mechanism cannot be used to protect a guest OS from its applications. Instead, we use ARM domain protection mechanism to protect the memory of a guest OS. A domain is a collection of memory sections and pages. ARM supports 16 domains in total and access to each domain is controlled by the Domain Access Control Register (DACR). We use three different domains for MobiVMM, guest OS, and applications. We set the access permission of the domain of a guest OS to "no access" in the virtual application mode. Therefore, no application can corrupt the memory of the guest OS which it belongs to. This approach is the same to the one used in [5] where more details are presented.

4. REAL-TIME SUPPORT

Embedded devices like mobile phones usually have requirements dependent on real-time. Although today mobile phones are multiple purposes devices, their primary role is still voice communication, and smooth phone calls must be a primary concern. Therefore, real-time has to be seriously considered in a VMM designed for mobile phones.

To support real-time in VMM architecture, the VMM must be carefully designed to provide smart scheduling and low-latency I/O processing. If the scheduler fails to grant the virtual processor to a guest OS at proper time, the task cannot be finished in given deadline. As many VMMs often convert a physical interrupt into an event and deliver it to destination VM in asynchronously, interrupt processing cannot be finished in predictable time limit. For a real-time system, predictability is one of the major concerns because the RT scheduler should always be able to finish tasks in a given deadline. Although a light-weight event delivery mechanism provides a good performance, as presented in Xen, asynchrony causes unpredictability in latency.

The-state-of-the-art VMM implementations including Xen on ARM are not explicitly designed to support real-time. Especially, because Xen's default credit scheduler is not a preemptive one, a real-time VM has to wait if time credit is exceeded. In addition, Xen redirects all interrupts to a special domain, Dom0, before they are delivered to the target VM. This introduces an amount of interrupt latency when the target VM is not scheduled immediately.

We treat the real-time property as the primary consideration in MobiVMM design. MobiVMM uses a preemptive scheduler so that a real-time VM is able to get the processor time whenever it wants. We also provide low-latency I/O processing which cooperates with the scheduler to timely deliver interrupts and events to a real-time VM.

Real-time scheduling has been extensively studied in previous work and it is hard to support real-time for all the VMs. We simplify the problem by providing real-time support for only one VM at a time, which makes sense for mobile phones because usually only voice communication requires real-time. Our scheduler gives the highest priority to the real-time VM and always schedules it to run unless it yields the processor. The real-time VM may also mark it as "non-real-time" to give more processing time to other VMs. This is useful because a VM might not always require real-time. For example, real-time is required for a phone call but not for idle or dormant period.

Many real-time systems often use polling instead of interrupt because polling has a more predictable behavior than interrupt. By nature, interrupt occurs at random time, and it will stop the current processing task and force the system to handle the external event at first hand. Therefore, it is hard to guarantee the real-time property when interrupt is extensively used because the property can be broken when an interrupt occurs during the time critical task is running. In contrast, a user is able to control the timing of handling events with polling I/O so that a time critical task cannot be preempted at an undesirable time.

We use pseudo-polling to provide low-latency I/O processing and to support real-time better. Pseudo-polling I/O is a translation mechanism of an interrupt into polling. Physical interrupt is stored temporarily inside the virtual machine monitor and delivers the event in explicit time under the control of the guest OS. To avoid preemption at the critical time, interrupts are enabled only when there is enough processor time for interrupt processing. Stored interrupt processing is delayed until the target guest OS is scheduled. When a real-time guest OS gets control, it performs I/O for devices for which interrupts are stored. The scheduler in a real-time guest OS checks whether it conflicts with the real-time schedulability test.

We are also considering whether nested interrupts are allowed. For nested interrupt processing, VMM does real-time schedulability test on the fly. According to the schedulability test, interrupts are discarded or granted to be processed. Thus, we can change unpredictable interrupt processing time into a bounded processing time.

Note that because the interrupt line is masked out when the processor time is not enough to process interrupts, other virtual machines would miss the interrupt. However, when it is enabled again, the guest OS gets pending interrupts. This gives a predictable I/O processing time because the guest OS may perform I/O without disrupting time critical real-time applications.

5. RESOURCE AND POWER EFFICIENCY

Although recent mobile phones have much more powerful hardware than before, their system resources like processing power and memory size are still limited, compared to commodity PCs. Especially, energy consumption is a hard limitation for battery-powered devices like mobile phones. One major reason that manufactures don't ship mobile phones with higher frequency CPU and more memory is due to the power limitation. Therefore, a VMM for mobile phones must be as resource and power efficient as possible.

