(19) Japan Patent Office (JP)

(12) **(UNEXAMINED) PATENT APPLICATION PUBLICATION (A)**

(11) Patent Application Publication Number: 2001-318797 (P2001-318797A)

(43) Publication Date: November 16, 2001

(51) Int. Cl.[7]	Identification Code	FI		Theme code (reference)
G06F 9/46 12/14	350	C06F 9/46	12/14	350 5B017
	310			310L 5B085
15/00	330	15/00		330A 5B098

Request for Examination: Requested Number of Claims: 42 OL (Total of 23 pages)

(21) Patent Application No.: 2000-281632 (P2000-281632)
(22) Application Date: September 18, 2000 (2000.9.18)

(31) Priority No.: 09/568629
(32) Priority Date: May 10, 2000 (2000.5.10)
(33) Priority Claim Country: U.S.

(71) Applicant: 000004237
NEC Corporation
5-7-1 Shiba, Minato-ku, Tokyo

(72) Inventor: Leslie J. French
c/o NEC USA, Inc.
4 Independence Way, Princeton, NJ 08540

(74) Agent: 100097157
Patent Attorney Yuji Katsuragi

F term (reference): 5B017 AA01 BA03 BB06 CA16 5B085 AE00 AE06 AE29
5B098 GD14 GD21 HH01

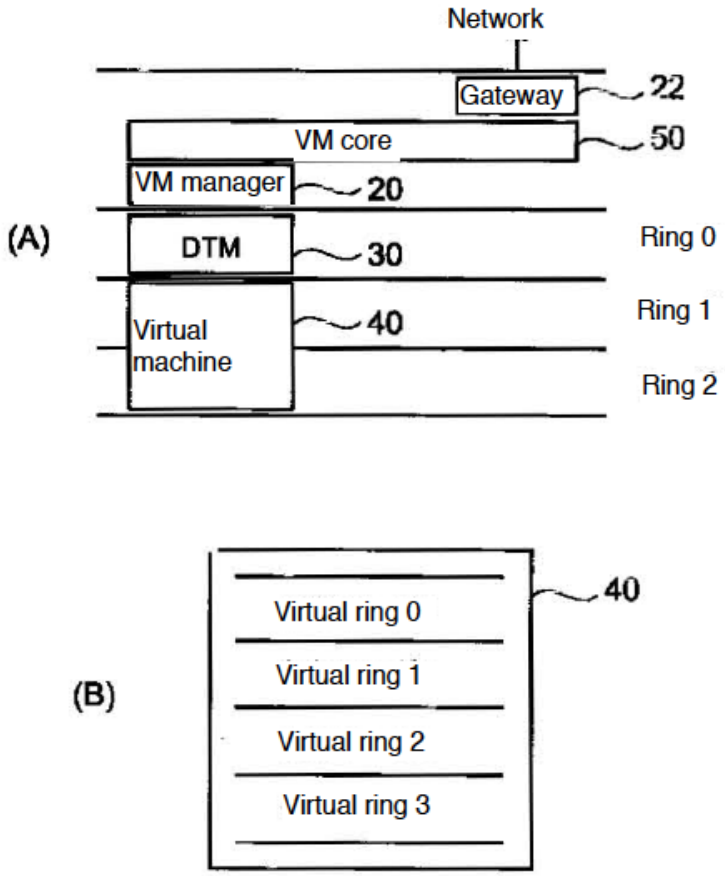
(54) Title of the Invention: Automatic data processing apparatus

(57) [Abstract]

[Problem] To provide a virtual machine architecture that provides virtual environments where codes are directly executed on a processor and a security shell is implemented around each virtual environment.

[Solution] A privilege ring hierarchy within a virtual environment is provided as a subset of a

privilege ring hierarchy that is used by hardware. Access from within a virtual environment to hardware requiring the highest privilege is strictly controlled. Communications between a virtual machine and the outside world are performed via a simulated network. The network is protected by a virtual machine operating as a firewall, and data transactions are protected by encryption. The hardware has an encryption key that cannot be used with a virtual environment. A virtual environment receives a unique key that it is acquirable from an external server by a secure transaction that uses a hardware key.



[Claims]

[Claim 1] In an automatic data processing apparatus that uses at least three real privilege rings to process information and control access to system resources and memory, provides at least one virtual machine environment and associates information and privilege to predetermined rings,

an automatic data processing apparatus comprising:

a real environment comprising a highest privilege real ring with restrictions imposed on processes that can change the real ring properties, a second highest privilege real ring and remaining real rings;

a virtual machine manager that operates in said highest privilege real ring;

a virtual machine environment comprising a portion of said remaining real rings;

a first data transfer manager that operates in said second highest privilege real ring and processes information transfers entering and exiting said virtual machine environment;

a hierarchical virtual privilege rings inside said virtual machine environment; and

a virtual environment descriptor table formed inside said virtual machine environment and includes associations between privilege rings and information inside said virtual machine environment;

wherein:

information inside said virtual machine environment is associated with said hierarchical virtual privilege rings and is stored in a part of said memory;

information processed by said automatic data processing apparatus is accessible by system resources and information that are associated with a real ring with the same or lower privilege;

information inside said virtual machine environment that is processed by said automatic data processing apparatus is accessible by system resources and information that are associated with a virtual privilege ring with the same or lower privilege and can only access a portion of the memory locations of the memory of said virtual machine environment;

real privileges that can be used by information associated with a virtual privilege ring and processed by said automatic data processing apparatus are controlled by said virtual machine manager if access is requested to a real privilege that exceeds the real ring among said remaining real rings and includes said virtual machine environment; and

a portion of said memory includes main memory and auxiliary storage area.

[Claim 2] The automatic data processing apparatus according to claim 1 wherein said automatic data processing apparatus further comprises shadow descriptor tables that are generated by said virtual machine manager for the virtual privilege rings and map information inside said virtual machine environment to the real privilege ring hierarchy, said virtual environment descriptor tables include associations to said virtual privilege ring hierarchy, and when the information associated with the virtual privilege rings is processed by said automatic data processing apparatus, the shadow descriptor table corresponding to the virtual privilege ring that is associated with the information is used to determine the access privilege.

[Claim 3] The automatic data processing apparatus according to claim 2 wherein said virtual machine manager generates a shadow descriptor table for a virtual privilege ring corresponding to information to be processed when processing is started in a virtual privilege ring.

[Claim 4] The automatic data processing apparatus according to claim 3 wherein said virtual machine manager updates a shadow descriptor table when a change occurs in said virtual environment descriptor table.

[Claim 5] The automatic data processing apparatus according to claim 3 wherein, if information that is processed by said automatic data processing apparatus changes a virtual privilege ring, said virtual machine manager generates a shadow descriptor table for the virtual privilege ring that would be changed by the processed [information].

[Claim 6] The automatic data processing apparatus according to claim 5 wherein said virtual machine manager holds a plurality of shadow descriptor tables including a shadow descriptor table for each virtual privilege ring for which information has been processed.

[Claim 7] The automatic data processing apparatus according to claim 6 wherein said virtual machine manager updates said plurality of shadow descriptor tables when change is made to said virtual environment descriptor table.

[Claim 8] The automatic data processing apparatus according to claim 6 wherein said virtual machine manager regenerates said plurality of shadow descriptor tables when information in a part of said memory is rearranged.

[Claim 9] The automatic data processing apparatus according to claim 2 wherein said virtual machine manager generates and holds a plurality of shadow descriptor tables including a shadow descriptor table for each virtual ring in the hierarchy of said virtual privilege rings.

[Claim 10] The automatic data processing apparatus according to claim 9 wherein said virtual

machine manager updates said plurality of shadow descriptor tables when a change is made to said virtual environment descriptor table.

[Claim 11] The automatic data processing apparatus according to claim 9 wherein said virtual machine manager regenerates said plurality of shadow descriptor tables when information in a part of said memory is rearranged.

[Claim 12] The automatic data processing apparatus according to claim 2 wherein the real ring hierarchy has four rings, the third highest privilege real ring has the highest privilege virtual ring and the fourth highest privilege real ring has the remaining parts of the virtual privilege ring hierarchy.

