Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones

Joo-Young Hwang, Sang-Bum Suh, Sung-Kwan Heo, Chan-Ju Park,
Jae-Min Ryu, Seong-Yeol Park, Chul-Ryun Kim
Software Laboratories, Corporate Technology Operations, Samsung Electronics Co. Ltd.
416, Maetan-3Dong, Yeongtong-Gu, Suwon-City, Gyeonggi-Do, Korea, 443-742
E-mail: {jooyoung.hwang, sbuk.suh, sk.heo, bestworld}@samsung.com
{jm77.ryu, seongyeol.park, cr2104.kim}@samsung.com

***Abstract*—Mobile phones security is becoming an important issue because they are being connected to an Internet through wireless modem technologies. System virtualization technology provides trusted computing capability by running isolated multiple virtual machines under hypervisor. In this paper, we propose a design of system virtualization for ARM CPU architecture and describe implementation of prototype called Xen on Arm using Xen hypervisor. Secure and nonsecure guest Linux virtual machines are executing under Xen on ARM isolated with each other and virtualization overhead is shown to be moderate compared to native Linux running on bare metal H/W.**

I. Introduction

Recently, beyond-3G mobile phones will be connected to Internet through packet-switched networks such as mobile WiMax and WiBro technologies. Mobile phones will face the similar security problems with malwares as shown in personal computer(PC) environments. Operating system (OS) level security solutions can be compromised if OS is compromised.

System virtualization technology has been applied to servers and workstations to helps system users be able to consolidate the hardware and provides flexibility. System virtualization means creating virtual machines by virtualizing a full set of hardware resources, including a processor, memory, storage resources and peripheral devices. A virtual machine monitor (VMM) or hypervisor is the software that provides abstraction of virtual machine to guest OSes which are running under the VMM. The virtual machine is a duplicated real machine, and VMM takes complete control of virtualized resources.

The security issue can be handled by system virtualization since it provides isolation of guest OSes running under VMM, so that any compromised guest OS cannot be propagated to other guest OS domains. This can guarantee availability of a computing system even when a guest OS domain fails.

To investigate virtualization technology for embedded, we choose ARM architecture since it is popularly used for mobile phones and choose Xen which is a popular open source hypervisor. The authors of this paper had reported on "Xen on ARM" which is Xen for ARM architecture including VMM level access control scheme in [1]. In this paper, we focus on ARM virtualization technology and its implementation issues.

This paper is organized as following. In section II, we describe virtualization technology backgrounds, implications on virtualization for ARM architecture. In section III related works are mentioned. In section IV and V, ARM CPU and memory virtualization are presented, respectively. Then we describe implementation issues and show performance results in section VI. Finally we conclude in section VII.

II. Backgrounds

A. Virtualization History

The notion of virtual machine dates back to the mid 1960's IBM M44/44X[2] and IBM CP-40, and has been used for various application areas such as server consolidation, disaster recovery, and testing of OS kernel. Rigorous survey of virtualization can be found in [3].

In [4], the authors described sufficient condition for a computer architecture to allow creation of VMM: A VMM may be constructed if the set of sensitive instructions[1] for that computer is a subset of the set of privileged instructions[2] and processor has at least two privilege modes (privileged mode and non-privileged mode). The theorem can be naturally extended to current machines.

Though several modern processors are not virtualizable in the author's sense because it has sensitive unprivileged instructions, VMMs have been built successfully by handling sensitive unprivileged instructions in various ways. Dynamic recompilation technique is used to discover those instructions at run-time and replace them with a trap into the VMM[5], which is known as *full virtualization* technology and does not require OS source code modifications. In contrast, *paravirtualization* approach modifies guest OS source code to replace those instructions with invocations of hypervisor routines (aka "hypercall"). Though paravirtualization requires OS modification, this is not critical issue because required modification is slight.

[1]Sensitive instructions are those that should be run in privileged mode to ensure correct function of them. Sensitive instructions can be control sensitive or behavior sensitive. Control sensitive instructions are those that attempt to change the configuration of resources in the system. Behavior sensitive instructions are those whose behavior or result depends on the configuration of resources.

[2]Privileged instructions are those that trap if the processor is in user mode and do not trap if it is in system mode.

