VIRTUS: A New Processor Virtualization Architecture for Security-Oriented Next-Generation Mobile Terminals

INOUE, Hiroaki†, Akihisa Ikeno‡, Masaki Kondo‡, Junji Sakai† and Masato Edahiro†

† System Devices Research Laboratories, NEC Corporation
1120, Shimokuzawa, Sagamihara, Kanagawa, 229-1198 Japan

‡ NEC Informatec Systems, Ltd.
3-2-1, Sakado, Takatsu-ku, Kawasaki, Kanagawa, 213-0012 Japan

h-inoue@ce.jp.nec.com, a-ikeno@pb.jp.nec.com, ms-kondo@pb.jp.nec.com, jsakai@bc.jp.nec.com, eda@bp.jp.nec.com

ABSTRACT

We propose a new processor virtualization architecture, VIRTUS, to provide a dedicated domain for pre-installed applications and virtualized domains for downloaded native applications. With it, security-oriented next-generation mobile terminals can provide any number of domains for native applications. VIRTUS features three new technologies: VMM asymmetrization, dynamic inter-domain communication and virtualization-assist logic, and it is first in the world to virtualize an ARM-based multiprocessor.

Categories and Subject Descriptors

C.1.4 [**Processor Architectures**]: Parallel Architectures – *mobile processors*; D.4.7 [**Operating System**]: Organization and Design – *real-time systems and embedded systems*;

General Terms: Design, Security

Keywords: Multiprocessor, Processor Virtualization

1. INTRODUCTION

In the security architecture specifications [11] [12] jointly announced by NTT DoCoMo, IBM, and Intel for security-oriented next-generation mobile terminals, such features as integrity measurement, tamper-resistant cryptographic processing, access control, and user authentication are provided for enhanced system security. This paper focuses on an especially important security feature among them: creating Operating System (OS) instances (*i.e.*, domains) on which users are able to execute native applications downloaded from open networks.

A domain (an OS instance) is specifically defined as an isolated execution environment prepared for a group of applications. Such domain isolation makes it possible to prevent illegal access to the address spaces of other domains and to limit the maximum amount of resources that applications on the domain may use. In the downloading of native applications to next-generation mobile terminals, it will be useful to be able to protect pre-installed applications on one domain from interference from applications downloaded to other domains, such interference as unintentional access, loss of needed excess resource allocation, etc. Further, this isolation makes it possible to use domains that have been created for different mobile terminals OSs, such as Symbian OS, μ-ITRON, Linux, etc.

While hardware domains (*i.e.*, a multiprocessor) offer high performance for downloaded applications [8], since the applications are directly executed on processors, the fixed number of such domains in a chip limits system flexibility, and an increased number of processors embedded in a chip (*e.g.*, more than four processors) increases system cost.

By way of contrast, software domains, for which Virtual Machine Monitor (VMM) software forms domains by virtualizing an OS or a processor, are highly flexible and cost-effective since VMM software enables a system to create any number of domains without additional processors. Unfortunately, however, the use of VMM software significantly degrades performance because the VMM interferes in the execution of applications (both pre-installed and downloaded) and its frequent scheduling of virtualized domains creates severe overhead.

While neither the hardware nor the software approach is, then, in itself satisfactory, we have managed to create a hybrid of the two which is. This paper reports our design and implementation of a new processor virtualization architecture, VIRTUS, to be used in security-oriented next-generation mobile terminals. VIRTUS forms a hardware domain for pre-installed applications on one processor and software domains for downloaded applications on other processors in order to satisfy the following design objectives:

- **Hardened Security:** Pre-installed applications on next-generation mobile terminals must be protected from downloaded applications. In VIRTUS, the creation of a separate, hardware-level domain for pre-install applications helps to ensure their greater protection.
- **Flexible Processor Virtualization:** Next-generation mobile terminals will require a large number of domains for downloaded applications. The advanced processor virtualization software in VIRTUS is able to provide

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

essentially unlimited number of software domains on virtualized processors.