[Claim 13] The automatic data processing apparatus according to claim 12 wherein the shadow descriptor table that is generated by said virtual machine manager for the highest privilege virtual ring maps all information inside said virtual machine environment to said third highest privilege real ring, and

the shadow descriptor table that is generated by said virtual machine manager for the remaining virtual rings in said virtual ring hierarchy maps information associated with said highest privilege virtual ring to said third highest privilege real ring, maps said fourth highest privilege real ring to information associated with virtual rings of the same or lower privilege as the virtual privilege ring for which said shadow descriptor table has been generated, and maps said highest privilege real ring to information associated with virtual rings of a higher privilege than the virtual privilege ring for which said shadow descriptor table has been generated, thereby causing said virtual machine manager to trap requests to access information associated with virtual rings of a higher privilege among said remaining virtual ring hierarchy.

[Claim 14] The automatic data processing apparatus according to claim 2 further comprising a global descriptor table for the entire said real environment, the global descriptor table comprising shadow descriptor tables and information associated with a hierarchy of real privilege rings.

[Claim 15] The automatic data processing apparatus according to claim 1 wherein information in said virtual machine environment is associated only with two rings consisting of a virtual privilege ring of the highest privilege and a virtual privilege ring of the lowest privilege.

[Claim 16] The automatic data processing apparatus according to claim 15 wherein the real privilege ring hiearchy has four privilege rings so that said virtual machine environment has two real rings, the third highest privilege real ring has the highest privilege virtual ring and the fourth

highest privilege real ring has the lowest privilege virtual ring.

[Claim 17] The automatic data processing apparatus according to claim 1 wherein information that is processed from inside said virtual machine environment would attempt to record the associations with said virtual privilege ring hierarchy inside said virtual environment descriptor table, said virtual machine manager records associations with the real privilege ring hierarchy inside said virtual environment descriptor table instead of associations with said virtual privilege ring hierarchy, and said virtual environment descriptor table is used to determined access privilege when information associated with the inside of said virtual machine environment is processed by said automatic data processing apparatus.

[Claim 18] The automatic data processing apparatus according to claim 17 wherein said virtual machine manager records alternative associations in said virtual environment descriptor table when said virtual environment descriptor table is formed and whevever attempts are made to modify said virtual environment descriptor table.

[Claim 19] The automatic data processing apparatus according to claim 17 wherein said virtual machine manager replaces real privilege ring associations by overwriting after granting permission to attempt to record virtual privilege ring associations in said virtual environment descriptor table.

[Claim 20] The automatic data processing apparatus according to claim 17 wherein attempts to record virtual privilege ring associations in said virtual environment descriptor table are trapped by said virtual machine manager, and said virtual machine manager instead records real privilege ring associations.

[Claim 21] The automatic data processing apparatus according to claim 1 wherein information transfers between said virtual machine environment and other virtual machine environments and between said virtual machine environment and said real environment are processed by a data transfer manager.

[Claim 22] The automatic data processing apparatus according to claim 1 wherein said automatic data processing apparatus further comprises a data transfer manager spool, and information transfers processed by the data transfer manager are temporarily stored in said data transfer manager spool.

[Claim 23] The automatic data processing apparatus according to claim 22 wherein said data transfer manager requests users to confirm information transfers exiting a virtual machine

environment.

[Claim 24] The automatic data processing apparatus according to claim 1 further comprising a virtual firewall comprising said first data transfer manager and said virtual machine environment.

[Claim 25] The automatic data processing apparatus according to claim 24 wherein said automatic data processing apparatus further comprises a second data transfer manager that operates in said second privilege real ring and processes information transfers entering and existing the virtual machine environment, said virtual firewall further comprises said second data transfer manager, said first data transfer manager processes information transfers between said virtual machine environment and the outside of the firewall and said second data transfer manager processes information transfers between said virtual machine environment and the inside of the firewall.

[Claim 26] The automatic data processing apparatus according to claim 24 wherein said first data transfer manager isolates information transfers between said virtual machine environment and the outside of the firewall from information transfers between said virtual machine environment and the inside of the firewall.

[Claim 27] The automatic data processing apparatus according to claim 24 wherein said automatic data processing apparatus further comprises a firewall spool, and information transfers through said firewall are temporarily stored in said firewall spool.

[Claim 28] The automatic data processing apparatus according to claim 27 wherein said firewall requests users to confirm information transfers from inside the firewall to outside the firewall.

[Claim 29] The automatic data processing apparatus according to claim 1 wherein said automatic data processing apparatus further comprises a non-volatile memory that stores a real environment private encryption key, said real environment private encryption key is unique to said automatic data processing apparatus, and access to said real environment private encryption key is limited to said highest privilege real ring and said second highest privilege real ring.

[Claim 30] The automatic data processing apparatus according to claim 29 wherein said virtual machine manager uses a real environment public encryption key corresponding to said real environment private encryption key to encrypt information transfers from said virtual machine environment to said auxiliary storage area that is a portion of said memory, said virtual machine manager uses said real environment private encryption key to decrypt information transfers from said auxiliary storage area that is a portion of said memory to said virtual machine environment,

and said encryption and decryption are invisible from inside said virtual machine environment.
[Claim 31] The automatic data processing apparatus according to claim 29 wherein the real environment encryption key encrypts information transfers exiting said virtual machine environment and is used outside of said virtual machine environment for decrypting information transfers that enter said virtual machine environment, and the use of encryption and decryption is invisible from inside said virtual machine environment.
[Claim 32] The automatic data processing apparatus according to claim 31 wherein decrypted information that is transferred into said virtual machine environment is inaccessible by other virtual machine environments.
[Claim 33] The automatic data processing apparatus according to claim 31 wherein the real environment encryption key is applied by said virtual machine manager.
[Claim 34] The automatic data processing apparatus according to claim 31 wherein the real environment encryption key is applied by said data transfer manager.
[Claim 35] The automatic data processing apparatus according to claim 29 wherein a virtual environment private encryption key and a virtual environment public encryption key are assigned to said virtual machine environment, and the encryption key is different from the encryption keys of the real environment and other virtual machine environments so that other virtual machine environments are inaccessible.
[Claim 36] The automatic data processing apparatus according to claim 35 wherein the virtual environment encryption key encrypts information transfers exiting said virtual machine environment and is used outside of said virtual machine environment for decrypting information transfers that enter said virtual machine environment, and decrypted information that is transferred into said virtual machine environment is inaccessible by other virtual machine environments.
[Claim 37] The apparatus according to claim 36 wherein the virtual environment encryption key is applied by said virtual machine manager.
[Claim 38] The apparatus according to claim 36 wherein the virtual environment encryption key is applied by said data transfer manager.
[Claim 39] The automatic data processing apparatus according to claim 35 wherein the virtual environment encryption key is accessible inside said virtual environment so that procedures inside said virtual environment decrypt information transfers entering said virtual machine

environment and encrypt information transfers that exit said virtual machine environment.
[Claim 40] The automatic data processing apparatus according to claim 35 wherein said automatic data processing apparatus acquires the virtual environment encryption key from an external key server.
[Claim 41] The automatic data processing apparatus according to claim 40 wherein transactions between said automatic data processing apparatus and the external key server concerning the acquisition of the key are securely performed using a real environment private encryption key.
[Claim 42] The automatic data processing apparatus according to claim 1 wherein the number of hierarchies in said virtual privilege rings is equal to the number of hierarchies in said real privilege rings.