1-4244-1457-1/08/$25.00 © IEEE

B. Issues in Virtualization for Embedded Devices

Embedded devices have scarce computing power compared to servers and desktops. Efficiency is a major concern in embedded virtualization. Paravirtualization approach is more efficient than full virtualization because expensive binary translation is not necessary, so we decide to use paravirtualization approach.

Among several open source paravirtualization solutions such as Xen [6] and L4, we chose Xen as the beginning point of our research because the interface of Xen is rather simple yet similar to machine interfaces than that of L4 which was originally designed as an operating system. It has implementation of architecture independent common components of VMM such as hypercall interface, VM scheduling, VM controls, and memory management.

Currently Xen is supporting x86-64, IA-32, IA-64, and PowerPC. However it cannot be directly ported to ARM CPU because ARM's virtualization capability is rather poor than x86 and PowerPC which have rich functions designed for desktops and servers. Compared to x86's virtualization capability [7], ARM CPU has the following problem in virtualization. ARM CPU has only one unprivileged mode.[3] Since guest OS should not run in privileged mode for VMM to take exclusive control of hardware resources, guest OS and applications should run together at user mode. However, conventional MMU based paging mechanism cannot protect the OS kernel from applications when they are running in the same user mode.

Simply separating the address spaces of applications and OS kernel will lead to significant cache/TLB flushing overheads since ARM v4/v5 architecture has virtually indexed virtually tagged (VIVT) cache, and Translation Look-aside Buffer (TLB) entries are not tagged with address space ID[4], so cache and TLB should be flushed when switching address spaces.

It is non-trivial task to protect OS kernel from applications and to protect VMM from OS kernel and applications efficiently with the difficulties in ARM CPU such as scarce CPU privilege modes and high address space switching costs.

III. Related Works

In [8], the author indicated the two implications mentioned above but did not address them fully. The author mentioned separating page tables of OS kernel and user processes but didn't implement it and no experimental results were reported. This scheme is emulated in our experiments to compare regarding context switch latencies as shown in Fig. 3.

In L4 microkernel[9], Linux server runs as separate L4 task side by side with Linux user processes under L4 microkernel. The memory protection is ensured by page table separation. In [10], they implemented single address space mechanism, where address spaces of all L4 tasks including Linux servers and Linux processes are not overlapping. Cache flush is not required for virtual machine switching or process switching. ARM's FCSE (Fast Context Switch Extension)[11][12] is exploited to relocate address space of a process at run-time. However, the size of address space of each process is limited to 32MB, which is not desirable for open platform mobile phones which provides connectivity to Internet, downloads and executes legacy applications.[5] Additionally, since most of operating systems do not support single address space scheme, significant engineering effort is necessary to modify new guest OS. However, the fast address-space switching (FASS) is a useful optimization for particular embedded applications on ARM. It can be our future work to employ FASS in Xen on ARM for further performance optimization.

There are a few commercial VMMs for ARM; Trango and VirtualLogix. They demonstrate running real-time OS and high level OS like Linux side-by-side. Trango system hypervisor is only 20KB which can be burn in ROM on CPU. However their source codes are closed, so detailed architecture comparison is not possible.

IV. CPU Virtualization

A. Virtual Privilege Modes

In order to deprivilege guest OS and allow full resource control to VMM, only VMM runs in the supervisor mode and guest OS runs in user mode together with applications. Compared to x86 architecture which has four privilege rings, it is more difficult to protect guest OS kernel from user applications because they have to run in the same mode. We describe isolation of kernel's CPU contexts from applications in this section and protection of kernel memory from user applications is detailed in section V.

ARM Linux kernel is implemented assuming that it runs at supervisor mode and all register contexts are supervisor mode context. To minimize Linux modifications, we decided to provide an abstract supervisor mode to guest OS kernel. User mode is split into two logical modes (user process mode and kernel mode), and virtual banked registers for those virtual privilege modes. Xen on ARM is in charge of switching between the user process mode and the kernel modes.