- **Small Performance Degradation:** Since VIRTUS provides a dedicated processor for the hardware domain, its use creates no degradation in the performance of pre-installed applications. Further, the code-efficient implementation enables VIRTUS to offer small performance degradation, even in native applications downloaded to software domains.
- **Little Performance Interference:** Next-generation mobile terminals will require the suppression of interference on pre-installed applications by downloaded applications. Processor-level separation enables VIRTUS to do this. The domain for pre-installed applications runs on a different processor than those use for the domains for downloaded applications.
- **Small Code Size:** Next-generation mobile terminals will need to be able to operate with limited resources, and the asymmetric design of VIRTUS helps it to operate with a smaller code size.

VIRTUS features three new technologies: 1) VMM asymmetrization, through which the hardware domain and the software domains operate cooperatively on different processors; 2) dynamic inter-domain communication, which enables an application on a running domain to communicate with an application on a dormant domain; 3) virtualization-assist logic for efficient virtualization. These technologies help make VIRTUS ideally suited to security-oriented next-generation mobile terminals. Note that *virtus* is the Latin word known as the root of "virtual".

The remainder of this paper is structured as follows: Section 2 describes related work, Section 3 explains the VIRTUS architecture in detail, Section 4 shows the results of our evaluation of VIRTUS, and, finally, Section 5 summarizes the paper.

2. RELATED WORK

Our research differs in a number of respects from the current body of research on virtual machine technologies. Thus, VIRTUS is designed to satisfy the above five design objectives to enable mobile terminals to support the downloading of native applications.

The major user-level VMMs, known as type-II, include Java Virtual Machine (JVM) [7], which emulates instructions (*i.e.,* non-native applications); User Mode Linux (UML) [5], which emulates OS system calls (*i.e.*, a port of Linux to run as a Linux process); and VMware [15], which emulates sensitive processor-instructions. These VMMs are user-level software and can create any number of domains. The emulations required to create domains, however, results in severe performance degradation. In contrast, our VIRTUS creates no degradation of performance in pre-installed applications because it employs a dedicated processor, and its VMM asymmetrization creates small performance degradation in downloaded applications, as well.

The major kernel-level VMMs, known as type-I, have an additional processor mode [1] [2] [9] and include Xen [3], which employs para-virtualization, and NGSCB [6], which is based on Windows. Both of these operate in a separate, additional processor-mode and can create any number of domains. They create little degradation in performance since, unlike the case of type-II VMMs, the additional processor mode enables many of an application's instructions to be executed directly on the processor without a need for emulations. There remains, however, a significant problem with performance interference on pre-installed applications by downloaded applications, since domains are interchangeably executed on a single processor. Further, when a traditional embedded processor (*i.e.*, one not implemented with such an additional mode) is to be used, the required processor modification increases system cost.

By way of contrast, the physical processor-level separation of VIRTUS suppresses the performance interference on pre-installed applications by downloaded applications, and system cost remains almost unchanged, since traditional embedded processors can be used without modification and area requirements are not a problem with fabrication technology of 90nm or better.

3. VIRTUS ARCHITECTURE

The basic VIRTUS architecture with three processors is shown in Figure 1. VIRTUS consists of a base domain on one processor (*i.e.*, physical processor#0) and other domains on the rest of processors.

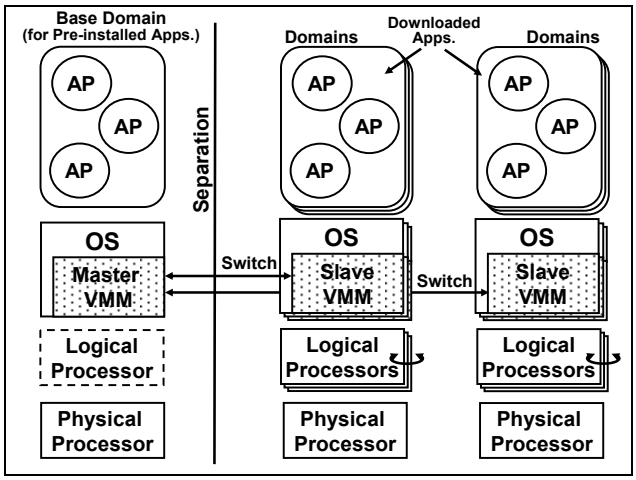

Figure 1: A New Processor Virtualization Architecture for Security-Oriented Mobile Terminals

