THE UT VIRTUAL MACHINE MONITOR

by

David J. Renaud

May 1978

TR-78
SPL/5

Systems Programming Laboratory Note 5

This work was supported in part by the Department of the Navy, Naval Electronic Systems Command, under Contract N00039-78-G-0080, and by the National Science Foundation under Grant MCS75-16425.

DEPARTMENT OF COMPUTER SCIENCES

THE UNIVERSITY OF TEXAS AT AUSTIN

ABSTRACT

This report demonstrates the feasibility of sharing a Nova 3/D minicomputer at the level of the basic machine interface (hardware/firmware environment). Sharing is made possible by the creation of a privileged software nucleus called a virtual machine monitor. Virtualization of the Nova is shown at three levels: theory, design, and implementation.

Chapter 1 defines the concept of a virtual machine monitor, demonstrates the theoretical feasibility of virtualization, discusses the history and utility of virtualization, and specifies the Nova 3/D virtual machine. Chapter 2 discusses initial design considerations, describes the top level decomposition of the monitor, discusses the notion of virtual processor states and their transitions, and describes some monitor components referenced in later decomposition. Chapter 3 describes the decomposition of a single top level component, the trap process, which handles all real machine traps. Chapter 4 describes the decomposition of all other top level components.

Chapter 5 concerns implementation. It describes the systems programming language used, describes needed systems variables and tables, and explains the function of all required programs.

Three appendices are provided. The first describes how to run a systems program on the Nova using the facilities provided by RDOS (real time disk operating system). The second demonstrates how to run the UT virtual machine monitor. The third presents the monitor command language syntax.

This report is a revision of a Master's Thesis of the same title.

Table of Contents

CHAPTER ONE

Prospects for a Virtualized Nova

1.1 Introduction.

This chapter explores the possibility of virtualizing a Nova 3/D (with Memory Management and Protection Unit). First, the notion of a virtual machine monitor is defined and a demonstration of the theoretical feasibility of virtualizing a Nova is presented. Next, a discussion of the history and utility of virtualization is given. Finally, a specification for the Nova 3/D virtual machine is stated.

1.2 Definition And Function Of A Virtual Machine Monitor.

A _virtual machine monitor_ (VMM) is a privileged software nucleus which creates copies of the basic machine interface on which it runs [4]. These copies are called _virtual machines_. A basic machine interface is defined by Buzen and Gagliardi [4] as "the set of all software visible objects and instructions that are directly supported by the hardware and firmware of a particular system." Figure 1 depicts this virtual machine architecture.

In order to distinguish VMMs from system software with similar functionality, several authors [2] [15] have cited three defining requirements. These are:

1. The Equivalence Requirement. Virtual Machines must be logically equivalent to real machines. Software designed for execution on the real machine must also execute on the virtual machine and produce equivalent results with the exception of timing considerations.

2. The Efficiency Requirement. The performance of a virtual machine should be degraded only by the need for resource sharing. Efficiency is a primary goal of VMMs; all instructions which may be safely interpreted in hardware (firmware) must be.

3. The Resource Control Requirement. The VMM must control allocation of all system resources (processors, memory, devices) and enforce isolation between virtual machines. A virtual machine may not interfere with another virtual machine or the VMM in any way, except as stated by the Efficiency Requirement.

A virtual machine monitor may also be defined as a collection of processes which create virtual processors and virtual devices. This definition serves to make two points. First, a VMM need not be a single

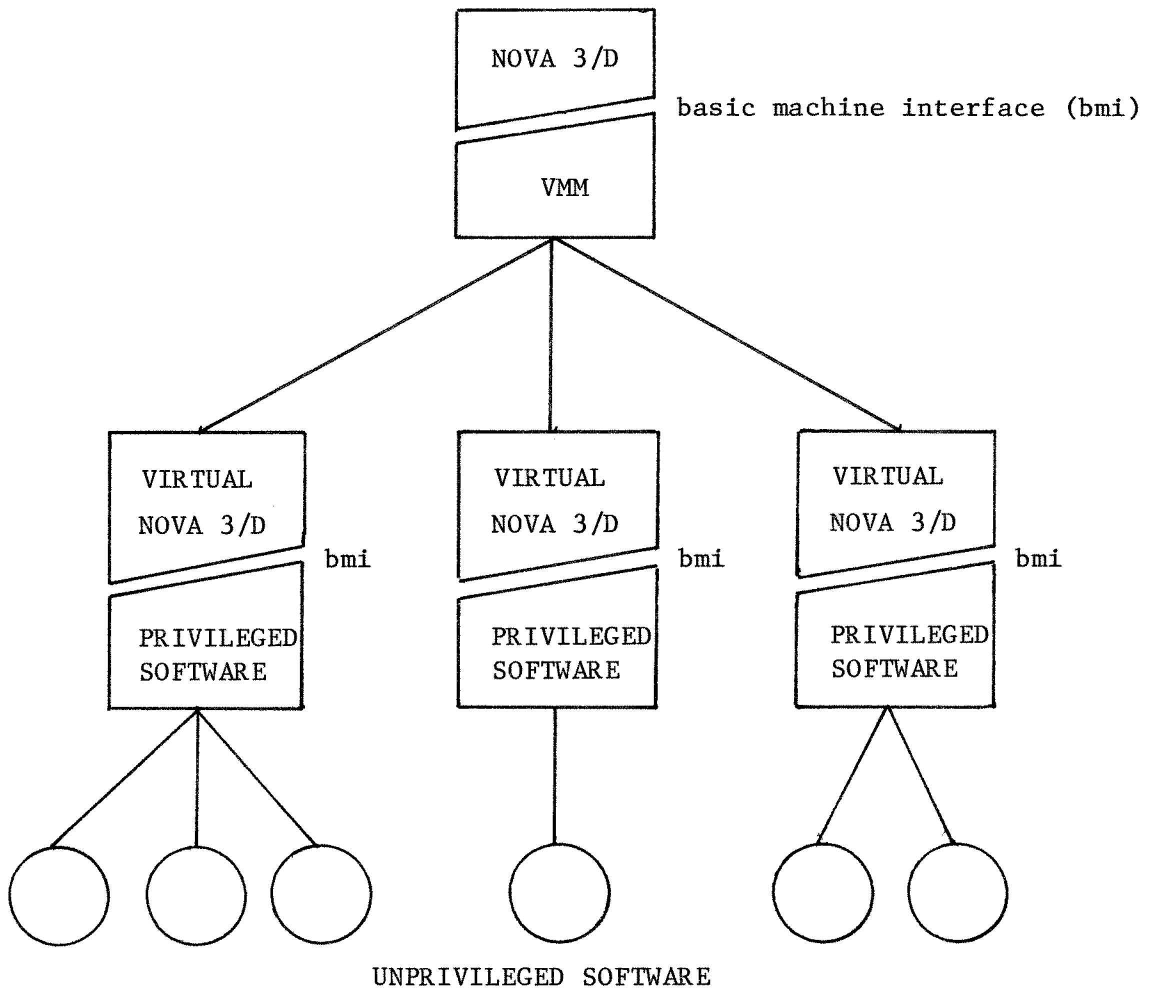


FIGURE 1

Nova 3/D Virtual Machine Architecure

sequential computation; rather, it is a collection of independent computations which could execute concurrently. Second, the notion of virtual machine may be decomposed into the notions of virtual processor and virtual device.

In order to determine the precise function of a virtual machine monitor, it is helpful to consider the relation between real and virtual processors (devices). A real processor (of the type this report concerns) is composed of two kinds of resources: _actions_ and _store_. The _store_ serves to hold machine state and consists of main memory and special registers. _Actions_ cause state to state transitions and include facilities such as instruction interpretation, interrupts, and traps. The relation between real and virtual processors is indicated with respect to these two resources.

The store of a virtual processor is logically equivalent to the store of a real processor in that it defines an equivalent state space. However, there are physical differences. Special registers of the virtual processor are not necessarily supported by real special registers; rather, they may be maintained as virtual special registers in the main memory of a real processor. Likewise, the main memory of a virtual processor may be maintained as a virtual memory on a backing store such as a disk or in real main memory. In order for the actions of the real processor to cause state to state transitions on the virtual store, it is necessary to copy portions of the virtual store into their logical equivalents in the real store. When this occurs, those portions of the virtual store that have been copied in are said to be _realized_ or _mapped_.

Actions of a virtual processor consist of those actions of the real processor which cause state transitions on the realized virtual store as well as actions, supported by the VMM, which cause transitions in the virtual store.

Real devices are also composed of actions and store. A _device store_ is a set of registers visible to software running on the real processor which controls the device and holds device state. In this context, memory associated with a device is not considered part of the store. _Device actions_ cause state transitions on the device store corresponding to the operation of a real device. A virtual device store may be directly supported by the real store or may be maintained in main memory of the real processor. In both cases, access to the virtual device store by software running on a virtual processor constitutes a virtual processor action supported by the VMM. Virtual device actions are also supported by the VMM and correspond to the operation of a virtual device, which is mapped to a real device or simulated.

