# Linux/RTOS Hybrid Operating Environment on Gandalf Virtual Machine Monitor

Shuichi Oikawa[1], Megumi Ito[1], and Tatsuo Nakajima[2]

[1] Department of Computer Science, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573, Japan
[2] Department of Computer Science, Waseda University
3-4-1 #61-505, Okubo, Shinjuku, Tokyo 169-8555, Japan

**Abstract.** This paper presents our Linux/RTOS hybrid operating environment constructed upon Gandalf VMM. Gandalf can host multiple RTOSes along with Linux, and RTOSes and Linux execute within their own isolated protection domains; thus, they can be spatially and temporally protected from each other. We design Gandalf from scratch as a simple and efficient VMM in order to minimize overheads incurred by virtualization. The simplicity and efficiency are achieved by the hybrid of para- and nearly full-virtualization approaches. The implementation of the presented hybrid operating environment is on the PC/AT compatible platform with the Intel IA-32 processor with $\mu$ITRON as an RTOS. From the measurement results, we make clear that the benefits of using a VMM to construct a hybrid environment exceed shortcomings by showing the impact on performance is limited.

## 1 Introduction

The current embedded systems, especially consumer electronics products, contain a number of complicated requirements for their operating environments that are difficult to satisfy them together at once by a single operating system (OS). Consumer electronics products need to be easy-to-use to appeal not only to enthusiastic users but to a wide range of ordinary people; thus, they are nowadays equipped with many applications along with fancy GUI. On the other hand, they have essential functionality that must work reliably in a timely manner. Those functionality requires real-time processing, and their requirements for real-time processing tend to be difficult to be satisfied by an OS that can provide fully featured GUI.

In order cope with such conflicting requirements imposed by complex embedded systems, the hybrids of a general purpose OS and a real-time operating system (RTOS) have been developed. While a general purpose OS is good at the provision of GUI with its rich set of middleware support, an RTOS covers real-time processing. Those hybrids have a general purpose OS kernel and an RTOS's application tasks share the same most privileged level of a processor without protection. This hybrid model may work well if a system is designed from scratch having applications properly classified into real-time and non real-time tasks. In reality, however, the existing software resources inherited from the past products need to be reused; thus, undesired programs are also brought

into the most privileged level to run on an RTOS, and can become sources of problems because of lack of protection.

This paper presents our hybrid operating environment constructed upon Gandalf virtual machine monitor (VMM). As an RTOS and a general purpose OS hosted on Gandalf, we use $\mu$ITRON [14] and Linux, respectively. The implementation of this hybrid operating environment is on the PC/AT compatible platform with the Intel IA-32 processor. We designed Gandalf from scratch as a simple and efficient VMM in order to minimize overheads incurred by virtualization. Unlike the existing approach described above, only Gandalf executes at the most privileged level, and has them execute within their own isolated protection domain; thus, hosted OSes can be spatially and temporally protected from each other and from a general purpose OS. Additionally, Gandalf can host multiple RTOSes along with a general purpose OS. We chose $\mu$ITRON and Linux because of their popularity. Since in Japan $\mu$ITRON is the RTOS that has been the most widely used in a variety of products, industries have a huge amount of the existing software resources. Linux's popularity is recently increasing for larger embedded systems.

The features of this system are summarized as follows:

- The provision of spatially and temporally protection is enabled by constructing a hybrid of an RTOS and a general purpose OS on a VMM.
- The hybrid of para- and full-virtualization approaches is implemented in Gandalf VMM in order to balance implementation cost and runtime cost.
- We introduce nearly full-virtualization, by which we allow a few straightforward modifications to bring up Linux on Gandalf. Nearly full-virtualization enables the significant reduction of both implementation and runtime costs.

### 1.1 Background and Related Work

Making hybrids of an RTOS and a general purpose OS is not a new idea. As a practical approach to deal with complex systems, several of them have been developed, and some are widely used. [3] introduced the executive that support the co-residence of an RTOS and a general purpose OS. RTLinux [1] and RTAI [8], which are poplar among Linux users, have Linux kernel execute on a RTOS kernel. There are some commercial products, such as Accel-Linux and Linux on NORTi, that enable $\mu$ITRON to run aside of Linux kernel.