A base domain contains pre-installed applications for the basic functions of a mobile terminal, such as a mailer and a browser. Downloaded applications are never executed on the base domain, which is only assumed to be sufficiently trustworthy to handle the basic functions of the mobile terminal.

In the downloading of native application, a download manager on a base domain controls the domains required for downloaded applications, such as a Symbian OS domain, a Linux OS domain, an operator domain, and a manufacturer domain, on logical processors of the other processors through processor virtualization software. In this way, pre-installed applications on a base domain are protected from the behavior of all applications on any number of other domains. Further, any mobile terminals' OSs are supported. Thus, VIRTUS has a better potential ability to realize a security-oriented next-generation mobile platform.

However, in order to put this VIRTUS architecture to practical use, we had to address three issues in the architecture: 1) processor virtualization software on a multiprocessor, 2) communication with applications on dormant domains, and 3) efficient hardware support for such virtualization.

3.1 VMM Asymmetrization

Traditional VMMs for processor virtualization are designed to run on a single processor. The simplest approach to apply such VMM software to a multiprocessor is parallel VMMs (*i.e.*, VMM symmetrization). However, such symmetrization causes a problem in which other VMMs can not access shared resources when a VMM that holds the resources might be destroyed due to bugs in downloaded applications. This means that using parallel VMMs is unsuitable for security-oriented mobile terminals. Thus, we developed a new VMM technology for VIRTUS, called VMM asymmetrization, that can separate a VMM into Master VMM on the base domain and Slave VMMs on other domains. This technology can address the problem, since the Master VMM handles all the resources (*e.g.*, domain contexts), and Slave VMMs work only as a stateless function to switch to a new logical processor.

The Master VMM, which works as a scheduler for logical processors in VIRTUS, provides only two APIs (functions) for the applications in the base domain. One API sets the new processor context of a logical processor to the Master VMM. In general, a processor context consists of the processor registers, which must be saved in order to enable the processor to be restored to the same condition (*e.g.*, mode registers, processor status registers, and CP15 registers for ARM926EJ-S). The other API activates a logical processor, which is set by the first API. When this API is called, the Master VMM sends a request to a Slave VMM on the physical processor to which the logical processor is assigned and waits for the completion of its activation. At the completion, the context of the previous logical processor on the requested physical processor is saved in the Master VMM through inter-VMM communication with an inter-processor interrupt. This inter-VMM communication protocol plays an important role in enabling VMM asymmetrization.

The Slave VMM, which is deeply embedded in an interrupt handler as a logical processor switcher, provides no APIs. The Slave VMM's main function is to save the context of the current logical processor in order to switch to a new logical processor.

The roles of the Master and Slave VMMs are shown in Figure 2. Three buffers, called "current", "previous", and "next", characterize this inter-VMM communication to handle multiple overlapped requests. Here, let us explain a simple example which the Master VMM has received a context switch request to a logical processor on the physical processor#k. First, the Master VMM puts the context of the logical processor into the "next" section allocated for physical processor#k on the shared memory. Second, the Master VMM sends the context switch request to the Slave VMM on the physical processor#k through inter-VMM communication and waits for acknowledgment. After that, the Slave VMM copies the data from the "next" section into the "current" section in order to prevent a context overwrite of the contexts requested by the Master VMM. The old context of the physical processor#k, which the interrupt vector has stored to a temporal buffer at the interrupt for inter-VMM communication, is saved to the "previous" section. Finally, the Slave VMM replies the acknowledgment to the Master VMM and switches to the context of the requested logical processor. The atomicity between the acknowledgment and the context switch are maintained by disabling the interrupts on the Slave VMM. After that, the Master VMM gets the old context of the physical processor#k from the "previous" section in the shared memory.