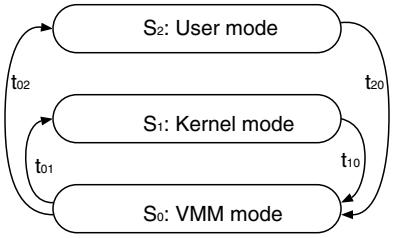

Fig. 1. VCPU mode transitions. t_{10}: on interrupts/faults/aborts/hypercalls, t_{20}: on interrupts/faults/aborts/system calls, t_{01}: upcall or return from exception, t_{02}: return from exception

[3]ARM has seven basic operating modes, six privileged modes (FIQ, IRQ, Supervisor, Abort, Undef, System) and one unprivileged mode(User).

[4]ARM11 has virtually indexed physically tagged cache (VIPT) and MPcore has physically indexed physically tagged (PIPT) cache.

[5]Microsoft Windows CE has supported single address space by using FCSE but recent Windows CE 6.0 does not support the limited model.

B. Exception Handling

Virtual CPU is modeled by a state machine as illustrated in Fig. 1 where each state corresponds to a virtual privilege mode. VMM mode is entered when exceptions such as interrupt, fault, abort, and software interrupt occurs. On entry to VMM mode, ARM CPU's SPSR (Saved Program Status Register) is saved in the VSPSR (virtual SPSR) which will be restored later when guest OS returns from exception. Stack pointer register is also saved in VSP register for later restoration.

Xen invokes *upcall* to deliver exceptions to kernel mode as virtual exception events. On upcall, Xen puts the VSPSR information on the kernel's stack so that kernel can get last running virtual processor mode. The upcall mechanism corresponds to hardware level exception handling that makes CPU to jump into exception vector table.

Since the guest OS cannot access sensitive registers such as ARM's FAR (Fault Address Register)[6] and FSR (Fault Status Register)[7], Xen on ARM provides virtual FAR and virtual FSR to guest OS. Guest OS invokes hypercall to return from exception and Xen restores the saved context. Exception handling procedure is described in Fig. 2.

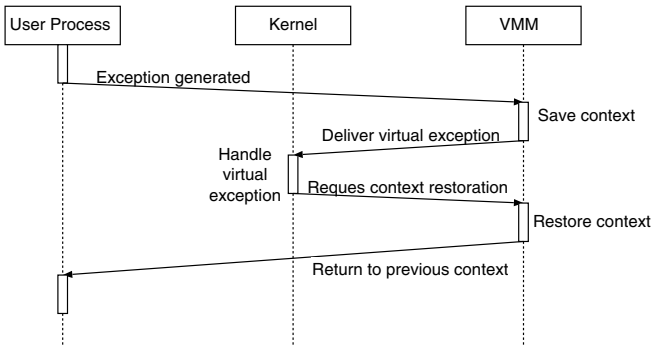

Fig. 2. Exception handling in virtualization environment

C. Sensitive Instructions

Since guest OS should run in user mode, all sensitive instructions contained in guest OS should be replaced with proper hypercalls. ARM v5 architecture has 14 sensitve instructions [13], [14]. Sensitive unprivileged instructions[8] silently fails when they are executed in user mode. For example, many Linux kernel operations like semaphore use MSR/MRS instructions that try to modify the status flag bits of CPSR (Current Program Status Register). Those instructions fail without any error report, and may lead to system malfunction in nondeterministic ways. Therefore, guest OS code should be scanned to find the sensitive unprivileged instructions and replace them with hypercalls.

[6]FAR contains the virtual address of an attempted access which caused the exception.

[7]FSR contains source of fault and protection domain which caused the fault.

[8]MRS/MSR/MOVS/STM/LDM/CMPP/CMNP/TEQP/TSTP

V. Memory Virtualization

The following (1) - (3) are the requirements to protect memory area between virtual privilege modes inside a virtual machine: (1) VMM memory region should be protected from guest OS kernel and user processes, (2) guest OS kernel memory should be protected from user processes, (3) User process memory should be protected from other processes. Additionally every virtual machine should be isolated each other.

Xen on ARM ensures isolation between guest domains as following. The guest domain's memory mapping is created and updated only by Xen. To modify memory mapping, guest OS should invoke hypercall to update page table. Xen blocks any attempt of guest OS to map physical memory area that is not granted to the guest OS.