From the previous discussion, it is apparent that a virtual machine monitor must function in two fundamental ways. First, it must allocate the resources of the real machine. This means creating and maintaining a virtual store for each virtual processor, creating virtual devices, mapping virtual devices to real devices, and assigning the real processor(s) to virtual processor. Second, the VMM must support actions of the virtual processors and devices which cause transitions directly in the virtual store. This includes interpretation of some instructions, simulation of traps and interrupts, and simulation of the actions of virtual devices.

1.3 Feasibility Of Nova 3/D Virtualization.

Intuitively, a real machine is virtualizable if all instructions which may potentially disrupt the system can be arrested before execution and if the state of the real processor before the arrest can be recovered. In this way, a virtual processor which is assigned the real processor may be prevented from interfering with other virtual processors or the VMM. To demonstrate feasibility, it is first necessary to indicate which instructions are disruptive. A formal model of Popek and Goldberg [15] is presented which identifies potentially disruptive instructions and defines a formal requirement for virtualization which the instruction set must meet. Next, the architecture of the Nova 3/D is reviewed in the context of the formal model. The Nova 3/D instruction set is then inspected; it is shown that the Nova 3/D nearly satisfies the formal requirement and that problems may be overcome with only a small loss of equivalence. Finally, it is shown that the Nova 3/D provides for the recovery of real processor state after a potentially disruptive instruction is arrested.

1.3.1 Formal Model. -

The model of Popek and Goldberg concerns machines having address translation hardware and more than one mode of operation. Address translation is a prerequisite for virtualization since a virtual processor requires a logical address space equivalent to that of a real processor. Logical addresses are those generated by a running program. Modes provide a mechanism for partitioning the instruction set. The requirement for virtualization suggested by the model concerns the manner in which the instruction set is partitioned.

According to Popek and Goldberg [15], machine state is specified by the following four components.

1. Executable Memory (E). The portion of the store used to hold data or addresses (logical addresses or displacements) needed to access data. This portion includes main memory, accumulators, index registers, carry bits, link bits, and stack pointers.

2. Program Counter (P). The register which holds the logical address of the next instruction.

3. Relocation Registers (R). The registers which implement the mapping of logical to physical addresses. The activation of this mapping may may depend on processor mode.

4. Mode Register (M). The register which indicates the current processor mode. In one mode (named supervisor, master, executive, kernel), all machine instructions may be executed. In the other mode(s) (named user, slave, program, unprivileged), only a subset of the machine instructions are available.

In the Popek and Goldberg model, special registers associated with devices and the CPU were excluded for simplicity. They will be included here for completeness. To do so does not diminish the model's validity. Hence, a fifth component of state is:

5. Special Registers (I). The registers of the store used to control the processor, operate devices, and buffer data between processor and devices. These registers include interrupt system control registers, device control registers, and input/output data buffers. They may also include other CPU special registers such as the on/off switch, data switches, and protection control registers.

In summary, the state of a computer is specified by the five tuple

$$S = \langle E, P, M, R, I \rangle.$$

Before proceeding to the requirement for virtualization, several definitions are required. An instruction is said to <u>trap</u> when

1. executable memory is not altered with the exception of status information storage, and

2. the program counter is redefined and processor mode is changed to supervisor.

Status information is normally defined as the address of the instruction causing the trap, the address which actually caused the trap, and the condition which caused the trap. An instruction is said to be <u>privileged</u> if it executes in supervisor mode and traps in user mode. An instruction is said to be <u>sensitive</u> if it is either <u>control sensitive</u> or <u>behavior sensitive</u>. Control sensitive instructions are those which alter M, R, or I. Instructions are behavior sensitive if their execution is dependent upon their location in physical memory or upon processor mode. An instruction which is not sensitive is said to be <u>innocuous</u>.

Assume a computer exists whose state is specified by the above five components and which has both innocuous and sensitive instructions, some of which are privileged. Is this machine virtualizable? Popek and Goldberg have shown that it is only if the set of sensitive instructions is a subset of the privileged instructions. Informally, this means that user mode programs cannot issue halt commands, change mode except through traps, alter relocation registers, control devices, or alter any component of I. An attempt to do so should always result in a trap.

1.3.2 Architecture Of The Nova 3/D. -

In the context of the formal model, the architecture of the Nova 3/D may be reviewed. For a complete description of the hardware, refer to Data General manuals [5], [6], and [7].

1. Executable Memory (E). The Nova 3/D has 32K (expandable to 128K) sixteen bit words in main memory and four sixteen bit accumulators. Accumulators two and three may be used as index registers. Two fifteen bit registers, the stack and frame pointers, are provided for stack operations. Locations 20 to 27 (octal) are auto-incrementing when addressed indirectly. Similarly, locations 30 to 37 are auto-decrementing. A carry bit is also provided.

2. Program Counter (P). The Nova 3/D has a fifteen bit program counter.

3. Relocation Registers (R). The Memory Management and Protection Unit (MMPU) provides four sets (maps) of thirty-two registers which implement translation of logical page addresses to physical page addresses. Page size is 1024 words. Two maps are dedicated to user programs and two are used by the data channel. Map selection for user programs is under program control. When a map is enabled, the high order five bits (logical page address) of a memory address are used as an index into the map. These five bits are replaced by seven bits (physical page address) from the selected map register. Only one program map may be enabled at a time; activation of data channel maps are independent from that of user maps. When mapping is disabled, logical addresses are equal to physical addresses.

4. Mode (M). The Nova 3/D operates in two modes, supervisor and user. Whenever program mapping is enabled, the processor is in user mode; otherwise, it is in supervisor mode. Mode changes from supervisor to user when appropriate bits in a status register are set and a defer (indirect address) cycle is executed. Transition from user to supervisor mode occurs in response to a trap or interrupt. The MMPU (MAP) busy flag serves to indicate mode.

5. Special Registers (I). Special registers of I/O devices are busy flags, done flags, interrupt disable flags, and I/O buffers, which may serve as control registers or data buffers. Setting a busy flag to one and done to zero normally starts a device and permits an interrupt to occur upon completion. Device completion normally sets done to one and busy to zero. The device interrupt disable flags make possible a priority interrupt scheme having sixteen levels.

Special registers of the central processor are the on/off switch, the interrupt enable flag (CPU busy flag), the power fail flag (CPU done flag), data switches, and the interrupt request line, which holds the addresses of all devices requesting interrupts. Special registers of the MMPU other than the relocation registers are the status register, which contains mode, protection, and map selection flags; the violation data register, which indicates the condition(s) causing the last trap; the violation address register, which indicates the logical address of the instruction causing the trap, the page check register, which is used to determine the contents of a map register; and the MMPU done flag, which indicates a protection violation during data channel mapping.

Protection flags of the status word are used to enable traps when user mode programs execute I/O instructions, use auto-increment/decrement locations, write or access protected pages, or execute more than fifteen consecutive defer cycles. Write and access protection is specified on a page by page basis by setting bits in the relocation registers. Auto location protection permits instructions, which indirectly address logical addresses 20 to 37 (octal), to be trapped.

1.3.3 Feasibility Demonstration. -

Having reviewed the Nova 3/D architecture, it is appropriate to examine the instruction set, identify sensitive instructions, and verify that they are privileged. Should all sensitive instructions be privileged, then the formal requirement will be met and one aspect of feasibility will be demonstrated. The Nova 3/D with MMPU has seven classes of instructions: memory reference, arithmetic and logical, stack, trap, device I/O, CPU control, and MMPU control. The latter two classes are forms of the device I/O instruction, however, it is useful to consider them separately. In the following discussion, the assembly language mnemonic for each instruction is given in parentheses.

1.3.3.1 Memory Reference Instructions - This class includes those instructions which reference main memory and accumulators (E). They are:

1. load accumulator (LDA).

2. store accumulator (STA).

3. increment memory and skip on zero (ISZ).

4. decrement memory and skip on zero (DSZ).

5. jump (JMP); load program counter from memory.

6. jump to subroutine (JSR); store old program counter in accumulator three and load program counter from memory.

Because only eight bits are available for a memory address, addressing is done either directly to the first 256 words of memory or relative to the program counter or an index register. Relative addressing ranges between a minus 128 or plus 127 word displacement. Indirect addressing is also provided.

Since memory reference instructions do not alter M, R, or I, they are not control sensitive. If they do not indirectly address the auto increment/decrement registers, they are also not behavior sensitive and, hence, are innocuous. Indirectly addressing the auto locations constitutes a behavior sensitive instruction because the outcome depends upon mode. In supervisor mode, the results are as expected. However, in user mode, proper incrementing or decrementing will not occur if logical addresses 20 to 37 (octal) are not mapped to physical addresses 20 to 37. In this case, the sensitive instructions are also privileged because the MMPU can trap attempts to access these locations indirectly while in user mode.

