Design of ISA for Efficient Virtualization

Liu Yuhang,Hao Qinfen,Xiao LiMin,Zhu Mingfa
School of Computer Science and Technology
Beijing University of Aeronautics and Astronautics
Beijing, China
liuyuhang@cse.buaa.edu.cn

***Abstract*—Popek, Goldberg have done some research on the formal requirements for virtualizable third Generation Architectures. This paper discusses how to design Instruction Set Architectures (ISAs) that supporting virtualization. Firstly, it defines formally the concepts that concerns with the issue. Secondly, it reclassifies the instructions according to instruction behaviors. Thirdly, it discusses how to reduce the proportion of instructions that are executed with intervention and interpretation of Virtual Machine Monitor (VMM) in the entire instruction set so as to expand the efficiency space in further with the premise of the whole Machine Can be Virtualized(MCV). At last, it not only gives but proves a theorem about the mapping between any instruction sequences when VMM doesn't exist and its counterpart when VMM does exist. These not only provide guidelines for the design of ISA and the construction of efficient VMM, but also help to assess the existing ISA and make some necessary modification to enable MCV.**

***Index Terms*—Instruction Set Architecture (ISA),Sensitive Instruction(SI), Efficient Virtualization**

I. Introduction

Virtual machine (VM) technologies were first introduced in the 1960s [1], but are experiencing a resurgence in recent years and becoming more and more attractive to both the industry and the research communities, and are developing rapidly[2].

Through establishing a simplified model of the third generation of computer systems, Popek and Goldberg [3] attempt to raise a criterion to test whether or not an architecture can support a virtual machine, i.e., whether or not MCV with a type of architecture. They putted forward the fundamental theorem about ISA supporting virtualization: any traditional third-generation computer, a VMM may be constructed, if this computer's sensitive instruction set is a subset of the privileged instruction set. The theorem provides a simple sufficient condition to guarantee virtualization. However, Popek and Goldberg only bring forward a sufficient but not full necessary condition which is inaccurate and sometimes ineffective. That is, if a kind of architecture meets the condition, then it supports the virtual machine; but if not, we cannot determine whether it supported virtual machines. And to satisfy the unnecessary conditions is at the cost of efficiency loss.

Hence seeking a necessary and sufficient condition for ISA supporting virtualization is necessary. An essential idea in this paper is to re-divide instructions set according to the behavior of instructions, while expanding the scope of sound instruction as large as possible.

In a computer with a type of instruction set architecture (ISA), if a control program with the following three characteristics can be constructed, then we call that the ISA supporting virtual machines, i.e., MCV. 1) The control program provides an environment in essence equals that of the real machine provides; 2) procedures running in the environment which the control program provides are efficient; 3) the control program completely controls all of the system resources.

The control program is referred as Virtual Machine Monitor (VMM), shown in Fig.1.

A virtual machine is considered as a copy of the real machine, and the virtual machines on the same real machine are isolated with each other[4]. The environment provided by the VMM is equivalent to that provided by the real machine, which is the first important characteristic of VMM.

The second characteristic of VMM is efficient. This requires most of the virtual processor instructions in the statistical sense, must be directly executed by the real processor, and not be interfered by VMM software. So we can conclude that the traditional simulator and complete software interpreter machine (CSIM) [4] does not belong to the category of virtual machine.

The third characteristic, VMM completely controls resources in the whole system, such as memory, peripherals, etc. The programs running in virtual machines cannot access any resource which is not explicit allocated to the virtual machine, and the VMM are able to regain control over the resources which have been allocated.