Note that the domain scheduling in VIRTUS would be the same as Java domain scheduling supported by current mobile terminals.

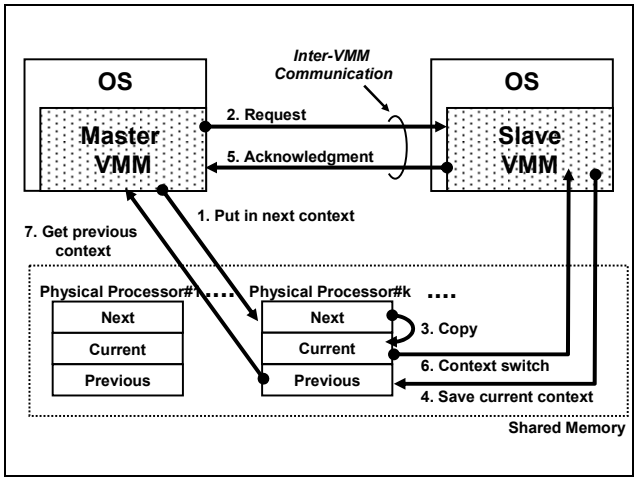

Figure 2: Efficient Inter-VMM Communication for VMM Asymmetrization

3.2 Dynamic Inter-Domain Communication

Inter-Domain Communication (IDC) plays an important role in achieving better cooperation between applications. However, since normal IDC can support only communication between applications on running domains, it was difficult to achieve communication between an application on a running domain and an application on a dormant domain (*i.e.,* a non-running domain), since the dormant domain has to be activated on demand. Thus, we developed a new IDC mechanism, called dynamic IDC, that achieves such IDC for dormant domains.

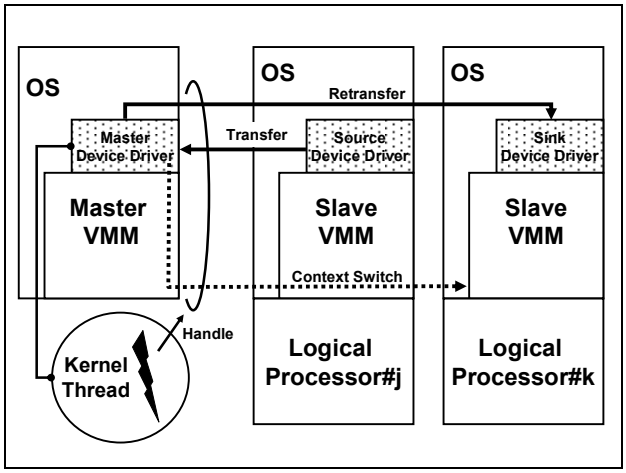

Figure 3: Dynamic Inter-Domain Communication

The main feature of dynamic IDC is a communication mechanism with context switches through the Master VMM (see Figure 3).

Dynamic IDC is embedded in device drivers (*e.g.*, drivers for intra-network). We assume that an application on logical processor#j communicates with an application on logical processor#k. When logical processor#k has already run on a physical processor, the basic function of the device drivers remains the same as the function of traditional device drivers. Otherwise, the source device driver sends a request for a context switch to the Master VMM in order to wake up logical processor#k. Furthermore, the source device driver transfers data to a master device driver on the Master VMM. After the context switch, the master device driver on the Master VMM retransfers the data to the sink device driver instead of the source device driver. A kernel thread inside the master device driver, which processes all the requests for context switches, retransfers the data form a source device driver to a sink device driver.

3.3 Virtualization-Assist Logic

Two kinds of virtualization-assist logic are effective for VIRTUS to achieve processor virtualization: logical processor ID registers and processor separation logic.