1.3.3.2 Arithmetic And Logical Instructions - Operations on the accumulators, the carry bit, and the program counter, display no behavior sensitivity, and are, therefore, innocuous. They include:

1. formation of one's complement (COM).

2. formation of two's complement (NEG).

3. transfer between accumulators (MOV).

4. increment accumulator (INC).

5. addition (ADD).

6. subtraction (SUB).

7. addition to one's complement (ADC).

8. logical and (AND).

In a single instruction, the source and destination accumulators are specified and the initial value of the carry bit is given. Also, options are available for shifting one place to the right or left, swapping bytes (high or low order eight bits), not loading the results, and skipping the next instruction conditionally or unconditionally.

The Nova 3/D also provides two additional instructions. These are:

1. integer multiplication (MUL).

2. integer division (DIV).

1.3.3.3 Stack Instructions - These instructions access only the stack pointer, the frame pointer, accumulators, and main memory. They display no behavior sensitivity and are, hence, innocuous. The stack pointer points to the top of a stack maintained in main memory. The frame pointer points to the last return block pushed. A return block consists of accumulators zero, one, and two, the previous frame pointer, the carry bit, and accumulator three (old program counter stored by JSR instruction). The stack instructions are:

1. move to stack pointer from accumulator (MTSP).

2. move to frame pointer from accumulator (MTFP).

3. move from stack pointer (MFSP).

4. move from frame pointer (MFFP).

5. push accumulator (PSHA).

6. pop accumulator (POPA).

7. push a return block (SAV).

8. pop a return block (RET).

1.3.3.4 The Trap Instruction. - The trap instruction allows explicit trap by a user mode program. It is control sensitive since it alters M, however, it is also privileged since a trap results. The trap instruction may specify one of 128 trap numbers. It is of the form:

1. trap (TRAP).

1.3.3.5 I/O Instructions - These instructions are of two types. The first permits transfer of data and control pulses between the processor and I/O devices. A single instruction specifies a device address, an accumulator, a data buffer of the device, the direction of transfer, and a control pulse. The latter concerns setting and clearing the busy and done flags which control device operation. The first type of I/O instructions are:

1. no operation (NIO).
2. data in (DIA, DIB, DIC).
3. data out (DOA, DOB, DOC).

The no operation command may also be used with I/O instructions addressing the CPU or MMPU. In the data in and data out instructions, the A, B, and C variations refer to data buffers. The control commands start (S), clear (C), and pulse (P) may be used with any I/O instruction. Depending on the device addressed, the interpretation of an I/O command is different. Because I/O commands can alter I, they are control sensitive. However, they are also privileged since the MMPU can trap I/O instructions while in user mode.

The second type of I/O instructions are those which alter the program counter based on the state of the busy or done flags of I/O devices, the CPU, and the MMPU. For the moment, instructions addressing the CPU and the MMPU are considered with those addressing devices for convenience. Strictly speaking, these instructions are not sensitive as they only alter P and cannot change I. Nonetheless, they are privileged since the MMPU can trap I/O instructions. These instructions are:

1. skip if done is zero (SKPDZ).
2. skip if done is one (SKPDN).
3. skip if busy is zero (SKPBZ).
4. skip if busy is one (SKPBN).

1.3.3.6 CPU Instructions - These are I/O instructions addressing the CPU that serve to control the interrupt system, read data switches, and halt the machine. They are sensitive since they alter I and privileged as the MMPU traps I/O instructions. These instructions are:

1. read switches (DIA).
2. interrupt acknowledge (DIB); fetch address of the closest device on the interrupt request line which is requesting an interrupt.
3. mask out (DOB); set selected interrupt disable flags to one.
4. clear I/O devices (DIC); clear all busy, done, and interrupt disable flags.
5. halt (DOB).

If the control command S is issued with any instruction addressing the CPU, the CPU busy flag is set to one and interrupts are enabled. Similarly, if a C command is issued, the busy flag is cleared and interrupts are disabled. The done flag (power fail indicator) is also cleared.

1.3.3.7 MMPU Control Instructions - These are I/O instructions addressing MAP and MAP1, the devices which correspond to the MMPU. They reference the relocation registers, the status register, the violation data register, the violation address register, and the page check register and serve to control the address translation and protection facilities. Since they reference R and I, they are sensitive, however, they are also privileged as they are I/O instructions. These instructions are:

1. load map (DOB MAP).
2. initiate a page check (DOA MAP1).
3. page check (DIA MAP1).
4. read MMPU status register (DIA MAP).
5. write MMPU status register (DOA MAP).
6. read violation data register (DIB MAP).
7. read violation address register (DIB MAP1).

If a C control command is issued with an instruction addressing MAP, the violation data register and the MMPU busy and done flags are cleared. If a P is issued, mapping is enabled for the next data fetch. IF a C is issued with an instruction addressing MAP1, all internal MMPU logic is initialized.

One instruction not covered explicitly by the MMPU commands is mode change from supervisor to user. To change modes, the program map enable flag (bit 0) of the MMPU status register is set to one and the mapping inhibit flag (bit 2) is set to zero. Mode change occurs on the next defer (indirect address) cycle, which should be a jump indirect through a user program counter. The jump instruction is sensitive as it alters M, however, it is not privileged.

1.3.4 Summary. -

The preceeding review of all instructions has shown that the virtualization requirement of Popek and Goldberg is almost satisfied. Only the jump to user mode instruction creates a virtualization problem. If this problem can be overcome, the first aspect of feasibility is demonstrated.

1.3.5 Solutions To The Virtualization Problem. -

There are several solutions to the virtualization problem created by the mode switch instruction. First, the VMM could simply fail to support the virtual MMPU facility. This is extremely easy to implement but would result in an unacceptable loss in equivalence. Second, every instruction following the status word change could be interpreted by the VMM. No alterations in real processor hardware or virtual processor software would be required, however, the VMM could become large and complex. Third, the real processor could be altered to trap on defer cycles. No changes in virtual processor software would be required; however, the impact on other operating systems using the same real processor would be unpredictable. Finally, virtual processor software could be required to issue a special explicit trap instruction immediately before the jump to user mode. No hardware changes would be required; however, a small loss of equivalence to the real processor would exist.

Of these four solutions, the last is chosen due to its relative ease of implementation and minor loss of equivalence. Existing programs may be altered with small complication since defer cycles following status word changes may be easily detected. Trap number 127 is used to indicate mode switch and should not be used by virtual processor software for any other reason. This number is selected because it is the largest and perhaps the least likely to be used by virtual processor software.

1.3.6 State Restoration Verification. -

In order to give a complete demonstration of virtualization feasibility, it is necessary to show that the state of the processor before the trap may be recovered after the trap. Relevant processor state is defined by the program counter, the carry bit, the accumulators, the stack pointer, and the frame pointer. Traps correspond to violations due to I/O protection, auto location protection, access (validity) protection, write protection, and defer protection. In all cases, the program counter prior to the trap is saved in the MMPU violation address register.

Whenever a trap occurs, the instruction causing the trap does not execute, with one exception. An instruction which does not execute cannot change the relevant processor state other than the program counter, hence, the state before the trap is recoverable.

The exception concerns the SAV and RET stack instructions. In all stack operations, the stack and frame pointer are not updated until the operation is completed, thus, there is no danger of losing these pointers if a write or validity protection occurs during the operation. However, accumulators and main memory may be altered corresponding to the progress of the operation before the trap. This is of no consequence since only the program counter is required to restart the instruction. Upon restart, the altered components will have their values recopied.

1.4 History Of Virtualization.

According to Buzen and Gagliardi [4], the use of I/O processors and multiprogramming in the early 1960s created "very serious potential problems for system integrity." The creation of dual state architectures was a step towards solving these problems. Software could then be run in two modes: one which allowed access to all machine facilities and one which allowed only nondisruptive instructions to be executed. Only a privileged software nucleus which created an "extended" machine was allowed to run in the privileged mode. This solution created further problems. Programs designed to run on an extended machine could be transported only to facilities having an identical extended machine. More than one copy of privileged software could not be run concurrently, precluding development and modification of privileged software without a dedicated machine. Also, hardware test and diagnostic software could not be run concurrently with privileged software.

A method was needed for sharing a computer at the lowest level, the basic machine interface. If a single computer presents several basic machine interfaces which are completely isolated from one another, the above problems are solved. Transported programs may run on the appropriate extended machine concurrently with programs running on different extended machines. Similarly, programs designed to run on the basic machine interface may be run concurrently.