II. A Formal Model of Modern Computer

Compared with the third generation computer, in terms of basic organizations, there is no significant change on modern computer system architecture. And modern computer systems

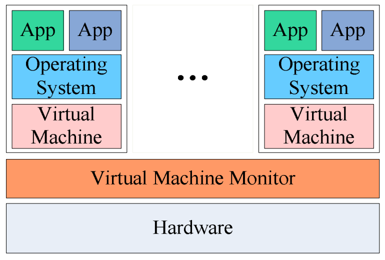

Figure 1. A general model of Virtual Machine

have generally provided users with extended machine that there is no explicit I/O device or I/O instruction, e.g., I/O device is treated as memory units, and I/O operation is implemented through appropriate memory transfer. In order to reflect the essence of the system and at the same time keep the simplicity to facilitate the following discussion, we exclude I/O device or I/O instruction out of the model.

Computer system can be in one of a limited number of states, $S =< E, M, P, R >$, here the status S is composed of four components: executable memory E, processor model M, program counter register P and relocation bound register $R = (l, b)$, as is shown in Fig.2.

Memory addressing is completed according to the content of R. Executable memory is a common word or byte addressing memory in size of q. $E[i]$ is referred as the content of the memory cell i, and thus $E = E'$ if and only if for any i that $0 \le i < q$, $E[i] = E'[i]$. No matter what is the current mode of the machine, the relocation bound register keeps active all the time. The relocation part of the register is an absolute address. The limit part b gives the absolute size of the virtual memory. If it is expected to visit all the memory cells, the relocation part l must be set to 0, and the limit part b must be set to $q-1$.

Processor model is either ***s*** (supervisor mode) or ***u*** (user mode). The content of program counter P is an address that is relative to the contents of R, indicating the next instruction to be executed as a pointer of E. The triple $< M, P, R >$ is psw (program status word). In order to facilitate the following discussion, we assume that a psw can be recorded in a memory location, and $E[0]$ and $E[1]$ can be used to store old program state word $old-psw$ and to extract a new state program word $new-psw$, respectively.

Each of the constituent elements of S can only be assigned one of a limited number of values. The finite state set is denoted as C. Then we can conclude that an instruction is a function $i: C \to C$.

Trapping is an important system protection mechanism. A instruction is called being trapped if $i(E_1, M_1, P_1, R_1) = (E_2, M_2, P_2, R_2)$, then for $0 < j < q$,

$E_1[j] = E_2[j]$, and

$E_2[0] = (M_1, P_1, R_1)$,

$(M_2, P_2, R_2) = E_1[1]$.

Hence when a instruction is trapped, there is no change in the whole memory except that the memory location 0 where the psw is stored is becoming effective just before the instruction being trapped. The psw which becomes effective after the instruction being trapped is popped out from the memory location 1.

There is a special trap type which is referred as ***memory trap***. That is, #define memory-trap "if $a + l \ge q \parallel a \ge b$ then trap".

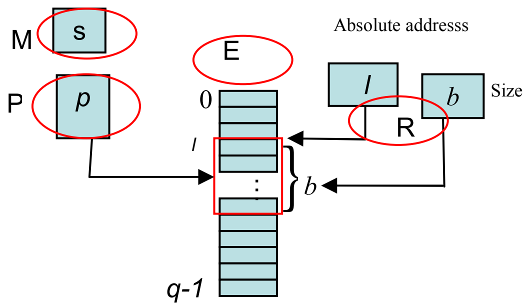

Figure 2. A formal model of modern computer

III. Reclassify ISA based on the Instruction behavior

According to the behavior that a instruction is a function from a state to another state, we can reclassify the instructions in ISA.

Instruction i is a privileged instruction, if and only if for any state $S_1 =< e, s, p, r >$ and $S_2 =< e, u, p, r >$ （here neither $i(S_1)$ or $i(S_2)$ memory trap）, $i(S_2)$ will trap but $i(S_1)$ not.

There is an important set of instructions called as sensitive instructions, and the members of the set are tightly associated with the virtualization capabilities of a machine. Popek have defined two types of sensitive instructions.

An instruction is control sensitive if there is a state $S_1 =< e_1, m_1, p_1, r_1 >$ that $i(S_1) = S_2 =< e_2, m_2, p_2, r_2 >$ to enable $i(S_1)$ does not memory trap, and to enable at least one of the following conditions is satisfied.