Xen is optimized for high performance, not for small code base. As mentioned in [11], the size of Xen and its tools are too big to fit into a mobile phone's flash memory. Xen also doesn't consider power efficiency much as it was designed for large-scale network server consolidation.

We design MobiVMM with resource and power efficiency in mind. MobiVMM is light-weight with minimal memory footprint. MobiVMM will suspend or shutdown those VMs which have been idle for certain amount of time. MobiVMM monitors the battery condition and will prompt the user to shutdown some functionalities in low battery, so that the core services keep running. Profiles are supported thus users are able to configure that low priority functions may be automatically shut down when battery is too low.

We do not assume that each guest OS is for general purpose. Instead, we believe that it is a promising way to reduce resource usage by using highly customized guest OSes. As each guest OS runs only a few applications, they can be compiled with only necessary supporting components for their applications built-in. OSes in different VMs are built with different sizes. For example, Windows Mobile may have a size from several MB to tens of MB depending on how many components are included. Since current mobile phones tend to have tens of GB storage but quite limited memory size. With MobiVMM, we are able to store many highly customized guest OSes in a mobile phone and launch them on demand, which helps reduce resource usage and improve power efficiency. And a highly customized guest OS also has better performance.

In addition, embedded systems typically have a hardware abstract layer (HAL) to deal with diverse hardware details. For example, Windows Mobile has the OEM Adaptation Layer (OAL) to host the target board-specific functions. MobiVMM leverages this by providing a well defined virtual HAL interface to the guest OSes so that the guest OSes can be further simplified.

6. PRELIMINARY RESULTS

Table 1: Average response time for a 1ms sleeping call

	VM0 alone	2VMs One App	2VMs Apps in one VM	2VMs Apps in different VMs
Avg.	0.855	0.798	0.872	1.030
Stdev.	0.453	0.419	0.407	1.349

We are actively implementing MobiVMM based on OMAP 2430 mobile phone development platform which has an ARM 1136 core at 330 MHz, 64MB NOR flash memory, and 64MB NAND flash memory. We have implemented the basic functions of MobiVMM and are able to run two guest OSes side by side. Currently I/O processing is performed in a dedicated guest OS and MobiVMM implements a simple time-sharing scheduler between two VMs. We use paravirtualized Linux 2.6.21.

We measured context switching time between guest OSes. In normal case, guest domain switching requires less than 32 microseconds (μs) including the ARM specific exception mode traverse. During the context switch MobiVMM visits abort, undef, irq, and svc modes to save banked registers in guest VMs.

We measured response time at one virtual machine while playing MP3 music in another virtual machine. Response time measurement process keeps running in the loop, and the loop repeats sleeping the process during 1ms. We use `nanosleep()` function to sleep for a millisecond. Actual sleeping time is measured by the time tick. Each time tick is set to one microsecond. Thus, microsecond and ticks are convertibly used. Table 1 presents averages and standard deviations of sleeping time in four cases. Four scenarios are as followings: 1) one VM runs response time measurement app, 2) two VMs, but only one app runs, 3) two VMs, mp3 playback and response time runs on the same VM, 4) one VM runs mp3 playback, the other measures.

Experiment result shows that it has a very small overhead in small-number of VM environment. Because of the `nanosleep()` implementation, the measurement process would not change state to a sleep state. This causes very short latency. However, we can still observe a fluctuation in response time. The standard deviation of 300 runs is considerable; it is about 1.5 milliseconds, which is almost three times larger than the other cases.

The performance result can be affected by the timer granularity. The default timer granularity is 10ms, and each virtual machine gets a time quantum of 100ms. That is, each VM can get up to ten timer ticks without preemption. Therefore, response time is very large. We can improve it by adopting more responsive scheduling policy. Current VM implementation gives a new time quantum to the VM which has an I/O processing. Thus, the guest VM execute mp3 playback with the smallest response time. We can improve it by adopting a smaller time quantum.

When a high-resolution time quantum is adopted, context switching overhead is increased at the same time. We changed the time quantum from 100ms to 10ms, and each virtual machine gets a timer interrupt every 20ms. Table 2 shows the similar results from the previous experiments. Through the result, we can figure out the overall response time is increased.