IBM began development of virtual machine systems in 1964 with the creation of CP-40 (Control Program 40), which ran on a System/360 model 40 modified to support virtual storage [11]. The virtual machine created by CP-40 did not support address translation. Later, CP-67 emerged, running on a System/360 model 67. This system did support virtual machines having virtual address translation. A single user operating system, CMS (Cambridge Monitor System), was developed concurrently to extend the created basic machine interfaces.

According to Meyer and Seawright [10], IBM's objectives in this effort were to research timesharing methods, to examine hardware requirements for timesharing, to develop an in-house timesharing system, and to develop performance analysis techniques. A later IBM development was the VM/370 system which created virtual 370s capable of supporting all the System/360 and System/370 operating systems [8]. IBM also developed the virtual machine like system M44/44X, which ran on a modified 7044, and the System 360/30, a single virtual machine system.

Other virtual machine systems are the Michigan Terminal System (MTS), which supports virtual 360s and runs on the System/360 model 67, the PDP 10 system, which runs on a modified PDP 10 at MIT [8], the HITAC 8400 system, which runs on a HITAC 8400 (RCA Spectra 70/45) [8], and the UCLA VM system, which runs on a modified PDP 11/45 [16].

If virtual machine methods are used to implement timesharing systems, sharing of data between users is prevented. This is because the virtual machine monitor is aware of all users but has no knowledge of any file structures. Conversely, user operating systems know about file structures but are unaware of each other. Several researchers [1] have developed techniques to permit data sharing in VM/370.

All virtual machine systems mentioned above, as well as the one described in this paper, use traps and simulation to support virtual machines. Goldberg [8] has pointed out that these methods are "clumsy and awkward." He proposes that hardware virtualizers be used to support virtual machines. These are hardware/firmware devices which provide a mapping function between virtual resources and real resources. The hardware virtualizer must store maps, activate virtual machines, compose maps, and pass control to the VMM after a map fault. A complete description of this concept is given in [9].

1.5 Utility Of A Virtual Machine Monitor.

Several authors [3] [8] [10] [11] [16] have enumerated the uses of virtual machine systems. These ideas are summarized below. Virtual machine systems provide the capacity

1. to run several operating systems concurrently,
2. to run several stand alone systems concurrently,
3. to develop and maintain operating systems, diagnostic software, or privileged software without taking over the machine,
4. to add hardware enhancement to the real machine without updating existing operating systems,
5. to prevent erroneous systems programs from crashing the machine,
6. to create a high degree of data security between users,
7. to insert software hooks which measure system performance,
8. to give students access to a basic machine environment in an inexpensive way,
9. to run a virtual configuration different from the real configuration.

This last use is of particular interest in a research environment. Virtual configurations are limited by the imagination of the VMM designer. For example, several virtual processors sharing memory could be created. Also, non-existing I/O devices or interprocessor links could be simulated.

1.6 Specification Of A Nova 3/D Virtual Machine.

Before the design of a virtual machine monitor can begin, it is necessary to give a specification for the created virtual machines. This specification describes objects visible to software running on a virtual machine as well as the operation of virtual actions.

1.6.1 The Virtual Store. -

The virtual store consists of the same components as the real store. Virtual main memory is restricted to thirty-two pages or 32,768 words. Store components include accumulators, stack pointer, frame pointer, carry bit, CPU busy and done flags, interrupt request line, and data switches. Store associated with devices and the MMPU is considered separately.

1.6.2 Virtual Instructions. -

All instructions which may be executed on the real machine may be executed on the virtual machine, with one exceptions. A jump to user mode instruction will not execute properly unless it is immediately proceeded by an explicit trap with trap number set to 127.

1.6.3 The Virtual MMPU. -

The virtual store of the MMPU consists of the same components as the real MMPU. There are the two program maps, the two data channel maps, a status register, a violation address register, a violation data register, a mode switch (busy flag), and a data channel error flag (done flag). Virtual memory management, that is, address translation directed by virtual maps, is supported. All five protection features (write, validity, auto location, I/O, and defer) are also supported. Virtual violations occur on virtual machines just as real violations would occur on real machines. The consequences of virtual violations are also equivalent.

1.6.4 Virtual Devices. -

The following virtual devices are available. For each virtual device, the real device which supports it is indicated.

1. card reader (CDR): supported by real card reader
2. line printer (LPT): supported by real line printer
3. teletype (TTI and TTO): supported by CRT which is connected to a multiplexer (QTY)
4. second teletype (TTI1 and TTO1): supported by the real first teletype (TTO and TTI)
5. 6030 diskette (DKP: drives 1 and 2): supported by real 6030 diskette (DKP: drives 1 and 2)

6. 4234 moving head disk (DKP: drive 0): supported by real 4234 moving head disk (DKP: drive 0)

7. real time clock (RTC): supported by real machine real time clock

Figure 2 depicts the mapping between virtual and real devices. The virtual store associated with each device corresponds to the store of the real device. This includes busy and done flags, interrupt disable flags, and device registers. In most cases, the operation of a virtual device is equivalent to that of the real device. Exceptions are the virtual card reader and the virtual 4234 moving head disk. Unlike the real card reader, the virtual card reader cannot lose data. Also, the virtual 4234 moving head disk has fewer cylinders than the real disk and cannot achieve all the real disk error states.

The teletype device configuration of a real Nova permits only two teletypes to be connected directly to device lines. Additional teletypes (CRTs) must be attached to a multiplexer (QTY) which uses a single device line. This organization is reflected in the configuration of virtual teletypes. A virtual multiplexer is not specified due to a lack of CRTs in the UT Nova configuration. Similarly, other peripherals (such as tape drives), which exist but are not present in the UT Nova configuration, are not specified.

1.6.5 Virtual Interrupts. -

Virtual interrupts occur in virtual machines exactly as real interrupts occur on real machines, assuming real processor and device states are equivalent. The consequences of virtual interrupts are also equivalent.

1.6.6 Virtual Stack Faults. -

A virtual stack fault occurs when virtual interrupts are enabled and the virtual stack pointer is set to a multiple of 256 during a PSHA, POPA, SAV, or RET stack instruction. This is equivalent to a real machine stack fault. The consequences of a stack fault are also equivalent.

1.6.7 Virtual Front Panel Switches. -

The functions of virtual front panel switches are provided through the command language used at the VMM operator's console. These functions are equivalent to those of the Nova 3/D with one exception. The memory reference functions are restricted to the first 1024 words of memory to prevent the possibility of a page fault. Front panel commands are fully described in Chapter 4.

VIRTUAL DEVICES	REAL DEVICES
CDR	CDR
LPT	LPT
TTI/TTO	QTY
DKP (all drives)	DKP (all drives)
TTI1/TTO1	TTI/TTO
RTC	RTC

FIGURE 2

Mapping of Virtual Devices to Real Devices

CHAPTER TWO

Design of a Virtual Machine Monitor for the Nova 3/D

2.1 Introduction.

The design of a virtual machine monitor is the creation of its structure and form. The goal of our design is to produce a VMM structure which is easy to understand and which suggests a straightforward implementation. Our main concern is with issues which are directly related to virtualization.

The method used for creating VMM structure is top down decomposition. The monitor is decomposed in an iterative fashion until an implementation is suggested.

There are two basic issues at the top level of virtual machine monitor design. The first issue concerns the initial decomposition of the monitor into computations, which we call processes. The meaning of the term "process" in this context is stated below. The mechanism of process scheduling is also considered as well as the problem of indivisible operation on shared data objects.

The second issue concerns virtual processors. In order to understand how the decomposition is to proceed, virtual processor states are identified and transitions between states are described. In this chapter, these two issues are discussed. The decomposition of VMM processes is described in Chapters 3 and 4.

2.2 Virtual Machine Monitor Processes.

The notion of concurrent processes used in this report is taken from Brinch Hansen [3]. Essentially, a process is "a sequence of operations carried out one at a time." Processes are concurrent if their executions overlap (or interleave) in time. In this report, processes are our unit of decomposition, serving to organize our system into logical components. Processes are the entities which are visibly scheduled for execution, either directly or by interrupts or traps.

2.2.1 Identification And Function Of VMM Processes. -

The following VMM processes may be identified. There is one process associated with each real device, one associated with real processor traps, one associated with stack faults, an initialization process, and a dispatcher process.

The process associated with the multiplexer (QTY) supports virtual actions for virtual teletype devices (TTI and TTO). The real time clock (RTC) process supports virtual actions for virtual clocks, provides an alarm clock function for signaling the end of virtual processor time slices, and provides a system clock. The disk (DKP) process supports virtual actions for the virtual 6030 diskette and virtual 4234 moving head disk. It also assists the support of page fault handling and command line interpretation (virtual processor bootstrap loading). The line printer (LPT) and card reader (CDR) processes support virtual actions for these virtual devices. The teletype (TTI and TTO) processes support virtual actions for the virtual second teletype device (TTI1 and TTO1). Finally, the processes associated with the second console (TTI1 and TTO1) provide console communication, system generation, and command line interpretation. Each of the device processes has the additional function of saving and restoring real machine state upon entry and exit and supporting the virtual interrupt and stack fault facilities.