$(a)\ r_1 \ne r_2$

$(b)\ m_1 \ne m_2$

Full control of system resources mentioned in the third characteristic of a VMM is necessary. Control sensitive instructions are those which potentially or explicitly affect the system control.

For an integer x, we define an operator $\oplus$ which satisfies $r' = r \oplus x = (l + x, b)$, that is, relocation bound register R has its own reference value which can be changed by x. It is clear that in any specified state the only memory domain that can be accessed are those which have been specified by relocation bound register R.

Since $r = (l, b)$, $E \mid r$ presents the contents of memory cells from l to $l + b$. The token $S =< e \mid r, m, p, r >$ can be used to designate a state.

Intuitively, an instruction is behavior sensitive if its execution effect depends on the values the relocation bound register, i.e. upon its location in real memory, or on the mode. The other two cases, where the relocation bound register or the mode does not match after the instruction is executed, fall into the class of control sensitive instructions.

Popek and Goldberg divide behavior sensitive instruction into two types [3], one is referred as location-sensitive that its execution effect partly depends on its memory location. The other case is referred as mode-sensitive that its execution effect partly depends on the machine mode which it is in.

We consider that the sensitive instructions should be divided into three categories: first are pure location-sensitive instructions; second are pure mode-sensitive instructions; third are the instructions which are both location-sensitive and mode-sensitive. Their formal definitions are respectively given as follows:

Instruction i is a ***pure location-sensitive*** instruction if and only if there are an integer $x \neq 0$ and the following states which satisfy specified conditions:

(a) $S_1 = < e \mid r, m, p, r >, and$

(b) $S_2 = < e \mid r \oplus x, m\ , p, r \oplus x >,$ where

(c) $i(S_1) = < e \mid r, m_1, p_1, r >,$

(d) $i(S_2) = < e \mid r \oplus x, m_2, p_2, r \oplus x >, and$

(e) neither $i(S_1)$ or $i(S_2)$ memory trap,

And at least one of the two conditions must be satisfied:

(f) $e \mid r \neq e \mid r \oplus x, or$

(g) $p_1 \neq p_2$, or both

Instruction i is a ***pure mode-sensitive*** instruction if there are the following states which satisfy specified conditions:

(a) $S_1 = < e \mid r, m_1, p, r >, and$

(b) $S_2 = < e \mid r, m_2, p, r >,$ where

(c) $i(S_1) = < e \mid r, m_1', p_1, r >,$

(d) $i(S_2) = < e \mid r \oplus x, m_2', p_2, r \oplus x >, and$

(e) neither $i(S_1)$ or $i(S_2)$ memory trap,

(f) $m_1 \neq m_2$,

And at least one of the two conditions must be true:

(g) $e \mid r \neq e \mid r \oplus x, or$

(h) $p_1 \neq p_2$, or both

Instruction i is the ***location-mode-sensitive*** instruction (note that it is neither pure location-sensitive nor pure mode-sensitive), if and only if there are an integer $x \neq 0$ and the following states which satisfy specified conditions:

(a) $S_1 = < e \mid r, m_1, p, r >, and$

(b) $S_2 = < e \mid r \oplus x, m_2, p, r \oplus x >,$ where

(c) $i(S_1) = < e \mid r, m_1', p_1, r >,$

(d) $i(S_2) = < e \mid r \oplus x, m_2', p_2, r \oplus x >, and$

(e) neither $i(S_1)$ or $i(S_2)$ memory trap,

(f) $m_1 \neq m_2$,

And at least one of the two conditions must be satisfied:

(g) $e \mid r \neq e \mid r \oplus x, or$

(h) $p_1 \neq p_2$, or both

Popek and Goldberg divided sensitive instructions into behavior sensitive ones and control sensitive ones, and in their views these two subsections are not overlapping with each other[3]. From the point of both logical and practical views, these are imperfect. Control sensitive instructions are those who attempt to influence resource allocation, and behavior sensitive instructions are those whose behavior and execution effect to some extent depends on resource allocation. So in theory, there are instructions which are both control sensitive and behavior sensitive, we might as well call them serious sensitive instructions.