With paging mechanism we can protect Xen memory from guest OS/user processes, however, cannot protect OS kernel memory from user processes because paging mechanism doesn't know whether user processes or OS kernel is running. We need another mechanism to protect OS kernel memory from user processes.

We exploit ARM processor's domain protection mechanism[11]. Among 16 domains (D0 - D15), we use three (D0 - D2) for virtualization purpose. Memory areas of VMM mode, kernel mode, and user process mode belong to D0, D1, and D2, respectively. Access permission to a particular domain is configured by setting corresponding bits of Domain Access Control Register(DACR)[9] Access permission to memory which belongs to a domain is one of three types; *no access* (unconditionally inaccessible), *client*(conditional, access permission check is to be performed based on the page table entry's AP flag setting), and *manager*(unconditionally accessible).

At the user process mode, access permissions to kernel memory area (D1) and VMM area are set to *no access*. At the kernel mode, access permissions to D0, D1, and D2 are all set to *client*. OS kernel is not allowed to access Xen memory area because Xen memory area's page table settings does not allow access from user mode. At the VMM mode, access permissions to all domains are set to *client*.

A. ARM Virtual Cache Optimizations

The x86 cache system is physically indexed and physically tagged. In contrast, ARM v5 processor has virtually indexed and virtually tagged cache[11], so cache may have multiple items that are mapped to the same physical memory location. For cache consistency, cache should be flushed when switching address spaces. Since cache flush takes significant time, cache flushing should be avoided as much as possible.

Xen hypervisor does not require cache flushing on switching between virtual priviledge modes inside the same virtual machine. The page table of a process has non-overlapping page mappings for the process memory, OS kernel memory,

[9]This is a 32 bit register consisting of 16 2-bit fields which indicates access permissions to memory area of a domain.

TABLE I
LMBENCH RESULTS. RATIO = PARAVIRT VALUE / NATIVE VALUE

Bandwidth measured in MB/sec			
Tests	Native	Paravirt	Ratio
bw_pipe	42.79	38.21	0.893
bw_unix	44.52	39.18	0.880
bw_mem 512 rd	1033.42	1075.79	1.041
bw_mem 512 wr	1034.67	1019.27	0.985
bw_mem 512 rdwr	1034.67	1019.27	0.985
Latencies measured in microseconds			
Tests	Native	Paravirt	Ratio
lat_pipe	135.13	234.42	1.735
fork+exit	2891.75	10021.0	3.465
fork+execve	3109.25	10524	3.385
SysV semaphore	45.974	81.42	1.77
lat_unix	251.41	431.85	1.70
signal handler	11.23	20.43	1.82
null syscall	1.13	2.83	2.50
read syscall	2.60	4.94	1.90
write syscall	2.25	4.16	1.85

and VMM memory. VMM address space is located at the uppermost 64MB address space (0xFC000000 - 0xFFFFFFFF). Linux kernel address space start from 0xC0000000 to 0xFBFFFFFF. Process memory address space is from 0x0 to 0xBFFFFFFF. Entry/exit to/from VMM does not require address space switch, so cache/TLB flush is not necessary. Caches flushing is required only when VMM switches domain or guest OS switches process.

We reduce TLB flushing overheads incurred while switching domains and processes. ARM provides eight lockdown TLB entries which are not invalidated though TLB flush occurs. We use two lockdown TLB entries to hold mappings for Xen memory area. This reduces VMM overheads slightly by keeping TLB entries always.

B. Inter-Domain Memory Isolation

Since paravirtualized guest OSes should run in user mode, they are not allowed to manipulate MMU but instead they can invoke hypercalls to control MMU, which is then validated by Xen on ARM. Any attempt to map/unmap/read/write other domain's memory page is prohibited by VMM. Though a domain is compromised by a malicious software, it cannot attack other domain as far as VMM is not compromised. Other research colleagues independently worked on making Xen on ARM secure [15].

VI. EXPERIMENTAL RESULTS

We modified Xen 3.0.2 to run on a smart phone development platform which is equipped with a 266MHz ARM926-EJS core, 64MB DRAM, and 32MB NOR flash. Linux 2.6 is paravirtualized to run under Xen on ARM. Two virtual machines(dom0 and domU) of paravirtualized Linux 2.6.11 are running on the VMM. Total modified/appended source codes of Xen is 23401 lines and Linux modifications are 4518 lines.