The process associated with traps supports the majority of virtual processor actions which operate directly on the virtual store. This process handles page faults, interprets sensitive instructions, supports the virtual auto increment/decrement facility, provides virtual memory management and protection, simulates virtual traps, saves and restores real machine state upon entry and exit, and simulates virtual interrupts.

The initialization process exists out of necessity and should not be considered a true process of the VMM. It is active only when the monitor is started and cannot run in parallel with any other process.

The process associated with stack faults supports the virtual stack fault facility and handles faults which occur during VMM process execution. It saves and restores real machine state upon entry and exit and simulates virtual stack faults.

The dispatcher process serves two functions. When no virtual processors are ready to take over the real processor, it serves as an idling process and merely waits for an interrupt. If at least one virtual processor is ready, it composes an address translation map, realizes a portion of the virtual store, and assigns the real processor to the virtual one, putting itself to sleep. The dispatcher process may also initiate simulation of virtual interrupts and stack faults.

2.2.2 Process Scheduling. -

Virtual machine monitor processes are scheduled in three ways. First, all device processes and the stack fault process are scheduled by interrupts. This means that, potentially, any combination of these processes may execute logically in parallel. In practice, some combinations are explicitly prohibited by interrupt masking. This prevents undesirable results such as data loss caused by performance degradation due to processor sharing.

Second, the trap process is scheduled by real machine traps, which occur while the processor is assigned to a virtual processor. Because interrupts may occur during execution of the trap process, it can potentially run in parallel with device processes or the stack process.

Finally, the dispatcher process is scheduled explicitly by either the initialization process, a device process, or the trap process. The scheduling process puts itself to sleep in awaking the dispatcher process. In the case of a device or trap process, the dispatcher is awakened when a virtual processor, which was assigned the real processor when the trap or device process was awakened, is preempted. Again, because interrupts may occur while the dispatcher process is executing, it can potentially run in parallel with device processes or the stack fault process. It cannot run concurrently with the trap process.

With minor exception, all VMM processes execute a sequential program and wake up another process. One exception is the dispatcher process which could idle forever in the absence of interrupts. Also, the dispatcher process does not awaken a VMM process; rather, it assigns the real processor to a virtual processor. Nor does the virtual processor awaken a VMM process; it loses the real processor. For this discussion, consider a virtual processor to be a null VMM process which only serves to awaken another process or be awakened when all VMM processes go to sleep.

The picture of processor sharing one obtains from this design is quite simple. It is similar to nested subroutine calls of sequential programs in that the last process to be awakened is the first process to be put to sleep.

2.2.3 Critical Sections. -

Virtual machine monitor processes perform indivisible operations on shared variables, hence, critical sections must exist. Critical sections must execute in a mutually exclusive fashion in order to prevent unpredictable results [3]. A rather straightforward but inelegant solution to the mutual exclusion problem may be found by observing that at least one of the communicating processes is always a device process, which is scheduled by interrupts. To insure mutual exclusion of critical sections, it is sufficient to disable interrupts on entry and reenable them on exit, as the Nova 3/D is a one processor system.

If more than one processor were available, it would be desirable to build the VMM upon a software nucleus which would hide interrupts and provide process scheduling as well as synchronization primitives.

2.3 Virtual Processors.

2.3.1 States Of Virtual Processors. -

From the vantage point of a VMM, a virtual processor assumes four states corresponding to the running condition of a real processor. These are called _ready_, _running_, _interpreting_ and _blocked_. There is also one state, called _terminated_, which corresponds to the real processor halt condition. These states have the following meaning:

1. _Terminated_. The virtual store is not realized with the possible exception of a portion of virtual main memory. The virtual processor is not a candidate for real processor assignment by the dispatcher process. Transitions may continue in the virtual store due to virtual device actions and in realized main memory due to virtual data channel transfers, however, the virtual processor may not respond. A real processor address translation map is neither composed nor enabled for the virtual processor.

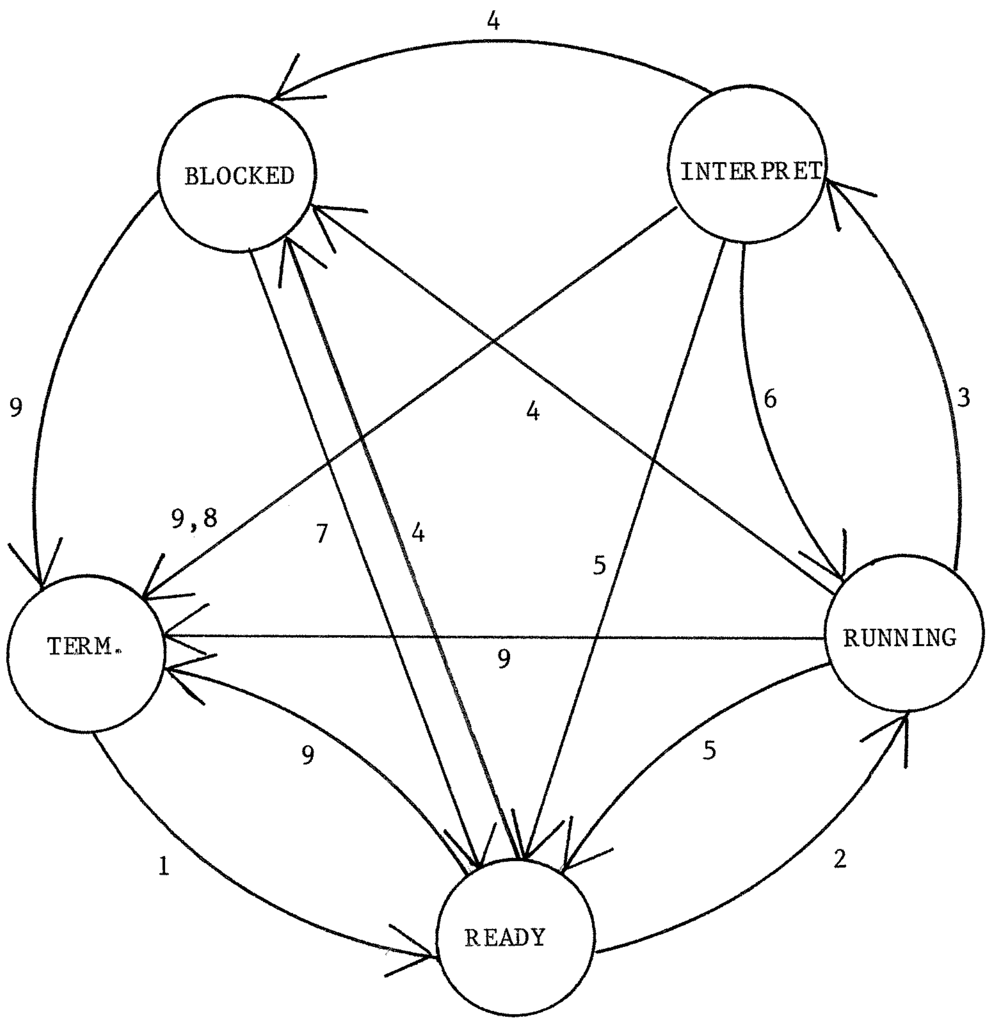
2. _Ready_. Same as terminated except that the virtual processor is eligible for real processor assignment, that is, promotion to the running state.

3. _Running_. The virtual processor is assigned the real processor for the purpose of innocuous instruction execution. Occasionally, the real processor may be lost when a trap, stack fault, or device process is scheduled, however, the real processor is returned unless a preemption occurs. Preemption means that the virtual processor loses the real processor until it is returned by the dispatcher process. A real address translation map is composed and enabled; virtual accumulators, frame pointer, stack pointer, and carry bit are realized. As in the terminated state, transitions due to virtual device actions and data channel and data channel transfers may occur.

4. _Interpreting_. A form of the running state in which a virtual processor action is being supported by the VMM. A real address translation map is composed but not enabled; the virtual store is not realized except a portion of virtual main memory.

5. _Blocked_. Same as terminated except that the virtual processor is eligible for promotion to the ready state upon realization of a portion of virtual main memory.

In summary, interpretation of virtual instructions and simulation of virtual interrupts, stack faults, and traps occurs in the running and interpreting states. The ready state exists due to processor sharing; the blocked state is required for the same reason and because of the length of time required for virtual memory realization. Virtual store state transitions caused by virtual devices are independent of virtual processor state, however, the consequences of these transitions are significant only when a virtual processor is running or interpreting.



1 - START or CONTINUE console command

2 - Dispatcher action

3 - Trap

4 - Page Fault

5 - Preemption

6 - Resume

7 - Page Fault handled

8 - Halt instruction

9 - STOP or RESET command

FIGURE 3

Virtual Processor States and Their Transitions

2.3.2 Virtual Processor State Transitions. -

Transitions between states of a virtual processor are supported by VMM processes. Of the twenty possible transitions between five states, only fourteen exist. Figure 3 shows these transitions. A description of each transition and its causes is given below.