For example, the POPFD instruction in IA-32 architecture is both behavior-sensitive and control sensitive. Its main function is to pop a 32-bit two-word from the stack top, and then to store it into the EFLAGS register. Its execution effect is that all the non-reserved bits except VIP, VIF, VM, are all may be modified. It is control sensitive, because IOPL in EFLAGS register (the bits indicates privilege level which the current process is in when accessing I/O space) are likely to be modified. If POPFD is not designed as a privileged instruction and be executed in the user space, then the VMM's absolute control over I/O resources will be impacted [6]. At the same time, because its interrupt-enable flag can be changed in the privilege mode, but in user mode cannot be changed, we can conclude the instruction is mode sensitive, and so it is behavior sensitive [8].

The concept "serious sensitive instructions" also help to the design of VLIW. Crusoe processor uses 128-bit VLIW instructions which can execute at largest 4 operations in one instruction cycle. If we put a number of conventional control sensitive instructions and behavior sensitive instructions into a VLIW , it will forms a serious sensitive instruction in the logical meaning and similar situations also appear in the Macro-Fusion technology which used in Core architecture for Pentium M processor. Therefore, the discussion of serious sensitive instructions is of not only theoretical significance, but practical significance.

According to a pure mode (pure location, location-mode) sensitive instruction is pure behavior sensitive or serious sensitive, we category it to two types that are ***1-type pure mode (pure location, mode-location) sensitive*** instruction and ***2-type pure mode (pure location, mode-location) sensitive*** instruction.

We refer an instruction that is sensitive, if and only if it is control sensitive or behavior sensitive. If it is not sensitive, then we call that it is ***innocuous.***

Considering the efficiency of the whole machine means to enable the direct implementation of the innocuous instructions as much as possible. Here C denotes a set, and the number of elements in it, i.e. cardinal number, is denoted as $\|C\|$.

The efficiency of the whole machine

$$Eff \propto \frac{\|\text{innocuous instrution set}\|}{\|\text{sensitive instruction set}\|}$$

Suppose the number of instructions in computer instruction set architecture (ISA) is n, that is

$$n = \|\text{ISA}\| = \|\text{innocuous instrution set}\| + \|\text{sensitive instruction set}\|$$

then

$$Eff \propto \frac{n - \|\text{sensitive instruction set}\|}{\|\text{sensitive instruction set}\|} = \frac{n}{\|\text{sensitive instruction set}\|} - 1$$

So we can conclude that the efficiency of the machine is of negative correlation with the proportion of sensitive instructions.

To sum up, based on instruction behavior, the classification results of ISA is shown in Fig. 3(In order to facilitate the expression, "sensitive instructions" is abbreviated as SI).

Now we can analyze VMM in a special and accurate manner.

IV. The corresponding relation between instruction (sequence) before VMM is created and that after VMM is created

First, we discuss the construction of VMM.

VMM=CP=$< D, A, \{v_i\} >$,as is shown in Fig. 4.

The initial instruction of Dispatcher D is placed at the location what the value of P in location 1 indicates and to which the hardware traps. Dispatcher D can be considered as the top-level control module of the control program. It decides which module to call, that is, it may invoke the allocator module or the interpreter modules.

The ***allocator*** A decides what system resource(s) are to be provided. In the case of a single VM, the allocator needs only to keep the VM and the VMM separate. When a VMM hosts several VMs, the allocator must take the responsibility to avoid giving the same resource (such as part of memory) to more than one VM at the same time. Whenever an attempted execution of a privileged instruction in a VM environment occurs which would have the effect of changing the machine resources associated with the environment, the allocator will be invoked by the dispatcher.