All of them take the same approach, which is that an RTOS, RTOS's application tasks, and a general purpose OS kernel share the same most privileged level; thus, there is no protection among them. Such lack of protection may not be a problem if a system is designed from scratch having applications properly classified into real-time and non real-time tasks. After proper classification, there should be only a small number of RTOS's application tasks that specifically require real-time execution. It is, however, problematic if the existing applications on an RTOS are simply reused by taking advantage of the hybrid of an RTOS and a general purpose OS. In such a system, there tend to be a lager number of RTOS's application tasks. Their misbehavior is directly connected to system malfunction or a crash. A general purpose OS kernel also can be a source of problems because of its execution at the most privileged level. A general purpose OS

kernel for the hybrid with an RTOS is usually modified not to touch hardware's interrupt controlling functions, so that interrupts are not disabled for a indeterministically long time. Since a general purpose OS kernel is still allowed to disable interrupts by controlling hardware, there are chances to introduce kernel modules that are not properly modified for the hybrid; thus, they cause temporal malfunction.

Our approach is that Gandalf VMM hosts RTOSes and a general purpose OS. This approach enables the provision of spatial and temporal protection. Spatial protection is implemented by having OSes run within their own protection domains. Since there is no means provided to corrupt or steal the programs or data of the other OSes, OSes' misbehavior does not affect the execution of the other OSes. Temporal protection is realized by limiting hardware access by OSes. The OSes hosted on Gandalf execute at a less privileged level, at which only limited hardware access is allowed; thus, the hosted OSes cannot directly disable interrupts at hardware's interrupt controller. Therefore, temporal malfunction incurred by disabling interrupts can be avoided.

The development of VMMs has a long history beginning with IBM CP/67 [10] followed by IBM VM/370 [5] and its alikes for mainframe computers. Since a few years ago VMMs revived to be a hot research and development topic because of VMMs' capability to accommodate multiple operating systems in a single machine. Researchers and developers consider VMMs to deal with increasing reliability and security. In order to achieve better performance on commodity platforms, which cannot be virtualized efficiently, para-virtualization was introduced [15]. With para-virtualization, VMM developers define easily and efficiently virtualizable hardware as interface with a VMM. Para-virtualization can be used for only I/O devices [13] or platforms [2]. In contrast to para-virtualization, which defines virtual non-existing hardware, the approach first taken by mainframe VMMs is called full-virtualization, which defines the same interface as the existing real hardware.

Gandalf VMM takes the hybrid of para- and full-virtualization approaches. Gandalf exports a virtual processor interface for RTOSes while it enables a general purpose OS to run on it with limited modifications. Such a hybrid approach can balance implementation cost and runtime cost.

### 1.2 Paper Organization

The rest of this paper is organized as follows. The next section describes the overall architecture of our hybrid operating environment constructed upon Gandalf VMM. Section 3 describes the presented system's design and implementation in more detail. Section 4 describes the current status of the implementation and shows the preliminary evaluation results. Finally, Section 5 summarizes the paper.

## 2 Overall Architecture

This section describes the overall architecture of our Linux/RTOS hybrid operating environment constructed upon Gandalf VMM.

Figure 1 depicts that Gandalf implements protection domains in order to isolate the execution environments of $\mu$ITRON RTOS and Linux. Realizing protection domains is

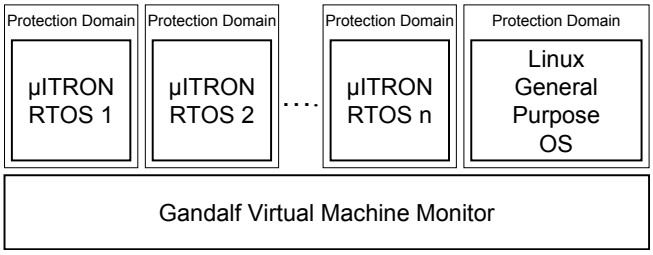


**Fig. 1.** Overall Architecture of Linux/RTOS Hybrid Operating Environment on Gandalf

hardware dependent. Since this hybrid operating environment is implemented on the Intel IA-32 processor, segments associated with protection levels can provide protection domains. Gandalf, RTOSes, and Linux execute within their own segments. The segments of RTOSes and Linux do not overwrap; thus, their memory access is contained within their segments. Gandalf executes at the most privileged level while RTOSes and Linux executes at the less privileged level; thus, only limited hardware access by RTOSes and Linux is allowed. Physical memory is statically partitioned at a time of configuration, and the fixed amounts of memory are allocated for Gandalf, RTOSes, and Linux at a boot time.