[Detailed Description of the Invention]
[0001]
[Technical Field of the Invention] The present invention relates to the field of digital data processing systems where hierarchical protection rings are used to control system operations and relates, in particular, to computer systems having one or more virtual machines (VM), three or more hierarchically organized protection rings that control access to memory locations and the executability of some instructions, and an active security layer outside the virtual machines.
[0002]
[Previous Technology] Problems with the security of "personal" computing devices (PCs) have become increasingly important in many areas of computing. PCs are becoming the mainstay of desktop machines. At the same time, the boundary between stand-alone computers and network machines is becoming blurred. While traditional workstations and small mainframe computers are being replaced with PCs, PCs are also known for their low level of security. As more and more systems are connected to open access networks (e.g., the Internet and access networks of cable modem providers), these systems have become more vulnerable to intrusions occurring at various access levels. Even with the most simple of modem data transactions, the information associated with these transactions is voluminous, and because of the rapidly increasing number of network driven applications, the first and the last lines of defense for system security have invariably depended on "trust."
[0003] A number of patchwork solutions has been proposed and implemented against the

problem of security. One of the simplest approaches is to prevent the spread of "boot block" viruses by preventing a computer from being booted from a removable media (usually a floppy (registered trademark) disk). With this approach, a machine must be rebooted before a user can access the firmware basic input/output system (BIOS), which is often password protected, and boot the machine from a floppy disk.

[0004] A second approach is to replace all existing software in a system with a more secure version. However, because of the increased complexity of software applications, there is an endless supply of security holes and bugs that can be exploited by malicious programmers.

[0005] A third approach is to provide a "veil of secrecy" by the use of passwords and the encryption of stored data. However, it is a mistake to confuse secrecy and security. The concealing of data may make its discovery difficult, but does not solve the problem itself. Even aside from obtaining data and a password file by theft and using a brute force approach to decipher them, the concealment approach is very vulnerable to "internal" attacks. One example of such attacks is where an illicit process obtains a copy of a computer's encryption key or a user's password by monitoring the user's key input. The illicit process then broadcasts the information to the outside world, allowing intruders to bypass even the most advanced encryption technology.

[0006] A fourth approach is to use a software or hardware "firewall" to monitor data downloads and network transactions. However, firewalls are only as effective as their filters and very often must rely on trust to determine whether to allow information to pass through the firewall.

[0007] A fifth method against viral codes and attacks is to scan the codes and monitor the execution of a software. Although this method is becoming widely used in PCs, its weakness has been much documented. In particular, this method is weak when an unrecognized code enters through a defect in an operating system and gains access to system resources located at an operating level lower than that of an anti-virus monitor. This method exhibits particular vulnerability with respect to codes such as word processor macros and network browser applications that run in an application environment. Inevitably, unless a "military-type" approach is implemented whereby the machine is completely isolated, internal attacks that are waged by undetected intrusions would compromise this security method.

[0008] Similar to the "military-type" approach, a sixth approach is to execute software in an emulated "sandbox" environment. Software is loaded into and is executed inside a computer

system emulator where the software execution is monitored for any suspicious activity so as to eliminate or minimize the risk of exposure of the larger system to applications suspected of being infected. However, since many viruses remain dormant until a specific event occurs, ensuring safety requires isolating the suspected application indefinitely. Unless this is done, an infected application that is erroneously deemed to be virus free would exit out of the sandbox, and the infected application would infect the rest of the machine to which it is granted access. However, executing an application indefinitely inside an environment emulator is undesirable for long-term operation. An emulator adds significant complexity to the execution of codes. Software in a sandbox environment is not directly executed by the computer's processor but is run by simulating each instruction. This slows down the system's operation.

[0009] Furthermore, with some solutions, a sandbox environment is created within the framework of a conventional operating system. However, this implementation can only be as safe as the underlying operating system on which the execution happens. For example, a privileged process running on the same computer as the sandbox can access the data and methods of the sandbox environment. It is possible for such privileged process to be the result of a malicious application running on a portion of the computer that is not under the control of the sandbox environment, or to be the result of an unexpected hole in the sandbox implementation.

[0010] These various “two-level” security models are “all-or-nothing” protections and have the flaw of the code being verified sharing a high privilege level with the operating system. This raises the question of the how much trust can be placed that the code would work correctly. Even the presence of minor holes in the interface can destroy the entire protection scheme.

[0011] One of the main problems with legacy operating systems lies in the determination of “intent.” It is difficult when monitoring security to infer whether an operation is an intended operation or not. Disk formatting utilities try to write to the boot block of a device in a way that is indistinguishable from viruses. When a user sends an email to all users on a mailing list, this may be indistinguishable from an email worm that spreads over a network. Attempting to determine which operation is legitimate based on “hindsight” is liable to eventually fail. This is because hackers are becoming better based on heuristic processes at convincing that their code is truly authentic.

[0012] To implement these security models at the hardware/firmware level, PCs (e.g., those based on the Intel architecture) use a hierarchy of privileged rings and segment tables to prevent

processes from interfering and hindering each other.

[0013] Memory 4a (FIG. 1(A)) is divided into blocks called segments 6a to 6e. Whether or not a specific segment can be seen from a process (visibility) and the access right that is granted to the process are determined by a segment table assigned to each process. The process rights are summarized and are determined by one or more descriptor tables (DTs) 8. The descriptors 10a through 10e (FIG. 1(B)) associated with memory segments 6a through 6e specify the starting address and the size (length) of the segment, and the access right to the area.

[0014] Depending on the processor architecture, segments can also be shared among multiple processes. If a segment is made visible to all processes with the same access right, the segment can be placed in a global table. Additionally, each process can have a local table that holds descriptors that are associated only with that process. In this way, the same memory area can have different access rights for different processes.

[0015] For example, computers using a CPU with an Intel architecture have a single global descriptor table (GDT) that is located in the main memory and is accessed via an architecture-specific global descriptor table register (GDTR). With this architecture, multiple GDTs can be provided and activated dynamically during program execution. Further, each process can have a local descriptor table (LDT) that is accessed through a local descriptor table register (LDTR) provided in the same manner as described above.

[0016] The processor architecture also includes another table that is used to determine the operation of the processor in response to interrupts generated by external sources or software. This table is accessed through an interrupt descriptor table register (IDTR).

[0017] The hardware refers to the descriptor table and checks the addresses used by the process. An exception is generated if an address is located outside the space allocated to the process. However, some segments are shared by several processes. To protect these shared segments, a hierarchy of privileged rings as shown in FIG. 2 is provided. Each descriptor table entry is associated with a ring protection level that is used to determine access rights to that entry in the table. All information is assigned to one privileged ring, including procedures that are executed as part of an executing process. The current execution ring of a procedure may change during the execution of a process and further determines the visibility of the memory associated with the procedure.

[0018] In the hierarchy of four rings 0 through 3, the innermost ring 0 is the highest privilege

ring, and the outermost ring 3 is the lowest privilege ring. A procedure in an inner ring can access data in an outer ring (arrow 12). Thus, procedures operating in the innermost ring (e.g., operating system procedures) can access data in all rings.

[0019] Conversely, procedures in an outer ring cannot access data in an inner ring without generating a protection violation exception (arrow 14). To access data in an inner ring, a procedure must branch through "call gate" 18 to the ring containing that data. A call gate is a special "procedure call" descriptor that is recognized by the hardware. Once passed through the gate, there is no guarantee that the procedure that was allowed to branch to a higher level will return to a lower level ring. There is also no guarantee that a procedure that reaches a target ring will do no harm.

[0020] A virtual machine environment is a useful tool for maintaining system security. The emulated sandbox described above is one example. Computer systems with one or more virtual machines typically compress the hierarchy of rings into a lesser number of real rings (which may be a single ring). A real ring has a virtual machine environment, and when an instruction is executed from within the virtual machine, the determination of privilege is delegated to the software responsible for this artificial environment. For example, if a virtual machine is emulated in ring 3, all rings in the virtual machine are simulated from within ring 3.

[0021] Another virtual machine environment has been developed that overcomes the problem of slow emulator execution by executing codes from within a virtual machine directly on a processor. Codes from within any virtual machine environment are executed in the same way as a normal task. The environment management software operates as a process that uses device drivers in the existing operating system or in cooperation with the operating system to regulate access to system resources. Unfortunately, although these methods overcome the speed disadvantages of environment simulation, sandboxes that are embedded in unsecured legacy environments lack security protection for software-based emulators that simulate code execution.