In general, a multiprocessor System-on-a-Chip (SoC) provides physical processor ID registers to identify a processor [4] [16]. The virtualization of such processor ID registers is useful for logical processors, whereas traditional VMMs for a single processor identify domains by software IDs like IP addresses. Here, let us explain an example of memory-mapped physical processor ID registers. Figure 4 shows logical processor ID registers. The main feature of this logic is writable processor ID registers, which can be over-written only from physical processor#0, for downloaded applications' domains. That is, this logic is designed to be handled by the Master VMM at a context switch so that an activated logical processor can identify itself.

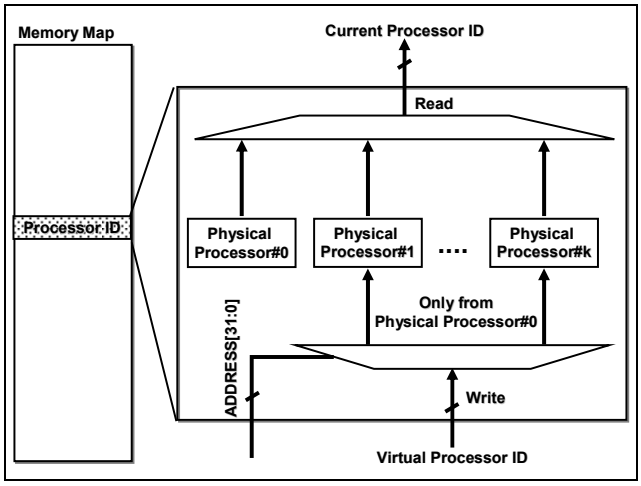

Figure 4: Logical Processor ID Registers

Processor separation logic, which is embedded in a system bus, is also effective to protect important resources of base domain from downloaded applications on other virtualized domains. The basic role of this logic determines whether access from a processor to a bus slave should be granted, which is based on the access matrix between bus masters and bus slaves at the bus level (*i.e.,* access control). For example, a processor for a base domain can access any I/Os and memories, whereas other processors for downloaded applications can not access important I/Os and address ranges of memories that the base domain has used.

Processor separation logic consists of two types of logics: physical processor separation logic [8], which enables resources used by a physical processor to be protected from access issued by other physical processors; and logical processor separation logic, which enables resources used by a logical processor on a physical processor to be protected from access issued by other logical processors on the physical processor by means of logical processor ID registers. Physical processor separation logic helps isolate a base domain from other virtualized domains, whereas logical processor separation logic helps isolate all of the processors' virtualized domains in addition to the separation of a base domain and other virtualized domains. Although the preferable type of processor separation logic depends on system security policy, processor separation logic would make VIRTUS more robust in addition to faster domain separation at the hardware level.

4. EVALUATION

4.1 Evaluation Environment

The evaluation environment is summarized in Table 1. We used a mobile application processor, MP211 [16] of NEC Electronics. The main feature of this chip is its multiprocessor architecture with three ARM processors. The MP211 helps extract process-level parallelism from mobile terminal software [13] in order to achieve higher performance and lower power dissipation than a single processor.

Table 1: Evaluation Environment

Item	Feature
SoC	MP211
Processor	ARM926EJ-S x 3 (200MHz)
Cache	I: 16KB, D: 16KB
Linux	Embedded Linux 2.4.20

4.2 Virtualization Overhead

In order to prove that VIRTUS creates little degradation of OS performance in downloaded native applications, we executed processes and context switching micro-benchmarks of LMbench [10] on a downloaded application domain. Note that this LMbench with 37 micro-benchmarks is a typical benchmark to measure single OS performance on PCs.

From 24 micro-benchmarks shown in [3], we selected 13 micro-benchmarks measurable in our evaluation environment. The evaluation results of VIRTUS are shown in Figure 5 and Figure 6. Further, in reference to [3], the results of other VMMs, Xen (a type-I VMM) and UML (a type-II VMM), are also shown in the figures. Here, respectively in two evaluation environments, each performance of the micro-benchmarks executed on non-virtualized Linux is normalized to 1 for the sake of comparison.