Gandalf VMM takes the hybrid of para- and full-virtualization approaches. We choose para-virtualization for RTOSes and full-virtualization for a general purpose OS in order to balance implementation cost and runtime cost. A general purpose OS is huge and very complicated software. It tends to be actively updated in order to incorporate new features and to fix their bugs. Applying para-virtualization to such an OS significantly increases implementation cost. It is also hard to maintain the source code modified for para-virtualization since it needs to keep up with rapid updates. On the other hand, the implementation of RTOSes is considerably simpler than a general purpose OS, and it tends to be stable for a long time because keeping reliability is more important than adding new features; thus, once their source code base is modified for para-virtualization, the amount of its maintenance work is limited. Therefore, its implementation cost is negligible. In fact, runtime cost is more important for RTOSes. In order to have unmodified OSes execute at a less privileged level, full-virtualization emulates a subset of hardware. Such emulation costs at runtime. Para-virtualization can decrease the overheads of emulation; thus, using para-virtualization is desired for RTOSes in terms of both implementation and runtime costs.

## 3 Detailed Design and Implementation

This section describes the detailed design and implementation of our hybrid operating environment implemented on the PC/AT compatible platform with the Intel IA-32 processor.

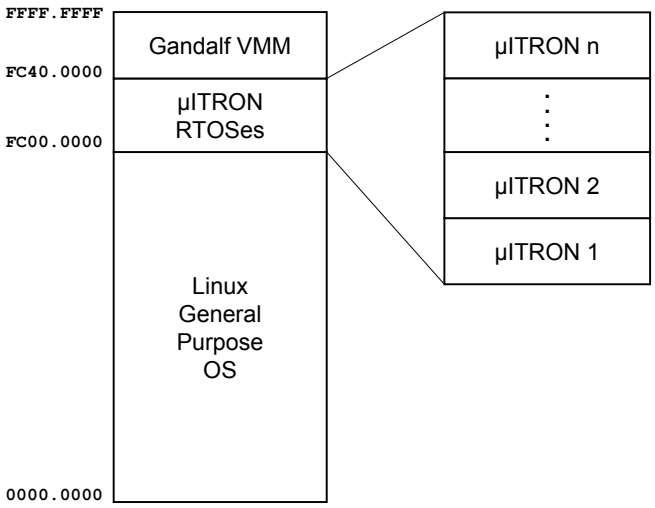


**Fig. 2.** Virtual Address Map

### 3.1 Address Map, Segments, and Protection

As described in the previous section, physical memory is statically partitioned at a time of configuration, and the fixed amounts of memory are allocated for Gandalf VMM, $\mu$ITRON instances, and Linux at a boot time. In a physical address map, Linux takes the first part of physical memory, then $\mu$ITRON instances comes next, and Gandalf takes the last part. There can be multiple $\mu$ITRON instances hosted on Gandalf. They, however, do not share physical memory.

The layout of Linux, $\mu$ITRON instances, and Gandalf in physical memory is reflected in virtual memory but at different addresses. At this moment, virtual memory is also statically partitioned at a time of compilation. The partitioning can be easily changed by parameters. Figure 2 depicts their current layout in virtual memory. Linux uses the first part from virtual address `0x0` to `0xfbff.ffff`.[3] $\mu$ITRON instances are located from `0xfc00.0000` to `0xfc3f.ffff`. Gandalf uses the top part from `0xfc40.0000` until the end of virtual address `0xffff.ffff`. Limiting a virtual address range available for Linux requires a simple modification to Linux's source code.

Protection domains of OSes and Gandalf are realized by segments of the Intel IA-32 processor. Each segment has its base address and size, and memory access is allowed only within the current segment. Gandalf allocates two segments, one for program code and another for data, to each of OSes and Gandalf. Only Gandalf has control of segments. Gandalf's segments cover the whole virtual memory from `0x0` to `0xffff.ffff`; thus, it can access any part of virtual memory. Linux's segments start at `0x0` and end at `0xfbff.ffff`. The segments of the first $\mu$ITRON instance start

[3] A 32-bit address is denoted with a pair of 16-bit numbers connected with a period for better readability.

at `0xfc00.0000`, and the last ones end at `0xfc3f.ffff`. Since the segments of RTOSes and Linux do not overwrap, hosted OSes are spatially protected from each other by segments.