[0022]

[Problems to Be Solved by the Invention] All of the afore-described relates to environments that plug holes in the interface while avoiding excessive emulation. The environment must be open and sandboxed to allow the use of legacy operating systems without modification, while providing reliable multilayered protection and defensive security restrictions.

[0023]

[Means for Solving the Problems] To solve the above problems, it is the object of the present invention to provide a virtual machine architecture whereby codes are directly executed on a processor but a security shell is implemented around each virtual environment. Access to critical system resources (including network access) through the shell is restrictively but transparently controlled. Security restrictions allow apparently open access to network and system resources while maintaining the integrity of each "sandbox."

[0024] The present invention provides an isolated environment by the use of a virtual machine with a very narrowly defined interface through which data must pass rather than trying to verify all interactions. This implementation is invisible to guest legacy operating systems running in a virtual machine.

[0025] Data verification mechanisms are used with transfers between isolated virtual environments that reside in the same hardware. The security mechanism is modular and includes virtual firewalls, inter-hardware and intra-hardware encryption and boot block verification. Protection is provided against moving a physical disk to an "unsafe" environment, changing the operating system configuration and returning the disk to a "safe" environment. Data theft from a discarded or stolen drive is prevented along with the destruction of the disk boot block. Data in non-volatile RAM is hidden or protected by restricted access. Each operating environment is isolated. Firewalls are created between machine data and the network and isolate secure environments containing sensitive data from "high risk" applications.

[0026] The present invention is preferably a small core operating system (e.g., on the order of 1000 lines of code) that protects physical resources so that all or a part of the operating system is stored in a protected bootable memory such as a ROM while minimizing the loss in machine performance. Making the core small makes it possible to adapt to authenticity certifications that show compliance with security models that may change in the future while minimizing the impact on existing systems and preventing loss in machine performance.

[0027] The virtual machine core operating system operates within the highest privilege ring, just like a virtual machine manager that operates as a privileged application. The virtual machine manager manages the emulation of system resources for each virtual environment. A data transfer manager (DTM) operates in the second highest privileged ring and manages data transfers in and out of each virtual environment. A virtual privilege ring is created inside the virtual environment to simulate a normal operating environment. Each virtual machine has its

own operating system but in a restricted environment where all access to machine resources is verified through the virtual machine core.

[0028] In general, almost all instructions in a virtual machine are executed "natively," but instructions that require a ring 0 execution level are trapped in the core. A part of the implementation is for the management of data structures (e.g., processor status registers) that hide privileged operations for virtual machines. With this scheme, a legacy operating system can see a set of system registers that apparently track changes in its own privilege level, but the use of ring 0 instructions, access to physical memory or access to a physical device requires first going through the virtual machine core.

[0029] So as to run an unmodified operating system in a virtual environment, the guest system is provided with a memory model that appears to be identical to a native machine. For example, each machine must see a boot ROM at location FFFF:FFF0 (possibly different) at (virtual) reset. Thus, the virtual memory management system maps the same logical addresses to different physical addresses on the real machine.

[0030] To ensure that each machine operates through its own boot sequence, results that are returned to different virtual machines may not reflect the actual hardware. For example, a machine can access more or less physical memory than present in hardware, and there is no gurantee that all devices on a bus that respond to a probe are visible. An individualized boot ROM is provided for each virtual environment so that the BIOS data structure reflects the virtual state required for that virtual machine.

[0031] Similarly, guest machines are blocked from accessing hardware resources. Input/output instructions and instructions that access external devices through memory-mapped pages are also trapped in the core. To realize this, the core uses a memory management system that allow individual I/O space addresses to be locked out, if possible, for a given task.

[0032] Hardware interrupts are trapped in the core handler and are dispatched from there to the appropriate machine. This mechanism must be used even if there is only one virtual machine. This is because the instructions that enable or disable interrupts are only emulated in a virtual environment. An interrupt can be dispatched directly to its handler only if a device driver can be loaded into ring 0 of the virtual machine. This is the case only if performance is an issue. An example is to avoid machine rescheduling or multiple data copying for large capacity devices (e.g., Ethernet (registered trademark) controllers) running in their own verification area.

Obviously, this mechanism is only available for use against highly trusted applications such as firewalls.

[0033] The concept of a virtual machine extends to auxiliary memories (auxiliary storage areas) such as hard (disk) drives on the machine. Even if multiple machines were to share the same physical device, accesses to absolute block addresses are trapped and converted to virtual disk addresses as if the machines had dedicated drives.

[0034] Other value-added services are possible within the architecture of the present invention by trapping all accesses in consideration of physical attacks to the disk. For example, the present invention can implement transparent disk mirroring or RAID (redundant array of independent disks) systems to improve data recovery from hardware failures. Also at this level, the core can use a key stored in a non-volatile RAM to encrypt all data on the virtual machine so that the disk cannot be read on other machines. For an even more complete protection, the core system can be protected by a unique key in a non-volatile RAM so that the virtual machine data uses a two-level key based on both the machine key and a key that is input by the user.

[0035] Choosing to use these schemes requires a compromise between system security and the impact on the overall system performance. The encryption method has the disadvantage that it is almost impossible to "hot swap" components in the event of a component failure. Also, data recovery from disk crashes becomes much more difficult.

[0036] It is possible for a code that is executed in a separate ring (1 through 3) on the native system to be executed in the same physical ring on the guest system. This presents the possibility that a user level application may change a data structure that should not be accessible. This concern is resolved by implementing a virtual ring system that uses a memory management subsystem. In this way, pages in different virtual rings are kept separate, and task switching between tasks in the same physical ring also switches memory access control.

[0037] The basic implementation of the present invention preferably comprises:

- A loader, preferably stored in a secure boot ROM, for the virtual machine core and virtual machine manager,
- An area in a non-volatile RAM for holding configuration data and security keys,
- A virtual machine core scheduler responsible for running individual virtual machines,
- A virtual machine manager that sets and loads the guest virtual machine configuration and handling the security policies,

- Configuration specifications for the guest virtual machines,
- Virtual boot ROM for the guest virtual machines,
- Virtual non-volatile RAM parameter area for guest virtual machines,
- Secure applications that are loaded to the ring 1 task for guest virtual machines, and
- Disk partitions containing the file systems for the guest virtual machines (including operating systems and application codes).

[0038] Almost all machines include a variety of hardware such as removable media devices, network interfaces and auxiliary hard drives. These resources can be shared between guest machines through the core or can be dedicated to specific machines. The exact placement of these components can also be determined by the security model. For example, having the virtual machine manager loaded from a disk can facilitate driver and policy upgrades (possibly reflecting new hardware configurations). However, in situations where automatic hardware reconfiguration poses a security threat, the entire system can also be loaded from a boot ROM.

[0039] Other security-specific hardware components can also be incorporated into the architecture. One example is a fusible ROM (i.e., a ROM from which data that is once written cannot be read) that implements the hidden key portion of a two-key encryption method. By using such PROMs, the hidden key is not stored in a readable form on the machine. Such PROMs can also be used to implement a challenge-response authentication mechanism.

[0040] The ability to boot a secure system from a removable media is an unacceptable security loophole, especially when it is known that an infected media has been carried near the site. Even if the machine is booted from a known good media, the user is given an opportunity to access the security options of the machine that is believed to be protected. However, removing this option makes it difficult or impossible to set up a new machine.

[0041] There are several solutions to this problem. A first solution is to create a special “setup” machine that allows setting up a hard drive from an existing system. A second solution is to provide a hardware jumper that allows booting from a removable media. A third solution is to provide a “disable” bit for the non-volatile RAM to prevent booting from a removable media and setting the bit when the machine is configured. A fourth solution is to download or insert a special boot ROM that is replaced when the machine begins to run. Which of these options to choose to prevent booting from a removable media depends on the individual site's security environment and operation procedures. After the core is loaded and the virtual machine manager

is started, individual virtual machines are created, and the virtual "boot from removable media" option is enabled (this can be disabled later from the core). This causes the operating system to be installed following the traditional mechanism.