In process micro-benchmarks for system call performance (see Figure 5), VIRTUS achieved an average performance 1.2 times higher than that of Xen and 58 times higher than that of UML. Particularly, VIRTUS improved "fork" and "exec" performance to be significantly better than Xen, since the OS on the Slave VMM can update large numbers of the page table without any calls to the Master VMM.

In context switching micro-benchmarks for the context switching time between different numbers of processes with different working set sizes (see Figure 6), VIRTUS achieved an average performance 2.1 times higher than that of Xen and 20 times higher than that of UML. In particular, VIRTUS dramatically improved the context switch time for smaller working set sizes, since an OS on a Slave VMM can change the page table base without any calls to the Master VMM. Thus, VIRTUS can offer less performance degradation even in downloaded native applications.

In addition, physical processor-level separation enables VIRTUS to create no degradation of performance in pre-installed applications and less performance interference on pre-installed applications by downloaded applications, since the base domain and other domains run on different processors.

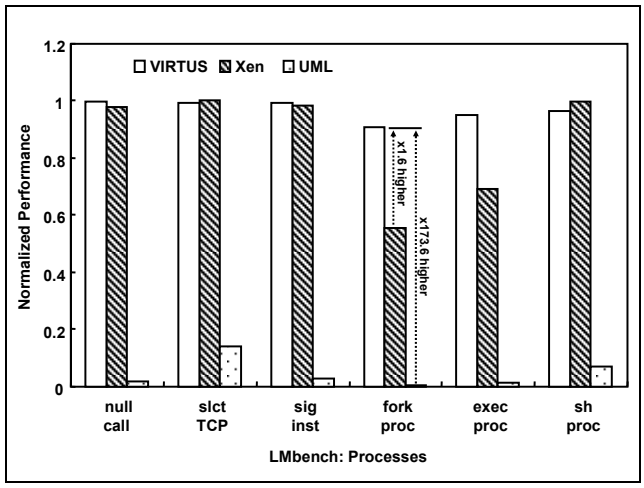

Figure 5: Enhanced Performance of Process Micro-Benchmarks

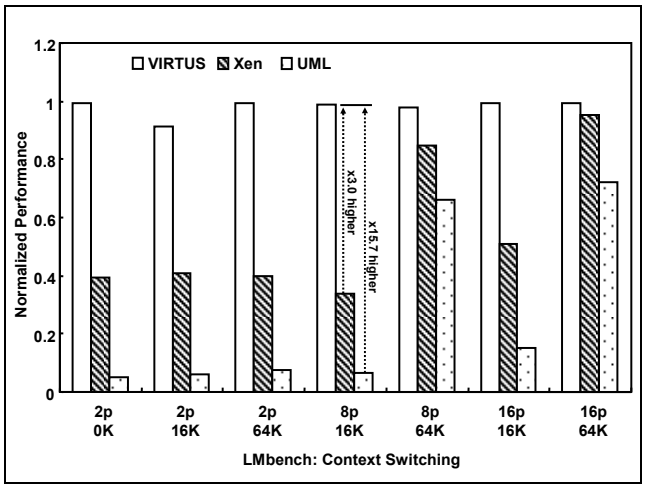

Figure 6: Enhanced Performance of Context Switching Micro-Benchmarks

4.3 Lines of Code

We compared the Lines Of Code (LOC) used in VIRTUS with those in the other VMMs, Xen and UML. Figure 7 shows that the VIRTUS virtualization software, which includes a Master VMM and a Slave VMM, can be implemented in the smallest size; the modified LOC of the kernel is 1.7 times smaller than that of Xen and 7.4 times smaller than that of UML. The text size of VIRTUS virtualization software is only 4 KB. Additionally, the VIRTUS network device driver can be implemented in modified code 2.0 times smaller than that of Xen and 3.7 times smaller than that of UML. This code reduction is achieved by VMM asymmetrization, which simplifies processor virtualization by having a dedicated processor for Master VMM, whereas other VMMs need extra functions, such as page table update and inter-domain data transfer, to handle multiple domains on a single processor.