1. Terminated to Ready. Caused by the interpretation of a START or CONTINUE virtual console command by the TTI1 device process.

2. Ready to Running. Caused by selection of a ready virtual processor by the dispatcher process. The dispatcher checks for a pending virtual interrupt or stack fault and simulates it if appropriate, composes a real address translation map, realizes virtual special registers, and assigns the real processor.

3. Running to Interpreting. Caused by a real machine trap while the virtual processor is assigned the real processor. Virtual processor state is saved in a local store while the trapping condition is handled. May also be caused by a stack fault or an interrupt from a device mapped to the virtual processor.

4. Running to Blocked. Caused by a real processor validity violation while a virtual processor is assigned the real processor. This corresponds to a page fault. The state of the virtual processor is saved in the virtual store and it is preempted.

5. Running to Ready. Caused by termination of a time slice. The state of the virtual processor is saved in the virtual store and it is preempted.

6. Interpreting to Ready. Caused by termination of a time slice while the VMM is performing an interpretation for the virtual processor. May also be caused by a virtual mode change due to execution of a virtual mode change instruction or simulation of a virtual trap, stack fault, or interrupt. The state of the virtual processor is saved in the virtual store and it is preempted.

7. Interpreting to Blocked. Caused by a page fault during interpretation of a virtual instruction or simulation of a virtual trap, stack fault, or interrupt. The state of the virtual processor is saved in the virtual store and it is preempted.

8. Ready to Blocked. Caused by a page fault during simulation of a virtual interrupt or stack fault by the dispatcher process. The state of the virtual processor is saved in the virtual store and it is preempted.

9. Interpreting to Running. Caused by completion of a virtual instruction interpretation or simulation of a virtual trap, stack fault or interrupt. Appropriate virtual special registers are realized and the virtual processor is reassigned

the real processor.

10. Blocked to Ready. Caused by completion of page fault servicing by the disk device process.

11. Interpreting to Terminated. Caused by interpretation of a virtual HALT instruction. May also be caused by an interpretation of a virtual STOP or RESET console command by the TTI1 device process. The state of the virtual processor is saved in the virtual store and it is preempted.

12. Running to Terminated. Caused by interpretation of a virtual STOP of RESET console command by the TTI1 device process. The state of the virtual processor is saved in the virtual store and it is preempted.

13. Blocked to Terminated. Caused by interpretation of a virtual STOP or RESET console command by the TTI1 device process.

14. Ready to Terminated. Caused by the interpretation of a virtual STOP or RESET console command by the TTO1 device process.

2.4 The Disk Subsystem.

The disk subsystem is a collection of programs shared by three processes which supports I/O to the 4234 moving head disk and the 6030 diskette. These programs are executed by the TTO1 process to support virtual processor program loading, by the trap process to support virtual disk I/O, again by the trap process to support page fault handling, and by the DKP (disk) process to support the previous three functions. When the TTO1 or trap process requires disk I/O, it first acquires a free disk record, generates appropriate information, loads the record, and passes it to the disk subsystem. The subsystem performs the I/O, recovers from certain real disk errors, and signals completion. A disk record contains the following information.

1. Identification: Specifies the record's origin, the virtual processor it is associated with, and its position in a sequence of related records.

2. Disk Drive: Specifies a drive where 0 is the 4234 moving head disk and 1 and 2 are the 6030 diskettes.

3. Physical Disk Address. The address on disk to be used in the transfer is composed of three parts:

    1. Cylinder (track): Specifies a real cylinder (track) address in the range 0..407 if the 4234 is selected and 0..63 if the 6030 is selected.

2. Surface: Specifies a real surface address in the range 0..3 if the 4234 is selected.

3. Starting Sector: Specifies a real starting sector address in the range 0..11 if the 4234 is selected and 0..7 if the 6030 is selected.

4. Sector Count: Specifies the number of sectors to be transfered in the range 1..16 if the 4234 is selected and 1..8 if the 6030 is selected.

5. Starting Address: Specifies a logical starting address in real main memory for the transfer.

6. Read/Write: Specifies the direction of transfer.

7. Data Channel Mapping: Specifies the logical to physical page address mapping for the data channel transfer. Only five mappings need to be specified since a maximum of sixteen contiguous sectors may be transfered in one operation. These sectors could span five pages.

The disk subsystem is composed of three parts, a disk record manager, a disk record queuer, and a disk driver. The record manager has the task of allocating and recovering disk records. The disk queuer maintains a queue of disk records and passes records to the disk driver according to a service discipline. Finally, the disk driver must actually perform the disk I/O based on the information contained in the disk record. This involves loading the data channel map, initiating a seek, initiating a read or write, and signaling completion. Should errors occur at any stage, the driver must attempt recovery; if the error is unrecoverable, the driver communicates this condition to the operator.

The disk subsystem is concerned only with the details of disk transfers. In the context of VMM design, this is not particularly relevant. What is more significant is the manner in which disk records are generated. In closing, recall that the disk subsystem is a collection of programs, not processes; separate activations of the same program could be executed concurrently by several VMM processes.

2.5 Simulation Of Virtual Interrupts And Stack Faults.

Whenever the virtual processor is assigned the real processor by a VMM process, a virtual interrupt or stack fault may be pending. Simulation is required when virtual interrupts are enabled, virtual mapping is not inhibited, and a virtual interrupt or stack fault is pending. Virtual interrupts are pending if a least one virtual done flag is high and interrupts for the corresponding device are not disabled. A virtual stack fault is pending if a real stack fault occurred while the virtual processor was running. Virtual interrupts are enabled when the virtual CPU busy flag is one and the current virtual program counter is more than one greater than the address at which the virtual CPU busy flag was set.

Interrupt simulation begins by fetching the contents of virtual physical location one. This is the address of the virtual interrupt handler or the beginning of an indirect chain to it. The chain is followed until the end; should a link exist in an unrealized portion of virtual main memory, a program is entered which realizes the required virtual store. The contents of the virtual program counter are stored in virtual location zero. The virtual program counter is loaded with the address of the virtual interrupt handler. The CPU busy flag (interrupt enable) and MAP done flag (mode switch) are cleared.

Simulation of stack faults proceeds in the same manner except that the address of the stack fault handler (or a link to it) is fetched from virtual physical location three. Virtual stack fault simulation has precedence over virtual interrupt simulation, reflecting this condition in the real processor.

CHAPTER THREE

The Trap Process

3.1 Introduction.

The trap process is awakened by a real processor violation trap or explicit trap instruction. In every case, the trap process first determines if the trap corresponds to a virtual trap. If so, a virtual trap is simulated; otherwise, actions associated with instruction interpretation or virtual memory are performed.

3.2 Virtual Memory.

In order for a virtual processor to an execute innocuous memory reference instructions, the addressed portion of virtual main memory must be realized. The virtual machine monitor must support some form of memory management, more specifically, a demand paged virtual memory system. The selection of paging, with page size of 1024 words, is dictated by the Nova hardware; demand paging is appropriate since the VMM cannot anticipate the addressing behavior of a virtual processor.

3.2.1 Hardware Support. -

The memory management and protection hardware of the Nova 3 was designed with the specific goal of supporting virtual memory. Address translation for running programs is supported by two thirty-two register program maps, which map logical page addresses (range 0..31) to physical page addresses (range 0..127). A logical page may also be mapped into a validity violation. Any attempt to access such a logical page results in a real processor violation trap; a validity violation may be interpreted as a page fault or as a signal to set an access bit maintained in software. Logical pages may also be individually write protected. A write violation may be interpreted as a signal to set a dirty bit maintained in software. When a trap occurs, the real processor saves the logical address of the instruction causing the trap and the logical page address of the violation. After a page fault, these registers indicate where to restart the virtual processor and which page to realize.

3.2.2 Address Translation And Virtual Memory Management. -

The notion of address translation supported by the VMM is twofold. First, while a virtual processor is running in supervisor mode, virtual processor physical addresses are mapped to real processor physical addresses. Second, while the virtual processor is running in virtual user mode, the VMM supports virtual memory management and virtual processor logical addresses are mapped to real processor physical addresses. This is actually a double mapping; the mapping from virtual processor logical addresses to virtual processor physical addresses is determined by virtual program maps. The mapping from virtual processor physical addresses to real processor physical addresses is determined by the VMM. Table 1 summarizes this organization. Composition of a real processor map of either type is performed by the dispatcher process immediately before a virtual processor is assigned the real processor.