[0042] The need for a manual intervention during installation may be an essential part of the policy in a desktop-type enterprise environment, but may be inappropriate for active network components that are deeply embedded within an international infrastructure. In such systems, active upgrades of the system software may be important. In such situations, it may be preferable to load the core system over the network. By embedding the decryption key in the boot ROM, it becomes possible to ensure that only authorized servers can supply the initial image.

[0043] The subsequent steps in the installation procedure of a fixed system remain essentially the same regardless of how the system is actually managed. Utilities (running on that machine or on another system) are required for formatting and partitioning the bootable hard drive. The boot ROM on the target hardware must be reprogrammed with an appropriate boot environment (note that it is also possible to save the original ROM as an image for use by guest systems). The bootable core and the virtual machine manager image must be installed in a secure partition.

[0044] Data in a non-volatile RAM must be set with appropriate hardware and security options. A security key for the machine must be generated and installed in the system. Perhaps the entire newly loaded partition will be encrypted.

[0045] At this point, the system is bootable into the machine of the virtual machine manager. It is possible for a target machine to inherit the machine configuration or for disk and programming information to be transferred from this machine to a target machine.

[0046] Profiles, which include the following, must be created for individual guest virtual machines:

a) An image to be installed as the boot ROM on the machine. This may be the ROM of the original machine, or it can be a ROM from another source if, for example, another system is emulated or developed.

b) A file containing the data to be supplied for non-volatile RAM requests. This can be initially empty or contain minimal configuration data appropriate for the virtual system.

c) Applications to be loaded into the ring 0 or ring 1 task area.

d) Designation of the disk area to be mapped to the disk drive of the virtual machine, and

designation of the protection blocks in the area (for example, disabling the writing to the boot sector).

e) Designation of dedicated hardware and external interrupt sources.

f) Access rights to dedicated I / O space addresses.

[0047] After these data are entered, the guest virtual machine can be started by mapping its boot ROM image and starting from the reset vector in virtual ring 0. At first boot, the machine appears like a new, unconfigured system. The parameters in the non-volatile RAM shadow image can be set using a boot ROM or a bootable removable media. FIG. 13 shows an exemplary machine image.

[0048] Each machine then formats its own disk space and installs the operating system. This makes the configuration system independent of the file system format (e.g., DOS or Linux). FIG. 5 shows an exemplary disk layout.

[0049] The auxiliary memory disk 60 has a partition table 102, a real environment partition and file system 104, a firewall partition 106, and partitions 108a through 108d dedicated to the respective virtual environments. The real environment partition 104 includes a virtual machine core image 104a, a virtual machine manager ring 0 task image 104b, another real environment privileged task image 104c, a data area 104d for the virtual machine manager 20, and a spool region 104e for the data transfer manager 30 (described later). The virtual machine manager data area 104d has specifications 140, a boot ROM image 150, and a nonvolatile RAM emulation image 160 for each virtual environment.

[0050] Under normal operating conditions, the hardware is booted from the boot ROM when power is turned on. This ROM sets the low-level machine parameters from data stored in non-volatile RAM or other sources, accesses the (non-removable) boot disk and loads the virtual machine core. Depending on the security model, the ROM may have built-in checksum and decryption functions to perform basic verification on the loaded system.

[0051] The virtual machine core initialization procedure loads the virtual machine manager as a ring 0 task and places this manager in the execution queue. Next, the core starts the virtual machine scheduler, which enters the machine of the virtual machine manager from the main entry point.

[0052] At this point, the virtual machine core executes the same set of “functions” as the original core for each virtual machine, not for each task. However, there are additional functions of

creating and deleting virtual machines. These functions are directly accessible only from codes in ring 0 and include the following:

a) Physical memory allocation and release.

b) Interrupt and trap handler installation.

c) Virtual machine creation and deletion.

d) Intertask communication between virtual machine environments using predetermined messaging protocols (only ring 0 and ring 1).

e) Implementation of critical sections by interrupt disabling service.

f) Scheduling and context switching between virtual machines.

g) Dispatching first level interrupts to registered handlers.

[0053] Communication primitives are used to pass data between virtual machines, but only through privileged programs. The virtual machine manager installs traps and interrupt handlers for external events, non-ring 0 accesses to I/O space addresses, and execution of privileged instructions. The virtual machine manager also includes drivers for accessing physical disks, keyboards, mouse and graphics devices, and drivers for reading non-volatile RAM and real-time clock information.

[0054] Within the “real” environment (outside the virtual machine environment), applications can access the actual file system containing system configuration information. Again, depending on the security model, applications running in this bootstrap environment can also access the disk area of the virtual machine. This is used, for example, to verify the contents of virtual sector 0 against a known stored copy or to check for known boot block viruses.

[0055] Thereafter, the virtual machine manager creates other guest virtual machines based on the configuration information in the core file system. These machines are added to the central scheduler execution list.

[0056] The virtual machine manager is responsible for creating and managing page tables for individual machines and managing shadow tables in the control registers of the machine. In this way, the central core does not need to know individual security policies or management strategies.

[0057] Individual machines start up in a state appropriate for their functions. For example, most protected mode machines boot from their reset (virtual) address but run in ring 2. A secure machine (e.g., a firewall machine) may be required to boot from a ring 0 application entry point.

[0058] All intended accesses to privileged operations and data are trapped by the virtual machine manager. The virtual machine manager emulates the result according to the policy set for that machine and the current state of the virtual machine's shadow control register.

[0059] External interrupts that are not inherited by other privileged machines are scheduled by the core with the virtual machine manager, which distributes these interrupts to the guest virtual machine in accordance with its state. With shared resources (e.g., keyboards), the virtual machine manager tracks the current "ownership" of the device and ensures that the correct input is sent to the correct device. For disk access, the data must be sent to (or obtained from) such machine that can access the disk space. Also, the apparent physical address must be mapped for each machine. Similarly, an "emulation" device such as a non-volatile RAM returns the appropriate data to the requesting machine.

[0060] In general, all applications operate transparently in a virtual environment. Each system appears to have only its own dedicated disk area. Data encryption and decryption are handled transparently. Devices mapped to the machine appear to be accessed through the normal interface (e.g., BIOS configuration menu).

[0061] The main motive for running virtual machines is for each environment to have its own protected and secured areas. An exemplary configuration would include the following four virtual machines:

a) An unsecure "open" machine that can access external networks and run software such as web browsers and email applications. Such machines may block outgoing emails to prevent an application stealing the data from sending the stolen data.

b) A low security file system that can be accessed by outgoing emails. Word processor applications and other virus targets may also be run there.

c) A highly secure machine that cannot be accessed from the outside and used for storing and procesing sensitive (or personally damaging) information.

d) A virtual machine that works as a firewall to the outside world and has direct control over the Ethernet interface. Such machine emulates simple mail transfer protocol (SMTP) transactions and allows other machines to "send" emails as long as they are locally spooled. However, an explicit user confirmation is required before the data leaves the machine (e.g., before data is sent to the outside from the physical machine or before data is sent between virtual machines via the virtual network adaptor).

[0062] Thus, access and damages are limited to the environment in which the individual programs are executed. While maintaining security with this configuration certainly requires some degree of user discipline, the isolation rules are effective in environments where procedures and policies are not only defined but also adhered to. In fact, by allowing users to create a "work" (play) environment, the overall security is improved while giving users a greater amount of freedom over their desktops but avoiding unnecessary data exposure.

[0063] This model is not limited solely to business use. For example, at home, one virtual machine can be dedicated just to running a "children's" web browser while an unsupervised version is accessible only in a virtual area that is protected by a different password. (Thus, data that is downloaded using a full-version is implicitly hidden from the eyes of others).

[0064] When a multi-machine virtual environment model is considered, there will be cases where data has to be exchanged between virtual machines. For example, an e-mail message received in an unsecure environment may have to be passed to a secured area. Such data transfer is handled through ring 1 applications.

[0065] A ring 1 application creates a "shadow drive" similar to that used by NFS (network file system) so that when an operation initiated by an application in a high security environment accesses a low security environment, the operation is trapped by ring 1. One advantage of this scheme is that virus checking can be performed through such an interface. By running a special application in ring 1, the transfer code uses a set of ring 0 gate calls to search and map data between machines.