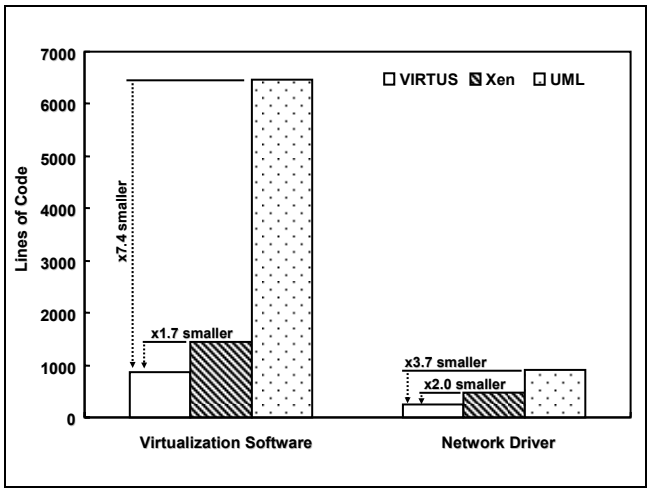

Figure 7: Smaller Modified LOC of VIRTUS

4.4 Dynamic IDC Overhead

In order to evaluate dynamic IDC overhead, we measured the overhead of the context switches between domains. The average context switch overhead of the 20 time measurements was less than 0.5 ms. This context switch overhead is not a problem with Linux-based mobile terminals, since the time quantum of a process on embedded Linux (*e.g.,* 150ms) is much longer than the context switch overhead.

4.5 Virtualization-Assist Logic

Logical processor ID registers can be achieved by making a minor change from basic processor ID registers. And, the estimated number of transistors needed for simple physical processor separation logic is less than 20 K. According to ITRS 2003 [14], the transistor density logic (M transitors/cm^2) of an MPU is 77 in the hp-90nm fabrication technology node. Thus, the logic can be implemented in a sufficiently small area (less than 0.026 mm^2). Further, the access latency and the power dissipation of the logic are not a problem.

4.6 Virtualization Example

Figure 8 shows that five logical processors (*i.e.*, five Linux OSs) run on three ARM physical processors of MP211. This is the world's first virtualization on an ARM-based multiprocessor. Physical processor#1 has two logical processors, #1 and #3, and physical processor#2 has two logical processors, #2 and #4. In the lower half of the figure, 3D graphic processes show the operating states of each logical processor. In the top half of the figure, a control board process on physical processor#0 manages running logical processors by context switches through dynamic IDC. In this figure, logical

processors #3 and #4 are running and logical processors #1 and #2 are dormant.

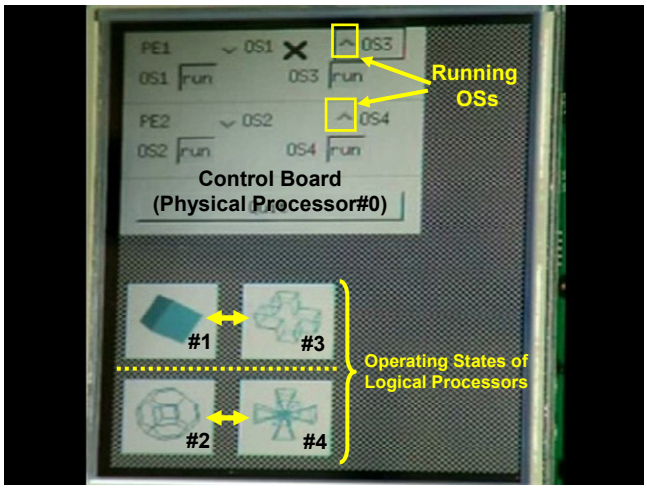

Figure 8: Five Linux OSs on Three ARM Physical Processors

5. CONCLUSION

Next-generation mobile terminals are expected to require a large number of domains in order to support the downloading of native applications. Numerous research has been conducted on domain creation, targeting only at non-embedded environments.