### 3.2 $\mu$ITRON RTOS

$\mu$ITRON is actually a specification of a simple embedded RTOS that provides real-time tasks, synchronization and communication mechanisms, system services, and device drivers; thus, there are several independently developed commercial and open source implementations. We use TOPPERS/JSP [4], which is an open source implementation of $\mu$ITRON, for our RTOS.

We chose para-virtualization for $\mu$ITRON as described in the previous section in favor of less virtualization overheads at runtime. Para-virtualization replaces privileged instructions, which can be executed at the most privileged level, with hypercalls, which are systemcalls provided by Gandalf VMM. By taking advantage of the knowledge of $\mu$ITRON's implementation and having Gandalf tailored to execute a $\mu$ITRON instance in specific segments, very few hypercalls are actually required in order to bring up $\mu$ITRON on Gandalf.

The current implementation of Gandalf provides $\mu$ITRON with only three hypercalls as replacements of privileged instructions. One replaces `lidt` instruction, which is used to register interrupt handlers. Another one replaces `sti` and `cli` instructions, which enables and disables interrupts, respectively. The last one replaces `hlt` instruction, which halt a processor until an interrupt is asserted. While some other privileged instructions are used, they were removed because tailoring Gandalf to provide an execution environment expected by $\mu$ITRON makes them no longer needed.

### 3.3 Linux General Purpose OS

We use Linux as a general purpose OS. We chose full-virtualization to support Linux since Linux kernel is huge and complicated software and is actively updated to incorporate new features and to fix their bugs. Truly full-virtualization, however, costs quite expensive to implement a VMM and to execute an unmodified OS on it. It requires a fully virtualizable processor [11], or it is made possible only in return for the overheads of virtualizing all computing resources. For example, an OS kernel is designed to utilize the whole virtual address space made available by a processor. If a processor is not fully virtualizable as most of the current processors, there is no room in the same virtual address space left for locating a VMM in a way that the it is protected from its guest OS kernel and user processes.

In this case, a VMM requires another virtual address space, and heavy context switching between a OS kernel and a VMM happens at every time the VMM's intervention is needed. Such intervention includes the emulation of privileged instructions, handling interrupts and exceptions, and so on. There are many other virtualization overheads caused by full-virtualization. Achieving practical performance that matches commercial VMMs, such as VMware [13] and Microsoft VirtualPC, requires many

[4] http://www.toppers.jp/

techniques including on-the-fly binary translation [12]; thus, the provision of truly full-virtualization is considered to be cumbersome.

We therefore decided to allow a few straightforward modifications to bring up Linux on Gandalf. We call this approach *nearly* full-virtualization. Allowing a few modifications enables the significant reduction of both implementation and runtime costs. For example, by reducing the virtual address range used by Linux, we can create room for $\mu$ITRON and Gandalf in the same virtual address space. It removes the necessity to switch virtual address spaces at each time when Gandalf is invoked. Such reduction of the virtual address range can be done only by modifying a single line in a Linux source code file. Thirteen lines in seven files are currently modified to achieve nearly full-virtualization of Linux on Gandalf.

### 3.4 Gandalf Virtual Machine Monitor

Gandalf is a virtual machine monitor designed and built from scratch. Our design principle of Gandalf is simplicity. In order to follow this principle, Gandalf takes the hybrid of para- and full-virtualization approaches, but full-virtualization is approximated as nearly full-virtualization. While the hybrid of two virtualization approaches may seem to add complexity, it actually simplifies the implementation since the distinction of callee OSes is made at the entry point to Gandalf and there is no need to differentiate them at each operation. Applying nearly full-virtualization to bring up Linux on Gandalf also makes Gandalf significantly simpler than real full-virtualization as described previously.

Gandalf is responsible for the provision of the execution environments of $\mu$ITRON RTOS and Linux. For that purpose, Gandalf provides three functionalities, 1) a boot loader for Linux and $\mu$ITRON, 2) the emulation of privileged instructions for Linux, and 3) the hypercalls for $\mu$ITRON. First, Gandalf works as a boot loader for Linux and $\mu$ITRON. We use GRUB as an actual boot loader, which loads Gandalf, Linux, and $\mu$ITRON into memory, and transfers the execution to Gandalf. GRUB passes to Gandalf the information about the memory locations of Gandalf itself, Linux, and $\mu$ITRON. By using the passed information, Gandalf performs the initialization to set up the physical and virtual memory maps as described in Section 3.1. Gandalf first boots up $\mu$ITRON and then Linux.