[0066] A shortcoming of this approach is that the shared devices must be controlled through a driver that ensures that requests from different machines are processed correctly and with proper validity checks. Input/output accesses must be modified as necessary to provide, for example, a buffer for converting disk addresses and for reading and writing data.

[0067] Since the present invention is intended to be run on a secure small operating system, drivers must be ported accordingly. The level of effort required for maintaining drivers up-to-date for various hardware components cannot be underestimated especially in light of manufacturers not disclosing such information. An example is the nearly complete lack of support at the source level for Linux graphics card drivers. As in the case of XFree86, in many cases, manufacturers supply only binary versions of the drivers.

[0068] Depending on whether the required security is absolute or relative, trade-offs to improve

execution time can be considered. For example, by allowing the "current" virtual machine environment to directly access memory mapped apertures used for video buffering, direct writing to the monitor is made possible. However, allowing access to the control register of the same video card creates the (latent) possibility that an application may lock up the system.

[0069] The present invention also allows for more secure interaction when the computing device is embedded in an open access networking framework. One such application is in the active network area. In this case, data streams that flow through a computer network are processed as they flow through the computer network. An important feature of an active network is that the process that is performed is not fixed in advance, and data and methods can be dynamically loaded to the processing unit depending on the contents of the data stream that is processed. These programs themselves are transferred to the processor through the same open network as the data and are thus exposed to observation and interference by malicious third parties.

[0070] In a typical system, an application (e.g., a Java-enabled web browser) downloads a module from a trusted source over a public unsecured Internet. Public key encryption is used to increase security. When an application tries to open a secure channel to a module repository, an encrypted message is sent to the repository using a private key. The repository decrypts this message using the public key and, in response, sends the appropriate module (that has been encrypted with the public key). When the module reaches the application, it is decrypted with the private key.

[0071] Usually, a private key is held by one machine and is shared among applications that run on the machine. Since the private key is very important, the private key is often stored in the boot ROM or a non-volatile RAM or is derived from information stored therein. The problem with this conventional method is that a malicious application can also request the key, thereby enabling data snooping of the activities of other applications. Equally dangerous, a malicious application can steal a key and broadcast it elsewhere.

[0072] Creating a secure virtual environment prevents an application running in one virtual environment from snooping on an application running in another virtual environment, but does not prevent overt key theft.

[0073] To prevent key theft, when a request for a key (e.g., a call to a predetermined address) is made by an application running in a virtual environment, the request is trapped in the ring 0 core. In that case, instead of providing the machine's actual key, the virtual machine manager (or some

other real-world task) sends a message encrypted with the physical machine's key to the key server and requests a private key that can be used only one time or having just a short life.
[0074] The short-lived private key is then passed to the requesting application for use during the session. These public and private keys are unique to each virtual environment. Thus, [all applications] in one virtual environment have a key that is independent of all other virtual environments, and this key can be changed without the knowledge of the applications in the virtual environment. Furthermore, because the core hides the real machine's private key, this key is never visible from within a virtual environment and is therefore secure.
[0075]
[Embodiments of the Invention] The automatic data apparatus according to the present invention creates a security shell around a virtual machine environment and at the same time executes code directly from within the virtual environment.
[0076] As FIG. 2 shows, the device controls access privileges using a hierarchy of real privilege rings 0 through 3. These real privilege rings consist of "real," i.e., normal, processing environments of the device. FIG. 3(A) is a diagram showing the basic components of the present invention including a virtual machine environment, and FIG. 3(B) is a diagram showing the hierarchy of virtual rings in the virtual machine environment of (A). As shown in FIG. 3(A), in the real environment, there is a small core system 50 that implements a set of functions including task switching, resource management and interrupt dispatch. One such task is the virtual machine manager 20 (hereinafter "VM manager"), which is in the highest-privileged real ring and regulates all processes that change the properties of the real ring. Within the real environment, VM manager 20 creates a plurality of virtual machine environments 40. The virtual machine environments 40 are present in real privilege rings 2 and 3.
[0077] When viewed from within the virtual environment 40, everything has the illusion of being "real." However, calls to physical resources are actually handled by the VM manager 20. The VM manager 20 either directly traps such calls or masks the calls so that access is restricted. FIG. 4 is a diagram showing the memory partition dedicated to the virtual environment. For example, virtual environment 40 is located in memory portion 40' of memory 4b. Normally, a process executed in virtual environment 40 expects to see a range of consecutive addresses starting from address 0. However, the portion of the memory allocated to the virtual environment is unlikely to start from 0 and is not necessarily consecutive. To adjust for this, the

VM manager 20 uses a facility of the processor's memory management unit (MMU) to map the addresses of memory portion 40' so that everything looks normal from inside the virtual environment 40. Since such mapping is typically performed by a page memory management unit or a controller, the VM manager 20 preferably utilizes such tool when available.

[0078] Similarly, in the case of an auxiliary memory such as a hard drive, attempts to access an absolute block address from within the virtual environment 40 must be trapped and converted by the VM manager 20. Although a plurality of "virtual" drives is actually implemented on the same physical device, each of the drives appears to be a dedicated drive from within the virtual environment 40. This is also achieved by a transparent drive partitioning, but this is a feature not directly supported by Intel architecture PCs. However, if transparent partitioning is supported by hardware/firmware, it is preferable for VM manager 20 to use such tool to manage the "virtual drives."

[0079] Data transfer with a virtual environment is performed through data transfer manager 30 located in privileged ring 1. From the viewpoint of processes that are executed in the virtual environment 40, the data transfer manager 30 would appear to be a communication port such as an Ethernet network connection. If there are multiple virtual environments, communications between the environments are performed through the data transfer manager, and the virtual environments would recognize each other as separate entities that are connected by a communication medium (e.g., a physical local area network). When virtual environment 40 attempts to communicate with a machine outside the physical computer, information passes from the data transfer manager 30 to the outside world through a port or gateway 22. Examples of such a gateway include modem drivers and Ethernet drivers, which perform Ethernet connections.

[0080] The virtual environment 40 is divided into virtual privilege rings (FIG. 3(B)). There are usually as many virtual privileged rings as real rings, but since the virtual environment includes only the rest of the real rings (i.e., real rings 2 and 3), VM manager 20 has control of the descriptor table.

[0081] The first embodiment further has two types of descriptor tables. The first is a virtual environment descriptor table (VEDT). The VEDT is created by the operating system software running in the virtual machine and has a virtual ring number assigned to the segment by the system. As far as the virtual machines are concerned, the VEDT can be used as a GDT or an

LDT. The VEDT maps the information in the virtual environment to the virtual ring, not to the real ring and maps the segments, not to the real address, but to the mapped address of memory portion 40' that contains the virtual environment.

[0082] The additional second type of descriptor table is a shadow descriptor table (SDT). The SDT is generated by the VM manager and maps segments in the local environment to the hierarchy of real privileged rings. The SDT is used to determine the privileges when a processor is executing instructions originating from within the virtual environment. In practice, a unique SDT is generated for each VEDT that is used in the virtual environment. The SDT is selected based on the privilege with which the instruction is executed.

[0083] FIG. 6 is a diagram illustrating privilege rings and call gates for both the real environment and the virtual environment according to the first embodiment. In the first embodiment, as shown in FIG. 6, the remaining parts other than the real ring including the virtual environment are divided so that all the segments associated with virtual ring 0 in the VEDT are associated with real ring 2 in the SDT. Instructions executed in the virtual environment in virtual rings 1 through 3 are assigned to real ring 3 by SDT. This allocation of ring assignments is preferable because within virtual environment 40, the legacy OS is situated, in most cases, in virtual ring 0, as is the driver for processing input/output operations, for example. One of the situations where the call gates are most frequently used is in accessing these privileges. As described below, VM manager 20 must intervene when virtual environment call gate 118 (FIG. 6) is used to request access from a lower privileged compressed virtual ring to a higher privileged compressed virtual ring. By configuring virtual ring 0 as an uncompressed ring, the need for VM manager's intervention is reduced, thereby speeding up the processing.