For all of the other instructions which trap, the interpreters $\{v_i\}$ are to simulate the effect the instruction which being trapped. There is one interpreter routine per privileged instruction.

The psw in location 1, which is loaded by hardware when a trap occurs, has mode set to *supervisor* and program counter (PC) set to the first location of the dispatcher. Here it is assumed that all other programs will run in user mode.

That is, the psw, which the control program loads as its last

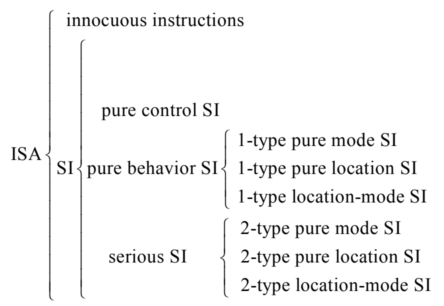

Figure3. Reclassification ISA according to instruction behavior

operation and turns control back to the running program, will have its mode set to *user*. So it is necessary to use a location in the control program to record the simulated mode of the VM.

Partition the collection of all possible machine states C into two parts. The first set C_v contains all those states for which the VMM is present in memory and the value of P in the psw stored in location 1 is equal to the first location of the VMM. The second set C_r contains the remaining states. These two sets reflect respectively the possible states of the real machine with and without a VMM.

A *virtual machine map* (VM map) $f: C_r \to C_v$ is a one-one homomorphism with respect to all the operators e_i in the instruction sequence set. That is, for any state $S_i \in C_r$ and any instruction sequence e_i, there exist an instruction sequence e_i' such that $f(e_i(S_i)) = e_i'(f(S_i))$. Here we demand as part of the definition of a VM map that for each e_i, the appropriated e_i' can be found and executed.

The mapping f should be one-one. To make this mapping concept more precise, Popek and Goldberg have given a particular f, shown as Fig.5. Let the control program occupy the first k locations of the physical memory. That is, $E[0]$ and $E[1]$ are reserved for $pswsā$, and the control program takes locations 2 through $k-1$. The next w locations will be used for a VM. Here it is assumed that $k + w < q$. So $f(E, M, P, R) = (E', M', P', R')$ where $S = < E, M, P, R >$ is the machine state without a VMM present.

Suppose that the value of b in $r = (l, b)$ is always less than w .Then $E'[i+k] = E[i]$,

for $i = 0, w-1$,

E'[i] = the control program, for i= 2 to $k-1$,

$E'[1] = < m', p', r' >$, m'=supervisor,

p'=first location of the control program,

r'=$(0, q-1)$,

$E'[0] = < m, p, r >$ as last set by trap handler,

$M' = u(user)$,

$P' = p'$,

$R' = (l+k, b)$, where $R = (l, b)$

The above is taken as the standard VM map[1].Now the concept "equivalence" or "essentially identical effect" can be expressed more precisely with the help of the above concepts. Suppose the two machines are started, one in state S_1, the other in state $S_1' = f(S_1)$. Then the environment provided by the VMM is equivalent to the real machine if and only if:

for any state S_1, if the real machine halts in state S_2;then the VM halts in state $S_2' = f(S_2)$.

Suppose the mapping between any instruction sequences t when VMM doesn't exist and its counterpart t' when VMM does exist is $Q: t \to t'$, here

$Q(t) = t' \Leftrightarrow \forall S \in C, ft(S) = t' f(S)$,

Then the following propositions come to existence:

(1) if $Q(t) = t', Q(i) = i'$,

then $Q(it) = i't'$.