Conventional Xen on x86 has many python scripts for XenBus utilities. Python interpreter cross-compiled for ARM is about 40MB size which is too heavy to fit into flash memory devices. Among 280 modules, we removed unnecessary modules, and reduced interpreter has only 40 modules and its size is 5.7MB. Since XenBus and XenStore are working on the ARM platform, conventional Xen tools can run without source code modifications.

A. Micro Benchmarks

LMBENCH is used to investigate performance of basic system operations. Paravirtualized Linux is compared with native Linux running on bare metal H/W without VMM. Direct performance comparison with optimized wombat on L4 microkernel[10] is not fair because the performance difference is due to FASS scheme which is not related to virtualization. If given native Linux with FASS support, paravirtualization of the Linux version will enhance overall performance. The benefit of FASS scheme will not be distinct for ARM11 core architecture because it has physically tagged cache. System call latency is better than optimized wombat ($\{$virtual Linux's syscall latency in usec$\}$ / $\{$native Linux's syscall latency in usecs$\}$ = $\frac{2.83}{1.13}$ (Xen on ARM) $< \frac{4.0}{0.82}$ (optimized L4)).

LMBENCH results are summarized in Table VI. Latencies of most operating system services are not higher than twice of native Linux performance. Process creation takes longer times than native Linux because a large number of page table updates occur when creating a new process and hypercall is invoked for every page table entry update. The number of hypercalls can be reduced by exploiting multi-call mechanism(multiple hypercalls are batched as a single hypercall) but not implemented yet. Memory bandwidth is not degraded and the IPC performance using pipe and unix socket show about 10% drop.

Fig. 3 shows context switching latency for four cases; (1) Native: native Linux case, (2) paravirtualized Linux with the TLB lockdown discussed in section V, (3) paravirtualized Linux without the TLB lockdown, and (4) page table separation scheme that means user processes don't have kernel memory mappings, so flushes cache on switching between user process and kernel mode to protect kernel from user processes. Compared to native Linux, paravirtualized Linux context switch time has about 50 microseconds additional overhead. This overhead is appended by virtualization and is moderate compared to separate page table scheme. TLB lockdown optimization shows slight contribution when context size is larger than 8 Kbytes.

B. Scalability Analysis

To analyze the performance impact of the number of concurrent VMs (n), we tested four cases; n=1,2,3, and 4. Xen on ARM uses 2MB, dom0 uses 14MB, and other domains use 16MB. Root file systems are mounted as read-only, then we run LMBENCH simultaneously on all running domains, which is iterated ten times for each n. Average throughput values of domains are aggregated and summarized in Table II, where aggregate throughputs don't degrade much as n increases.

C. Macro Benchmarks

To see the virtualization's performance impact on common operations in mobile phones, we compared User Interface (UI)

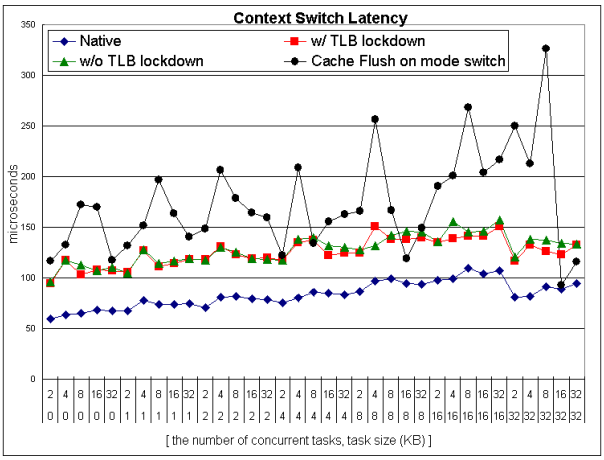

Fig. 3. Context switching latency. X axis numbers in a row denote concurrent number of tasks and context size of a task.