[0084] As described in connection to FIG. 2, the privilege rules among virtual privilege rings in a virtual environment are the same as in the real environment. Instructions in the virtual environment can access information and resources associated with virtual rings with the same or lower privilege. Instructions that attempt to access a higher privileged ring must use one of the virtual call gates 118. These virtual call gates 118 appear in the VEDT, and the corresponding call gates are mapped in the SDT.

[0085] With the first embodiment, requests for information from a process in the virtual environment are determined using SDT instead of VEDT regardless of whether the privilege is higher, lower or the same.

[0086] FIG. 7 shows a VEDT 70 and four SDTs 80a through d. (For purposes of this explanation, the descriptor table of this figure shows only privilege information, but in practice these tables also include other information normally found in descriptor tables such as memory addresses.) SDTs 80a through d are each associated with one virtual privilege ring in the hierarchy of virtual privilege rings.

[0087] For example, when an instruction operating in virtual ring 0 requests information, the SDT of virtual ring 0 (80a) is used by the processor for descriptor information. The processor views this instruction as an instruction operating in real ring 2. Since the instruction is running in the highest virtual privilege ring, the instruction can access any information in the entire virtual environment 40. Thus, the VM manager 20 places real ring 2 into SDT0 (80a) for all segments in the virtual environment. When an instruction operating in virtual ring 0 requests access to a segment in the virtual environment, the processor compares the real privilege ring (real ring 2) where the instruction is operating against the real privilege ring that is recorded in SDT0 (80a). Since the privilege level is the same, the processor allows access.

[0088] On the other hand, an instruction operating in the virtual ring 3 must use the virtual call gate 118 to gain access outside virtual ring 3. For instructions operating in virtual ring 3, the processor uses SDT3 (80d) to determine privileges. In this case, as shown in VEDT 70, for all segments associated with virtual ring 3, VM manager 20 places real ring 3 in the corresponding SDT3 (80d) location. Thus, requests in the virtual privilege ring 3 become valid, and the processor grants access.

[0089] Within SDT3 (80d), VM manager 20 places real ring 2 in virtual ring 0 for all segments. When an instruction in virtual ring 3 requests access to virtual ring 0, the processor compares the real ring (real ring 3) of the instruction against the required privilege (real ring 2) and issues an exception. This is because this transition is invalid unless it is executed through a call gate. If the transition is attempted through an appropriate call gate, the transition is allowed.

[0090] The VM manager intervenes when access is requested from a compressed virtual ring to a higher privileged compressed virtual ring. Within SDT3 (80d), VM manager 20 places real ring 0 in the higher privileged compressed virtual ring (rings 1 and 2) for all segments. As before, an exception occurs if a transition is attempted from virtual ring 3 to virtual ring 1 or 2 without using a call gate. The execution of virtual environment call gate requests to real ring 0 is trapped by VM manager 20. This is similar to attempts to access a privileged operation or privileged

data from within the virtual environment 40. In these cases, the actual privilege level required for the memory segment in which the target address of the call resides is not sufficient for the privilege level of the virtual gate. This causes trapping by the VM manager 20 so that the VM manager 20 can change the virtual ring and switch the real SDT table to reflect the new virtual environment. The VM manager 20 then emulates the result based on the contents of the corresponding VEDT entry for the requested segment according to the policy set for that virtual machine.

[0091] The shadow descriptor tables (SDT) 80a through 80d are created by the VM manager 20 either one at a time or all of them at once when the virtual environment 40 is formed. With the first embodiment, the SDT formation policy is adaptively selected by the VM manager 20 based on the known or observed behavior of the virtual environment 40.

[0092] The simplest policy is to create one SDT for each virtual privilege ring whenever a process starts executing in that ring. This SDT is modified when the location in memory portion 40' of a segment changes (or is deleted or added) or when some other change occurs in VEDT but the running process does not change the access privilege. When a running process changes the privilege ring, the SDT is discarded, and a new SDT is generated. If changes to the virtual rings are frequent, this policy will increase the overhead but provides the advantage of minimizing the number of SDTs that are maintained.

[0093] The second SDT policy is to create SDTs for the entire virtual ring hierarchy when the virtual environment 40 is formed. Every time a change occurs in VEDT, all SDTs are modified. This policy has the advantage of a high-speed transition between the virtual rings but also increases the number of SDTs that are maintained. This policy is best suited to situations where processes in a virtual environment frequently change the rings.

[0094] The third SDT policy, a hybrid of the above two, calls for creating one SDT at a time as needed and maintaining and reusing an SDT that is once created.

[0095] When the VM manager 20 recognizes a legacy operating system in virtual environment 40, an SDT policy suitable for the properties of the legacy OS is executed. If the contents of the environment are not recognized, the VM manager 20 selects a default policy (for example, the first policy in the first embodiment) and switches the policy according to the indicated efficiency.

[0096] In systems with multiple descriptor tables, each descriptor table must be shadowed. This

is true even when there are both global and local tables. In a typical implementation of the present invention, the global table is changed when the VM manager 20 switches between virtual machines, and local tables are changed when the operating system of the virtual machine switches between tasks.

[0097] FIG. 8 is a diagram illustrating privilege rings and call gates for both the real environment and the virtual environment according to a second embodiment. With the second embodiment of the present invention, the virtual environment has only two virtual rings. As an example, in an Intel architecture machine, a standard DOS implementation uses only two rings: the highest and lowest privilege rings. When the VM manager 20 recognizes such legacy operating system, virtual environment 40 is formed using "quick-pass-through." As FIG. 8 shows, the two virtual rings are directly mapped to the two real rings, which include virtual environment 40. As a result, if the shadow descriptor table (SDT) policy that is used requires maintaining SDTs for all rings, only two SDTs need to be maintained. For this reason, with the second embodiment, a preferable default is an SDT policy where all shadow descriptor tables are formed when the environment is created.

[0098] With the third embodiment of the present invention, instead of using a shadow descriptor table (SDT), the VM manager 20 modifies the virtual environment descriptor table (VEDT) to replace the association with virtual rings with the association with the corresponding real privilege rings. When instructions are executed from within a virtual environment, the processor uses VEDT to determine the access privileges.

[0099] To prevent a code from within the virtual environment from recognizing this replacement, the memory containing VEDT is marked as not readable from within the virtual environment. When an attempt is made to read VEDT from inside the virtual environment 40, the attempt is trapped by the VM manager 20. The VM manager 20 masks the real privilege replacement with an appropriate virtual environment information.

[0100] VEDTs are modified in one of two ways. With the first method, when an attempt is made to write to VEDT from inside the virtual environment 40, the attempt is trapped by VM manager 20. The consequnce of this is for the VM manager to intercept the intended entry and substitute it with a replacement entry. The first method is preferable when the guest OS does not attempt to confirm (requires reading) the VEDT entry. The second method allows instructions in the virtual environment 40 to write to the VEDT and the entry is later overwritten by the appropriate

real privilege ring that is assigned. If a guest OS confirms the entry after the entry is written, attempts to read the VEDT is used as a trap that instructs the VM manager to perform an overwrite. It is preferable to observe the environment during operation and to select a method that minimizes the number of traps per VEDT modification.

[0101] Continuing with the description of the embodiment of the present invention, information transfers with the virtual environment are handled by a data transfer manager (DTM) operating in real ring 1. To manage the data stream, the DTM spools the data in a storage area as necessary (FIG. 5, DTM spool 104e). FIG. 9 is a diagram illustrating an exemplary configuration for communication with the virtual machine environment using the DTM 30. The particular configuration that is used is based on factors such as the user's initial settings, hardware capabilities and machine requirements. The gateway 22 includes a ring 0 communication driver and a data connection.

[0102] In the first configuration illustrated in FIG. 9(A), the virtual machine 40 regards DTM 30 as a local hub 34. To ensure that only complete transactions are communicated between virtual environments or to verify the transaction contents, DTM 30 also implements a spooling system for buffering multiple data blocks. The hub 34 can be implemented using whatever communication protocol is required. If the user so desires, because spooling is typically used, it is possible to simulate different protocols for different virtual environments, in which case DTM 30 performs the switching transparently.