(2) $Q(i) = \begin{cases} i, & \text{when } i \text{ is innocuous;} \\ \text{InterRoution} \mid i, & \text{when } i \text{ is sensitive.} \end{cases}$

(3) For instruction sequence with any length $t = i_1 i_2 \cdots i_r \cdots i_n (1 \le n)$,

$Q(t) = \begin{cases} t, & t \text{ is an innocuous instruction sequence;} \\ d_1 d_2 \cdots d_r \cdots d_n, & \forall \text{ instruction } d_r (1 \le r \le n), d_r = Q(i) \end{cases}$

That is $Q(t) = Q(i_1) Q(i_2) \cdots Q(i_r) \cdots Q(i_n)$.

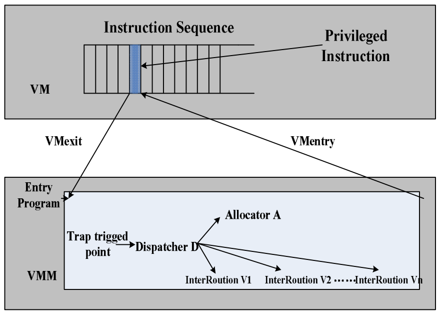

Figure 4. The execution process of a privileged instruction.

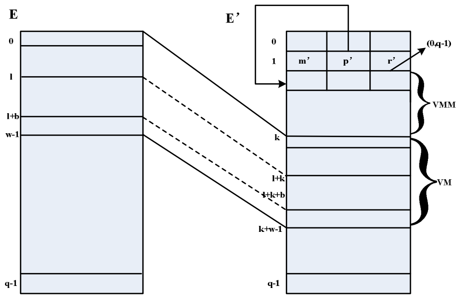

Figure 5. A virtual machine map f

(4) if $Q(t_1) = t_1', Q(t_2) = t_2'$, then $Q(t_1 t_2) = t_1' t_2'$.

The above t_1, t_2 are instruction sequence with any length≥ 1, and the lengths of $i, i_1, i_2, \cdots, i_r, \cdots, i_n$ are all one.

Proof: Suppose S_1 is a any state of the machine.

(1) $\because Q(t) = t'$

$\therefore ft(S_1) = t' f(S_1)$ (*1)

$\therefore i' ft(S_1) = i' t' f(S_1)$ (*2)

Suppose $t(S_1) = S$,

$\because Q(i) = i'$

$\therefore fi(S) = i' f(S)$ (*3)

$\therefore fit(S_1) = i' ft(S_1)$ (*4)

By uniting (*4) and (*2), it is concluded that

$fit(S_1) = i' t' f(S_1)$ (*5)

(*5) is equivalent to $Q(it) = i't'$.Q.E.D.

(2) suppose i is an any innocuous instruction, S is an any real machine state,

and $S' = f(S)$ .here $S = (e \mid r, m, p, r)$, $S' = (e' \mid r', m', p', r')$ .However, according to definition of f, $e' \mid r' = e \mid r$, $p' = p$, and the limit values in r' and r are same. And $i(S)$ cannot rely on m or l (the re-location part of r), all other parameters in S and S' are same. Hence $i(S) = i(S')$ is true. Q.E.D.

(3) Proposition (3) is the generalization result of proposition (2). According to the software architecture of VMM and the return address after the end of the trap, which was shown in Figure 4, we can conclude proposition (3) is true.

(4) $\because Q(t_1) = t_1'$

$\therefore ft_1(S_1) = t_1' f(S_1)$ (*6)

Suppose $t_1(S_1) = S_2$,

$\because Q(t_2) = t_2'$

$\therefore ft_2(S_2) = t_2' f(S_2)$ (*7)

$\therefore ft_2 t_1(S_1) = t_2' ft_1(S_1)$ (*8)

According to (*1), it is concluded that

$t_2' ft_1(S_1) = t_2' t_1' f(S_1)$ (*9)

By uniting (*8) and (*9), it is concluded that

$ft_2 t_1(S_1) = t_2' t_1' f(S_1)$ (*10)

(*10) is equivalent to $Q(t_1 t_2) = t_1 ' t_2 '$. Q.E.D.