Benchmark	n=1	n=2	n=3	n=4
bw_pipe (MB/sec)	37.28	37.20	36.80	36.84
bw_unix (MB/sec)	32.15	35.31	38.32	35.51

TABLE II
PERFORMANCE IMPACTS OF THE NUMBER OF CONCURRENTLY RUNNING VMS. PIPE AND UNIX SOCKET PERFORMANCES ARE MEASURED FOR n=1,2,3, AND 4.

loading time, image file saving time, and codec performance with native Linux. The results are shown in Table III. Virtualization overheads are not distinct for all those tests. For UI loading test, we used Qtopia PDA Edition and binaries are installed at NOR flash memory. For image file saving test, we prepare 200 files whose size are distributed evenly from 20KB to 90KB. Then we measure the time taken to save all those files from NFS server to NAND flash memory. For codec tests, Xvid MPEG4 stream encoder/decoder are used.

VII. CONCLUSIONS

In this paper, we described design and implementation of Xen on ARM, which is a secure system virtualization of ARM architecture. We described how the open source Xen hypervisor is migrated to ARM architecture and guest OS kernel is protected from applications when guest OS and applications are executing in the user mode. Within our knowledge, Xen on ARM is the first successful implementation of Xen based ARM virtualization which can run multiple isolated high level operating systems. Xen on ARM has shown moderate virtualization overheads. It introduces trusted computing capability to mobile embedded devices by isolating secure OS domain from nonsecure OS domain.

Benchmark	Native	Paravirt	Ratio
UI loading time (seconds)	14.48	14.54	1.004
Image saving time (seconds)	52.71	53.81	1.020
Encoding rate (fps)	5.71	5.63	0.986
Decoding rate (fps)	24.54	24.41	0.995

TABLE III
MACRO BENCHMARKS RESULTS. RATIO = $\frac{Paravirt\ value}{Native\ value}$

REFERENCES

[1] S. B. Suh, "Secure architecture and implementation of xen on arm for mobile devices," in *4th Xen Summit, Yorktown Heights, NY*, 2007. [Online]. Available: http://www.xensource.com/files/xensummit_4/Secure_Xen_ARM_xen-summit-04%_07_Suh.pdf

[2] L. Belady, "A study of replacement algorithms for virtual storage computers," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[3] M. Rosenblom and T. Garfinkel, "Virtual machine monitors: current technology and future trends," *IEEE Computer*, vol. 38, no. 5, pp. 39–47, May 2005.

[4] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[5] C. Waldspurger, "Memory resource management in vmware esx server," 2002. [Online]. Available: citeseer.ist.psu.edu/waldspurger02memory.html

[6] 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 ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[7] J. S. Robin and C. E. Irvine, "Analysis of the intel pentium's ability to support a secure virtual machine monitor," in *Proc. 9th USENIX Security Symposium, Denver, Colorado, USA*, August 2000.

[8] D. R. Ferstay, "Fast secure virtualization for the arm platform," Master's thesis, Department of Computer Science, University of British Columbia, March 2006.

[9] B. Leslie, C. van Schaik, and G. Heiser, "Wombat: a portable user-mode linux for embedded systems," *Proceedings of the 6th Linux.Conf.Au Canberra*, April 2005.

[10] C. van Schaik and G. Heiser, "High-performance microkernels and virtualisation on arm and segmented architectures data objects," in *Proceedings of the 1st International Workshop on Microkernels for Embedded Systems, Sydney, Australia*, 2007.

[11] *ARM926EJ-S Technical Reference Manual*. ARM Limited, 2003.

[12] A. Wiggins, H. Tuch, V. Uhlig, and G. Heiser, "Implementation of fast address-space switching and tlb sharing on the strong arm processor," *Advances in Computer Systems Architecture*, pp. 352–364, 2003.

[13] D. Seal, *ARM Architecture Reference Manual, Second Edition*. Morgan Kaufmann, Addison-Wasley, 2001.

[14] C. Andres N.Sloss, Dominic Symes, *ARM System Developer's Guide*. Morgan Kaufmann, 2004.

[15] S.-M. Lee, S.-B. Suh, B. Jeong, and S.-D. Mo, "A multi-layer mandatory access control mechanism for mobile devices based on virtualization," in *Proc. 4th IEEE Concumer Communication and Networking Conference, Las Vegas, Nevada, USA*, January 2008.