Gandalf emulates privileged instructions executed by Linux. Since Linux kernel executes at a less privileged level, its issuing a privileged instruction causes an exception. The type of such an exception is a general protection fault in case of the Intel IA-32 processor. Gandalf registers an exception handler for a general protection fault, so that Gandalf handles it when it occurs. Gandalf's exception handler for a general protection fault examines the instruction at the address where an exception occurred. If the instruction is a privileged instruction, Gandalf emulates it accordingly.

$\mu$ITRON uses the hypercalls instead of privileged instructions. Gandalf implements only three hypercalls as replacements of privileged instructions, such as `lidt`, `sti/cli`, and `hlt`, as described in Section 3.2.

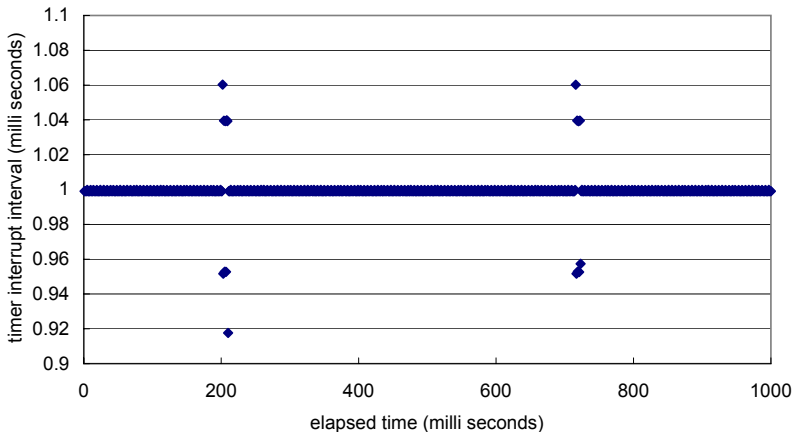


**Fig. 3.** Timer Interrupt Intervals in $\mu$ITRON

## 4 Current Status and Evaluation Results

We have implemented Gandalf and constructed the our Linux/RTOS hybrid operating environment upon it. The rest of this section shows the preliminary evaluation results obtained from the current implementation. All measurements reported below were performed on the Dell Precision 470 Workstation with Intel Xeon 2.8GHz CPU. [5] Hyper-threading was turned off. Please note that since there is no benchmark program that is appropriate for the evaluations of a hybrid environment, we are currently developing evaluation programs for this purpose.

First, we measured the switching cost between $\mu$ITRON and Linux. It was measured by using a pair of `hlt` instruction in Linux and its replacement hypercall in $\mu$ITRON. Gandalf was instrumented to switch to the other OS when they were executed. We used cycle counts obtained from `rdtsc` instruction for this measurement. The average cost to switch from $\mu$ITRON to Linux and then back to $\mu$ITRON, which means that two OS switchings are involved, is 1.02 $\mu$ seconds after repeating 10,000 times.

Second, we measured the timer interrupt intervals in $\mu$ITRON. We used cycle counts obtained from `rdtsc` instruction for this measurement, too. Figure 3 shows the measurement result. The measured timer interrupt intervals are mostly the same at 1 millisecond as the timer device was configured to periodically raise an interrupt every 1 millisecond. There are, however, some spikes around 200 and 700 milliseconds in elapsed time. We need more investigations to be performed in order to find out a cause of these spikes.

Finally, in order to evaluate our nearly full-virtualization approach used for Linux, we ran several programs included in lmbench benchmark suite [9]. Figure 4 and 5 show the results of lmbench programs. We ran the same programs on the original Linux

[5] Linux reports this CPU as 2794.774 MHz. We use this number to convert cycle counts obtained from `rdtsc` instruction to micro seconds for accuracy.