V. Seeking a Necessary and Sufficient Condition for ISA Supporting Virtualization

Those conditions that Goldberg proposed prove to be difficult to meet. For example, the IA-32 architecture has 17 “problem instructions” that are extremely sensitive to be running in a virtualized environment, but cannot be trapped. Creative programming however has overcome IA-32s virtualization weaknesses that the VMM must monitor the running virtual machine to insure it does not execute these “problem instructions”, using detecting technology[4]. However, obviously the VMM has to be required to go to these great lengths to virtualize the IA-32 architecture.

The control sensitive instructions especially the serious sensitive instructions, potentially or explicitly affect the system control, so these must be set as privileged ones.

However, under certain conditions, some of the pure behavior SI can fall into the category of innocuous instruction, e.g., some of 1-type pure location SI. MCV may still be achieved that under certain conditions, that is, if it is possible to construct a VMM that resides in high core, letting other program execute without being relocating. In that case, 1-type pure location SI would no matter and so can be fall into innocuous category.

Some of the 1-type pure location SI that is so-called sensitive instructions, may be pulled into the category of the sound instructions.

VI. Conclusions and Future Work

Based on a formal model of modern computer systems, this paper discussed the issue that ISA supporting virtualization. By re-classifying instructions in a new way, we discuss how to pull some of the so-called sensitive instructions into the category of the sound instructions, so as to minimize the proportion of the instructions which are interfered by VMM and interpreted to execute. That is, under the prerequisite of guaranteeing MCV, to exploit in further the potential efficiency of the whole system. We also gives and proves a theorem about the mapping between any instruction sequences t when VMM doesn’t exist and its counterpart t' when VMM does exist. These results help to seek the necessary and sufficient conditions of ISA supporting virtualization, can contribute to the design of ISA supporting MCV, as well as the construction of high-efficiency VMM. Based on the reclassification, to seek and give explicitly a necessary and sufficient condition for ISA supporting virtualization is the further research direction.

Acknowledgment

This work is supported by National High Technology Research and Development Program (“863”)of China under the research project 2006AA01Z108 and 2007AA01A127.

References

[1] R. P. Goldberg. Survey of Virtual Machine Research. Computer,pages 34–45, June 1974.

[2] M. Rosenblum and T. Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. IEEE Computer, pages 39–47, May 2005.

[3] G.Popek and R.Goldberg. Formal Requirements for Virtualizable 3rd Generation Architectures. Communications of the A.C.M,17(7):412-421,1974

[4] J.S. Robin and C.E. Irvine, "Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor," *Proc. 9th Usenix Security Symp.,* Usenix, 2000, pp. 129-144.

[5] Gagliardi, U.O., and Goldberg, R.P. Virtualizable architectures, Proc. ACM AICA lnternat. Computing Symposium, Venice, Italy, 1972.

[6] IA-32 Intel® Architecture Software Developer’s Manual Volume 2B,4-111,2007,Intel Corporation.

[7] Galley, S.W. PDP-10 Virtual machines. Proc.ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, Cambridge, Mass., 1969.

[8] James E. Smith, Ravi Nair. 2006. 7. Virtual Machines: Versatile Platforms for Systems and Processes, pp. 385. Printed in China by Publishing House of Electronics Industry, under special arrangement with Elsevier (Singapore) Pte Ltd.

[9] IBM Corporation. IBM Virtual Machine Facility/370: Planning Guide, Pub. No.GC20-1801-0, 1972.

[10] Lauer, H.C., and Snow, C.R. Is supervisor-state necessary? Proc. ACM AICA Internet. Computing Symposium, Venice, Italy, 1972.

[11] Hugh C. Lauer , David Wyeth, A recursive virtual machine architecture, Proceedings of the workshop on virtual computer systems, p.113-116, March 26-27, 1973, Cambridge, USA.

[12] Meyer, R.A., and Seawright, L.H. A virtual machine time sharing system. IBM Systems J. 9, 3(1970)