EXECUTION STATE	REAL PROCESSOR MODE	MAPPING
VMM Program	Supervisor	None
Virtual Processor Supervisor Mode Program	User	Virtual Processor Physical to Real Processor Physical
Virtual Processor User Mode Program	User	Virtual Processor Logical to Virtual Processor Physical to Real Processor Physical

TABLE 1

Memory Mapping Organization

3.2.3 The Page Fault Handler. -

When a validity violation occurs, the trap process is awakened. The trap process must first determine if the violation is to be interpreted as a page fault or if it corresponds to a virtual validity violation, that is, a violation which occurred while a virtual processor was running in user mode with virtual validity protection enabled on that logical page. Should the violation correspond to a page fault, a page fault handling program is executed. This program may also be entered during a sensitive instruction interpretation or during simulation of a virtual trap, stack fault, or interrupt if an addressed portion of virtual main memory is not realized.

The goal of the page fault handler is to redefine the mapping between virtual processor physical pages and real processor physical pages (frames) such that restarting the virtual processor will produce a minimal number of future page faults. This first involves selection of a frame which is currently unmapped (free) or which is mapped but can be replaced with minimal unfavorable consequences. In the latter case, a replacement algorithm is used to select a frame whose contents may need to be saved in the virtual store (swapping out). Swapping out is performed by giving the disk subsystem a record with the following contents:

1. Id: page fault / current virtual processor / first record.

2. Drive: 0

3. Cylinder: Surface: Sector: disk address for virtual processor physical page being swapped out.

4. Sector Count: 4

5. Memory Address: virtual processor physical page address for page being swapped out.

6. Read/Write: write.

7. Data Channel Mapping: virtual processor physical page address for page being swapped out to frame address.

Page fault handling is completed by swapping in the needed virtual processor physical page and updating status information. Swapping in is done in the same fashion as swapping out. The record contents are:

1. Id: page fault / current virtual processor / last record.

2. Drive: 0

3. Cylinder: Surface: Sector: disk address for virtual processor physical page being swapped in.

4. Sector Count: 4

5. Memory Address: virtual processor physical page address for page being swapped in.

6. Read/Write: read.

7. Data Channel Mapping: virtual processor physical page address for page being swapped in to frame address.

Information which should be updated is the frame table, which indicates which virtual processor physical page is associated with each frame, the page descriptors, which tell what frame if any is associated with each virtual processor physical page, and the dirty bits.

Dirty bits are maintained to prevent unnecessary swapping out of frames which have not been altered since they were swapped in. When a frame is swapped in, its dirty bit is normally set to false and real processor write protection is enabled. Any attempt to write on the protected frame causes a trap, awakening the trap process. The trap process must determine if the write violation corresponds to a virtual violation. If not, the dirty bit for that frame is set to true and write protection is removed. The virtual processor is then restarted and the write instruction is allowed to execute normally.

Pages of virtual processors should be realized only in mutually exclusive partitions of the frames not used by the VMM. This prevents problems associated with frame poaching. Partition size is initially defined during system generation. In order to change partition size, the system must be regenerated.

3.2.4 The Replacement Algorithm. -

The replacement algorithm is perhaps the most important feature of a page fault handler. There are several possibilities, such as, first in first out, least recently used, and random selection. It is not clear which of these would yield the best performance. In the face of uncertainty, a modified random selection algorithm is chosen. Performance measurements would be needed to determine if this algorithm is the best.

The modified random selection algorithm holds that a frame should be selected at random from an unlocked subset of the virtual processor's frame partition. Frames are locked when it is certain that their replacement would cause a page fault. The following frames are always locked:

1. Frame containing virtual processor physical page zero.

2. Frame containing the instruction which caused the trap.

3. Frames containing indirect links in a current data fetch.

4. Frames involved in a virtual data channel transfer.

Locking of frames occurs at the beginning of the replacement algorithm. The frames holding virtual physical page zero and the trapping instruction are easily determined through page descriptors. Others are harder to identify. Every time a page is swapped in, the corresponding frame is entered on a locked list for the appropriate virtual processor. When a replacement is needed, the address of the instruction causing the fault is compared to the instruction address of the previous fault. If they are identical, then all frames in the locked list are locked. Otherwise, the locked list is cleared.

Suppose an indirect chain spanned several pages and each link reference produced a page fault; the locked list would grow as each page was swapped in and links would not be swapped out. The list would be cleared when another instruction produced a fault. Should the number of frames spanned by the indirect chain be greater than the partition size, all frames will become locked and the virtual processor would be permanently blocked. This condition is detected in the following way. If the number of locked frames equals the frame partition size, the address chain for data fetch of the current instruction is followed. If it spans the frame partition, the permanently blocked condition is present. A message indicating this fact is sent to the VMM operator's console.

It is possible to cause the locked list to grow without proper cause. For example, the instruction sequence

1. address memory indirectly through a pointer.

2. increment the pointer.

3. go to 1.

might produce a page fault from the same instruction every time a page boundary was crossed. To allow correct execution of such sequences, the locked list is cleared whenever all frames become locked.

3.3 Control Sensitive Instruction Interpretation.

A primary function of the VMM is to provide virtual processor actions which correspond to instruction execution. These actions work on the entire virtual store. In all cases, instructions which require VMM support are I/O instructions. They may be addressed to the CPU, to the MMPU, or to an I/O device. While a virtual processor is assigned the real processor, any attempt to execute an I/O instruction results in a violation trap, awakening the trap process. The trap process must first determine if the violation corresponds to a virtual I/O protection violation. If not, the instruction is interpreted. This is done by fetching the instruction from the virtual store, decoding it, and executing an appropriate program.

3.3.1 CPU Instructions. -

Instructions which address the CPU (device 77) function to control the virtual interrupt facility, read the virtual switches, and halt the processor. Components of the virtual store referenced by these instructions are:

1. virtual accumulators

2. virtual program counter

3. virtual data switches

4. virtual busy and done flags for each virtual device

5. virtual interrupt disable flag for each virtual device

6. virtual CPU busy flag (interrupt enable flag)

7. virtual CPU done flag (power failure indicator)

8. virtual MMPU busy and done flags

The action designated by the transfer field is performed first followed by that of the control field. If the transfer field is SKP, control field values are BZ, BN, DZ, or DN. The following virtual actions are performed corresponding to transfer and control commands.

Transfer:

DIA (read switches): Contents of the virtual console switches are loaded into the designated accumulator. Virtual console switches are set by virtual console commands.

DIB (interrupt acknowledge): The done and interrupt disable flags of each virtual device of the virtual processor are searched until an interrupt pending condition (done=1, disabled=0) is found. The order of search is DKP, CDR, LPT, RTC, TTI, TTI1, TTO, TTO1. The device address of the first device meeting the condition is loaded into the designated virtual accumulator. If no device meets the condition, zero is loaded.

DOB (mask out): The interrupt disable flags for virtual devices of the virtual processor are set according to the bits in the designated virtual accumulator. A flag is set if its mask out bit is one, cleared if it is zero. The correspondence between bits and devices is: 7 - DKP; 10 - CDR; 12 - LPT; 13 - RTC; 14 - TTI and TTI1; 15 - TTO and TTO1.

DIC (clear I/O devices): The busy, done, and interrupt disable flags for each virtual device of the virtual processor are set to zero; the virtual MMPU busy and done flags are also cleared.

DOC (halt): The virtual processor is preempted.

NIO: No operation is performed.

SKP (skip): Interpret control command as BZ, BN, DZ, and DN.

Control:

S (enable interrupts): Set the virtual CPU busy flag to one and save the address of the instruction.

C (disable interrupts): Set the virtual CPU busy flag to zero.

BN (skip if interrupts enabled): The virtual program counter is incremented if the virtual CPU busy flag is one.

BZ (skip if interrupts disabled): The virtual program counter is incremented if the virtual CPU busy flag is zero.

DN (skip if power failure): No operation since there is no virtual power failure.

DZ (skip if power available): Increment the virtual program counter.

3.3.2 Skip Tests And NIO. -

If the transfer field of an instruction addressing an I/O device or the MMPU is SKP, action is determined by the control field. These actions are:

BZ (skip on busy zero): Increment the virtual program counter if the virtual busy flag is zero.

BN (skip on busy non zero): Increment the virtual program counter if the virtual busy flag is one.

DZ (skip on done zero): Increment the virtual program counter if the virtual done flag is zero.

DN (skip on done non zero): Increment the virtual program counter if the virtual done flag is one.

If the transfer field of an instruction addressing an I/O device or the MMPU is NIO, then no action due to the transfer field occurs. However, action specified by the control field is performed.

3.3.3 MMPU Instructions. -

The MMPU instructions address two devices, MAP (device 2) and MAP1 (device 3). They function to control the virtual MMPU facility by reading and writing virtual MMPU registers. Also included is the explicit TRAP instruction required for virtual mode change. Components of the virtual store affected by instructions addressing MAP are:

1. virtual accumulators

2. virtual program counter

3. virtual program maps A and B

4. virtual data channel maps A and B

5. virtual MMPU status words (current and previous)

6. virtual MAP busy flag (mode switch)

7. virtual MAP done flag (data channel error indicator)

8. virtual violation data register

The following actions correspond to transfer and control commands addressing MAP (device 2).