This paper described a new processor virtualization architecture, VIRTUS, which satisfies embedded systems requirements: hardened security, flexible virtualization, small performance degradation, little performance interference, and small code size. In forming a hardware domain for pre-installed applications on one processor and software domains for downloaded applications on other processors, VIRTUS is a hybrid approach to exploit the advantages of traditional approaches, both hardware and software domain approaches.

VIRTUS features three new technologies: VMM asymmetrization, through which the hardware domain and the software domains operate cooperatively on different processors; dynamic IDC, which enables an application on a running domain to communicate with an application on a dormant domain; virtualization-assist logic for better virtualization.

In evaluations, VIRTUS was able to virtualize five Linux OSs on three ARM processors. This represents the world's first virtualization on an ARM-based multiprocessor. Evaluations have further shown that VMM asymmetrization results in significantly less degradation of performance and significantly less LOC increase than do other VMMs. Further, dynamic IDC overhead is low enough for Linux-based mobile terminals, and virtualization-assist logic can be implemented in a sufficiently small area. We strongly believe VIRTUS to be ideally suited to next-generation security-oriented mobile terminals that will require an increased number of domains over four processors.

6. ACKNOWLEDGMENTS

Authors would like to express our deep appreciation for great contributions from Hiroshi Chishima, Masakazu Yamashina, Masajiro Fukunaga, Naoki Nishi and Naotaka Sumihiro.

7. REFERENCES

[1] Alves, T. and Felton, D. *TrustZone: Integrated Hardware and Software Security.* White Paper, July 2004.

[2] Armstrong, W. J. et al. Advanced Virtualization Capabilities of POWER5 Systems. *IBM Journal of Research and Development, 49*, 4/5(July/Sept. 2005), 523-532.

[3] Barham, P. et al. Xen and the Art of Virtualization. In *Proceeding of the ACM Symposium on Operating Systems Principles,* October 2003, 164-177.

[4] Culler, D. E. and Singh, J. P. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1999.

[5] Dike, J. A User-Mode Port of the Linux Kernel. In *Proceedings of the 4th Annual Linux Showcase & Conference,* October 2000, 63-72.

[6] England, P. et al. A Trusted Open Platform. *IEEE Computer, 36*, 7(July 2003), 55-62.

[7] Gong, L. and Ellison, G. *Inside Java 2 Platform Security: Architecture, API Design and Implementation (The Java Series)*. Addison-Wesley, 2003.

[8] Inoue, H. et al. FIDES: An Advanced Chip Multiprocessor Platform for Secure Next Generation Mobile Terminals. In *Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS),* September 2005, 178-183.

[9] Intel. *Intel Virtualization Technology Specification for the IA-32 Intel Architecture.* C97063-002, April 2005. ftp://download.intel.com/technology/computing/vptech/C97063-002.pdf.

[10] McVoy, L. and Staelin, C. lmbench: Portable Tools for Performance Analysis. In *Proceedings of the USENIX Annual Technical Conference,* January 1996, 279-294.

[11] NTT DoCoMo, IBM, and Intel Corporation. *Trusted Mobile Platform: Hardware Architecture Description.* June 2004. http://www.trusted-mobile.org/.

[12] NTT DoCoMo, IBM, and Intel Corporation. *Trusted Mobile Platform: Software Architecture Description.* June 2004. http://www.trusted-mobile.org/.

[13] Sakai, J. et al. Multi-Tasking Parallel Method on MP211 Multi-core Application Processor. In *Proceedings of the IEEE Symposium on Low-Power and High-Speed Chips (COOL Chips VIII),* April 2005, 198-211.

[14] Semiconductor Industry Association et al. *International Technology Roadmap for Semiconductors 2003 Edition, Executive Summary.* 2003.

[15] Sugerman, J. et al. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of the USENIX Annual Technical Conference*, June 2001, 1-14.

[16] Torii, S. et al. A 600MIPS 120mW 70uA Leakage Triple-CPU Mobile Application Processor Chip. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC), Digest of Technical Papers*, February 2005, 136-137.