Table 2: Response time for a smaller time quantum

	VM0 alone	2VMs One App	2VMs Apps in one VM	2VMs Apps in different VMs
Avg.	1.084	1.100	0.946	1.12
Stddev.	1.084	1.100	1.124	1.187

7. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a VMM called MobiVMM for mobile phones to deal with issues which are not fully explained in previous research: real-time support, resource limitation, and power efficiency. We believe that these issues are critical for mobile phones and they are the primary considerations in MobiVMM. Real-time property is achieved by priority-based preemptive scheduling and pseudo-polling mechanism. The light-weight VM design and implementation, highly customized guest OSES, and virtual HAL interface lead to good resource and power efficiency. We also report our preliminary results.

Our work is in progress and basic functions of MobiVMM have been implemented. We will continue the development of MobiVMM and perform comprehensive evaluations. In addition, currently we support only Linux as a guest OS, but we plan to port other OSs like Windows Mobile onto MobiVMM. Because different mobile phones have very diverse hardware configurations, we believe that it is important to have a well-defined Virtual Machine Interface (VMI). We will define such a VMI to reduce the porting effort of a guest OS onto MobiVMM.

8. REFERENCES

- [1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03)*, pages 164–177, New York, NY, USA, 2003. ACM.
- [2] C. v. S. Ben Leslie and G. Heiser. Wombat: a portable user-mode linux for embedded systems. In *6th Linux.Conf.Au, Canberra*, Apr. 2005.
- [3] A. Bose and K. G. Shin. On mobile viruses exploiting messaging and bluetooth services. *Securecomm and Workshops, 2006*, pages 1–10, 28 2006-Sept. 1 2006.
- [4] J. Brakensiek, A. Droge, H. Hartig, A. Lackorzynski, and M. Botteck. Virtualization as an enabler for security in mobile devices. *Workshop on Isolation and Integration in Embedded Systems, IIES, 2008. Glasgow, UK. 1st ACM SIGOPS*, Apr. 2008.
- [5] L. Cox and P. Chen. Pocket hypervisors: Opportunities and challenges. *Mobile Computing Systems and Applications, 2007. HotMobile 2007. Eighth IEEE Workshop on*, pages 46–50, March 2007.
- [6] D. R. Ferstay. Fast secure virtualization for the arm platform. Master's thesis, University of British Columbia, 2006.
- [7] M. Gschwind, E. R. Altman, S. Sathaye, P. Ledak, and D. Appenzeller. Dynamic and transparent binary translation. *Computer*, 33(3):54–59, 2000.
- [8] P. Gum. System/370 extended architecture: Facilities for virtual machines. *IBM Journal of Research and Development*, 27(6):530–544, Nov. 1983.
- [9] G. Heiser. The role of virtualization in embedded systems. *Workshop on Isolation and Integration in Embedded Systems, IIES, 2008. Glasgow, UK. 1st ACM SIGOPS*, Apr. 2008.
- [10] G. Heiser, V. Uhlig, and J. LeVasseur. Are virtual-machine monitors microkernels done right? *SIGOPS Operating Systems Review*, 40(1):95–99, 2006.
- [11] J.-Y. Hwang, S.-B. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim. Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones. *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 257–261, Jan. 2008.
- [12] R. Kaiser. Alternatives for scheduling virtual machines in real-time embedded systems. In *IIES08 - 1st EuroSys 2008 ACM SIGOPS Workshop on Isolation and Integration*, Apr. 2008.
- [13] Microsoft. *Microsoft Virtual PC*. <http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx>.
- [14] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling i/o in virtual machine monitors. In *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 1–10, New York, NY, USA, 2008. ACM.
- [15] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.
- [16] Tower Group. *Cyber-criminals target mobile banking, 2007*. <http://www.vnunet.com/vnunet/news/2173161/cyber-criminals-targetmobile>.
- [17] Trango. *Trango: secured virtualization on ARM*. <http://www.trango-vp.com>.
- [18] C. van Schaik and G. Heiser. High-performance microkernels and virtualisation on arm and segmented architectures. In *Workshop on Microkernels for Embedded Systems, Sydney, Australia*, Jan. 2007.
- [19] VirtualLogix. *VirtualLogix Real-Time Virtualization and VLX*. <http://www.osware.com>.
- [20] VMWare. *VMware ESX Server*. <http://www.vmware.com/products/vi/esx>.