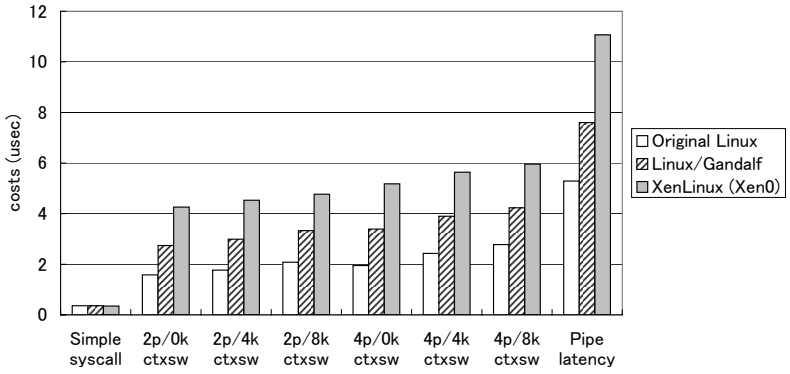


**Fig. 4.** Linux Performance Comparison (1)

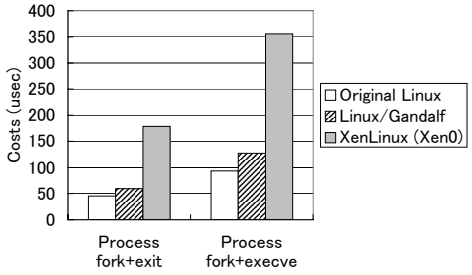


**Fig. 5.** Linux Performance Comparison (2)

(without virtualization) and XenLinux (Dom0) for comparison of performance. The measurement results show that nearly full-virtualization approach reduces the runtime costs significantly as Linux on Gandalf outperforms XenLinux in all cases. The costs of process fork and exec are close to the original non-virtualized Linux and significantly better than XenLinux.

## 5 Summary

We presented our Linux/RTOS hybrid operating environment constructed upon Gandalf VMM. we use $\mu$ITRON as an RTOS. Since RTOSes and Linux execute within their own isolated protection domains, they can be spatially and temporally protected from each other. We designed Gandalf from scratch as a simple and efficient VMM in order to minimize overheads incurred by virtualization. The simplicity and efficiency were achieved by the hybrid of para- and nearly full-virtualization approaches. The presented hybrid operating environment was implemented on the PC/AT compatible platform with the Intel IA-32 processor.

From the measurement results described in the previous section, we showed that the impact on performance by using a VMM to construct a hybrid environment is limited.

The benefits to use a VMM, by which spatial and temporal protection can be provided, exceed the limited performance loss. Therefore, our Linux/RTOS hybrid operating environment on Gandalf, which is constructed upon the hybrid of para- and nearly full-virtualization approaches, is considered to be very practical and useful.

## References


1. M. Barabanov and V. Yodaiken. Real-Time Linux. *Linux Journal*, March 1996.
2. 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 19th ACM Symposium on Operating System Principles*, pp. 164–177, October 2003.
3. G. Bollella and K. Jeffay. Support for Real-Time Computing within General Purpose Operating Systems - Supporting Co-Resident Operating Systems. In *Proceedings of the 1st IEEE Real-Time Technology and Applications Symposium*, May 1995.
4. E. Bugnion, S. Devine, K. Govil, and M. Rosenblum. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. In *Proceedings of the 16th ACM SIGOPS Symposium on Operating Systems Principles*, pp. 143–156, October 1997.
5. R. J. Creasy. The Origin of the VM/370 Time-Sharing System. *IBM Journal of Research and Development*, 25 (5), 1981.
6. R. P. Goldberg. Survey of Virtual Machine Research. *IEEE Computer*, pp. 34–45, June 1974.
7. Intel Corporation. IA-32 Intel Architecture Software Developer's Manual.
8. P. Mantegazza, E. Bianchi, L. Dozio, and S. Papacharalambous. RTAI: Real Time Application Interface. *Linux Journal*, April 2000.
9. L. McVoy and C. Staelin. lmbench: Portable Tools for Performance Analysis. In *Proceedings of the USENIX Annual Technical Conference*, pp. 279–294, January 1996.
10. R. Meyer and L. Seawright. A Virtual Machine Time Sharing System. *IBM Systems Journal*, 9 (3), pp. 199–218, 1970.
11. G. Popek and R. Goldberg. Formal Requirements for Virtualizable 3rd Generation Architectures. *Communications of the A.C.M.*, 17(7):412–421, 1974.
12. M. Rosenblum and T. Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *IEEE Computer*, pp. 39–47, May 2005.
13. J. Sugerman, G. Venkitachalam, and B. H. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of 2001 USENIX Annual Technical Conference*, 2001.
14. H. Takada ed. $\mu$ITRON4.0 Specification. TRON Association, 1999. (In Japanese)
15. A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pp. 195–210, December 2002.