Transfer:

DOB (load map): A virtual map register is loaded according to the contents of the designated virtual accumulator.

DIA (read MMPU status): The current virtual MMPU status word is loaded into the designated virtual accumulator.

DOA (write MMPU status): The contents of the current MMPU status word are copied into the previous MMPU status word. The contents of the designated virtual accumulator is loaded into the current virtual MMPU status word.

DIB (read violation data): The virtual violation data register is loaded into the designated virtual accumulator.

Control:

P (map single cycle): The entire map single cycle sequence must be completely interpreted. If the next instruction is realized then it is fetched from the realized virtual store. If not, the page fault handler is entered. The effective address for the data fetch of the instruction is computed; this is a virtual processor logical address. A corresponding virtual processor physical address is fetched from the virtual program map indicated by the single cycle map select bit in the virtual MMPU status word. Should a virtual validity violation be present, a virtual single cycle validity violation is simulated.

If the virtual processor physical address is not realized, the page fault handler is entered. The virtual processor physical address is used in software interpretation of the command. If the command causes writing in the virtual store and if single cycle write protection is enabled, a single cycle write violation is simulated.

If the page fault handler is entered, the map single cycle instruction is restarted. Should subsequent page faults occur, needed frames will not be swapped out since they are on the locked list.

C (clear violation): The virtual violation data register, virtual MAP busy flag, and virtual MAP done flag are cleared.

Components of the virtual store referenced by instructions addressing MAP1 are:

1. virtual accumulators

2. virtual page check register

3. virtual violation address register

The following actions correspond to transfer and control commands addressing MAP1 (device 3):

Transfer:

DOA (initiate page check): The contents of the designated virtual accumulator is loaded into the virtual page check register.

DIA (page check): The contents of the virtual map register selected by the virtual page check register are loaded into the designated virtual accumulator.

DIB (read violation address): The contents of the virtual violation address register are loaded into the designated virtual accumulator.

Control:

C (clear map): All virtual registers associated with the MMPU are cleared.

Explicit Trap to User Mode. A real processor enters user mode by appropriately setting bits in the MMPU status word and executing a defer cycle. As previously discussed, a virtual processor may enter virtual user mode by executing an explicit trap. The instruction following the trap should specify a defer cycle. A trap executed while a virtual processor is running awakens the trap process.

The trap process first determines if the trap corresponds to a jump to virtual user mode. If not, a virtual explicit trap is simulated. The previous virtual MMPU status word is compared to the current virtual

MMPU status word. Action is not taken unless differences are found in the enable bit, the inhibit bit, or the map select bit. Interpretation continues only if the enable bit is one and the inhibit bit is zero. In this case, the virtual violation data register is inspected for error conditions. If any exist, a virtual violation trap should be simulated. Otherwise, the next instruction is fetched if it is realized. If not, the page fault handler is entered. This instruction should be a jump indirect to a virtual processor physical address. If the address is realized, its contents are fetched; if not, the page fault handler is entered. These contents are either a virtual processor logical starting address or a pointer to one. In the latter case, the indirect chain must be followed until the starting address is found.

In mapping virtual logical addresses to real physical addresses, a virtual validity violation is simulated if a virtual map so indicates and the page fault handler is entered if a virtual processor physical page is not realized. Once the starting address is determined, it is loaded into the virtual program counter. The virtual busy flag (mode switch) for MAP (device 2) is set to one and the virtual processor is preempted. When it is later promoted to running, virtual processor logical addresses will be mapped to real processor physical addresses and a virtual user mode will exist.

3.3.4 TTO And TTI (Teletype) Instructions. -

The VMM supports mapping between virtual teletypes and the CRTs connected to the multiplexer. Hence, all virtual teletype I/O will result in multiplexer I/O by the VMM. Components of the virtual store affected by instructions addressing TTO or TTI are:

1. virtual TTI busy flag
2. virtual TTI done flag
3. virtual TTO busy flag
4. virtual TTO done flag
5. virtual TTI input buffer
6. virtual TTO output buffer

The following actions correspond to transfer and control commands addressing TTI.

<u>Transfer</u>:

DIA (read character buffer): The contents of the virtual input buffer are loaded in the designated virtual accumulator.

<u>Control</u>:

S: Set TTI busy to one and TTI done to zero.

C: Clear virtual TTI done and busy.

The following actions correspond to transfer and control commands addressing TTO.

Transfer:

DOA (write character buffer): The contents of the designated virtual accumulator are loaded into the virtual output buffer.

Control:

S: Set virtual TTO busy to one and virtual TTO done to zero. Issue a multiplexer instruction to send the character in the virtual output buffer to the appropriate CRT.

C: Clear virtual TTO busy and virtual TTO done.

3.3.5 RTC (Real Time Clock) Instructions. -

All virtual processors have access to a virtual real time clock. The concept of virtual real time at first seems contradictory. It is, however, a required facility if virtual processors are to support system software. Time is of interest to a virtual processor only when it is the running or interpreting states. When it is blocked or ready, time is at a standstill; this reflects the fact that real processors never experience time losses due to processor sharing. Transitions through these states should appear instantaneous in virtual time.

Real time clocks are simulated by counters which decrement only when a virtual processor is running or interpreting. Virtual clock counters are decremented in response to clock pulses from the real machine real time clock.

Components of the virtual store effected by instructions addressing RTC are:

1. virtual accumulators
2. virtual RTC output buffer
3. virtual RTC busy flag
4. virtual RTC done flag

5. virtual clock counter

The following actions correspond to transfer and control commands.

Transfer:

DOA (set clock frequency): Load the output buffer with the contents of the designated virtual accumulator. The value in the output buffer determines virtual clock frequency, that is, the initial value of the virtual clock counter.

Control:

S: Set RTC busy to one and done to zero, enabling virtual RTC interrupts.

C: Clear RTC busy and done.

3.3.6 DKP (Disk) Instructions. -

Instructions addressing DKP correspond to virtual disk I/O commands for the virtual 4234 moving head disk (drive 0) and the virtual 6030 diskettes (drives 1 and 2). Virtual diskette I/O is possible only if the virtual processor is mapped to a real diskette drive. I/O on a virtual 4234 is allowed only if the virtual processor has been allocated real 4234 cylinders. The VMM supports virtual disk I/O by causing transitions in virtual DKP registers and by creating disk records corresponding to the specified commands. When a disk record is completed, the VMM realizes any unrealized portions of the virtual store needed for the virtual disk transfer and passes the record to the disk queuer.

Virtual disks differ from real disks in several ways. Virtual seeks cause only updating of a virtual register, not a real seek. On the other hand, a virtual read or write results in a real seek and a real transfer operation. It may also initiate handling of page faults associated with unrealized virtual store required in the virtual transfer. Several errors which are possible during real disk I/O are not present in virtual disks. These are the invalid status condition, unsafe condition, address error, checkword error, and data late error. The VMM recovers from these errors before completing virtual disk I/O.

Virtual disk errors which are possible are the seek error, which occurs if the specified track or cylinder is out of range, and the end of cylinder error, which happens if the end of cylinder or track is reached before the I/O completes.

Software mapping of disk addresses is performed only with respect to cylinders on the 4234 moving head disk. A virtual 4234 disk has n cylinders numbered 0..n-1 where $n \leq 408-k$. There are 408 cylinders on a real 4234 disk and k cylinders are used by the VMM. When a virtual processor is allocated cylinders, a base cylinder address is defined which is the physical address of virtual cylinder zero. Hence, the physical address corresponding to a virtual cylinder address may be computed by

phy. cyl. address = virt. cyl. address + base cyl. address.

Components of the virtual store referenced by instructions addressing device DKP are:

1. virtual accumulators
2. virtual disk address and sector count register (DASCR)
3. virtual command and cylinder register
4. virtual DKP status register
5. virtual memory address register

The following actions are performed corresponding to transfer and control commands.

<u>Transfer</u>:

DOC (specify disk address and sector count): the virtual DASCR is loaded with the contents of the designated virtual accumulator; if drive 0 is selected, the diskette bit of the virtual DKP status register is cleared; otherwise it is set to one.

DIC (read disk address and sector count): the designated virtual accumulator is loaded with the contents of the virtual DASCR.

DOA (specify command and track): if the command is a seek, the virtual command and track register is loaded with the contents of the designated virtual accumulator; otherwise, only the command field is loaded.

DIA (read status): the designated virtual accumulator is loaded with the virtual DKP status register.

DOB (load memory address): the virtual memory address register is loaded with the contents of the designated virtual accumulator.

DIB (read memory address): the designated virtual accumulator is loaded with the virtual memory address register.