[0103] With the second configuration shown in FIG. 9(B), DTM 30 simulates a communication channel (xfer) 32 for each virtual environment. Examples of channels include modem connections and virtual file system services such as NFS. The channel can look like a real network to which the ring 0 gateway driver 22 is actually connected or can simulate something else. For example, the gateway can be an Ethernet TCP/IP link, but for some reason, a user may have the DTM 30 simulate a modem.

[0104] With the third configuration shown in FIG. 9(C), a plurality of DTMs 30 is operated and is managed as different tasks. In this case, the DTM 30 must have a shared spooling area for spooling among virtual machines. This configuration is appropriate when the user needs various environment-specific communication functions. This is because it is preferable to provide a customized ring 1 manager.

[0105] With the DTM configuration (FIG. 9), a choice must be made between security and

speed. The fastest configuration is where data flows directly from the gateway 22 to the virtual machine 40 without involvement by the DTM 30. This presents the risk of disintegrating the sandbox because this generally enables interactions between the gateway 22 and the virtual environment 40. Protocol emulation with DTM 30 obviously slows down the operation. The safest but the slowest configuration is to spool all data and check (by way of DTM30 and other real environment applications) for viruses and for transmission by data stealing applications. It is also possible to use a combination of a plurality of configurations based on the hardware communication capability (for example, a plurality of data connections) and the trust given to each virtual environment by the user.

[0106] Continuing with the explanation of the embodiments of the present invention, the system can be set up to include a firewall operating from its own virtual environment. FIG. 10 is a diagram illustrating three exemplary configurations of the present invention where a firewall operating from within a virtual environment is used. The firewall software can be custom developed for the architecture of the present invention or can be a commercially available firewall software for stand-alone machines. Each firewall has two sides: inside and outside. The environment to be protected is inside the firewall. Outside the firewall are networks and untrusted environments. Communications on either sides of the firewall are isolated from each other.

[0107] The three configurations shown in FIG. 10 are modifications of the data communication configuration described with reference to FIG. 9. In FIG. 10(A), the virtual machine containing firewall 42 has separate DTMs 30 to link the inside and the outside. In FIG. 10(B), the DTM 30 simulates separate channels 32 to link the inside and the outside. In FIG. 10(C), the DTM 30 simulates the local hub 34 inside the firewall while the dedicated DTM 30 is connected to the gateway 22. These configurations are merely examples, and many other combinations are obviously possible. A requirement that must be complied with is the isolation between the inside and outside of the firewall 42. Since the firewall 42 is a simple requirement, an operating system that is compact but has a strict security is preferred for environments with a firewall.

[0108] The virtual environment including the firewall 42 is an example of an environment in which a direct data stream from the gateway 22 is desired depending on the trust that can be given to the firewall 42. Further, depending on the level of trust, it may be preferable to reduce the security restrictions, such as data stream scanning for data theft, imposed on DTM 30 inside

firewall 42 to avoid redundant security restrictions.

[0109] As an addition to the embodiments of the present invention, the DTM 30 or firewall 42 is configured to require user confirmation before data transfer is permitted. While the user confirmation is pending, the data is stored in a spool (FIG. 5, DTM spool 104e or firewall spool 106a). This feature is particularly useful for detecting data theft.

[0110] FIG. 11 is a diagram illustrating an exemplary configuration for encrypting — from within a virtual environment — data that is stored in a disk partition. Continuing the description of the embodiment of the present invention, data is encrypted between the virtual machine 40 and the virtual machine partition 108 in the auxiliary memory 60. This encryption is invisible from within the virtual machine 40 and preferably includes all data in the virtual machine partition. By combining this encryption with a key unique to the physical machine, all data on a stolen hard drive is better protected from the peering eyes of others. Providing a password protection when the virtual machine environments are created further improves security. This protection can be obtained without making changes to the legacy operating system or without any indication in the virtual environment that the data is encrypted.

[0111] Encryption is performed by the VM manager 20 (FIG. 11(A)), which includes an encryption portion 90 and a decryption portion 92, or by a separate encryption/decryption filter 94. Filter 94 is implemented as a layer through which all data passes or as a collaborative "helper" that supports privileged real environment applications. Filter 94 can be a specialized software (e.g., operating as an application in ring 0 or 1 (FIG. 11(B)) or a specialized hardware (e.g., a fusible ROM). Furthermore, it is possible to use different encryption schemes for different partitions. A similar method is used for removable storage devices and, in the most extreme cases, is used to encrypt the virtual environment portion 40' of the main memory (this is also useful with systems using a non-volatile main memory).

[0112] Similarly, encryption is also performed on data streams entering and exiting the virtual machine. For example, FIG. 12 is a diagram illustrating an exemplary configuration for encrypting and decrypting data transfers between a virtual environment and an external network. In FIG. 12(A), an encryption portion 90 and a decryption portion 92 are incorporated within data transfer manager 30. In FIG. 12(B), the data passes through the encryption/decryption filter 94.

[0113] As another embodiment, the present invention is useful for protecting applications that run on the components of an active network. The security requirements for such a configuration

are different from models that are used in environments where a higher degree of mutual trust is assumed. This is indicated, for example, by the limitations of the Kerberoskey distribution scheme for operations that take place in an unsecure infrastructure, as described in various literature. Data security is improved when the above embodiments are incorporated in an active network environment. As an example, FIG. 14 shows a portion of a reprogrammable network which comprises:

- An active network component (ANC) 200 that can execute downloadable modules supplied by other networking elements and is reachable through a public, unsecured Internet;
- Active repository (AR) 210 that includes downloadable modules;
- KServ1 (230a) and KServ2 (230b), which are "key servers" in the network; and
- Application-specific data (ASD) 240 (e.g., billing data), which is accessible only by active components with proper authorization.

The ANC 200 comprises a machine that includes a virtual environment 40 that requires an encryption key.

[0114] With such configuration, there are two different security areas. The first security area is for the authentication of the ANC machine in the active network. The second security area is for the verification of active components loaded into ANC 200. If there is no secure environment on the ANC 200 as provided by the present invention, the first security cannot be achieved. Furthermore, the second security cannot be guaranteed if the downloaded component is modified by a malicious third party while in transit or on the ANC 200. Described below is how both levels of authentication can be performed using the well-known two-key encryption method with the secure configuration of the present invention.

[0115] According to the present invention, an active element is represented by Name. Name can represent a machine or an application. One may think of a name space with a hierarchical structure as a domain such as DNS (domain name system, distributed Internet directory service), but an active Name for a machine need not be related to the Internet configuration. (In other words, Name is an abstract entity that may be associated with different physical instances at different times).

[0116] Each Name is associated with Clef (public key) known to the outside world and Key (private key) known only to the entity that instantiates the name. Data that is encrypted by Key according to a usual model can be encrypted only with Clef, and the opposite is also true.

[0117] When the ANC machine 200 is set, an authorization ternary set {Name, Clef, Key} is assigned. Also provided are at least one access binary set {Name, Clef} and the corresponding IP information {machine name, address} of the key server machine 230a/b (this may be used by the ANC machine 200 for key inquiry). At the same time, the access binary set {Name-ANC, Clef-ANC} of the ANC machine 200 is input to the key server 230a. All operations related to private key generation and installation are performed securely.

[0118] The virtual environment 40 receives a unique Clef (public key) together with a private key (secret key). This key is a use-limited key (Key-ANC) requested by the ANC 200 to the key server 230a. Preferably, the initial request is made when an application in virtual environment 40 calls the address associated with the machine's private key, and the call is automatically trapped in ring 0 core. Requests made by the core for a limited use private key using real environment encryption are encoded using the private key of the actual hardware. When the key server responds, the limited use private key (Key) is passed to the virtual environment (as if it were located at the address that was called), so that its internal applications can use the limited use key to request modules to active repository 210. As is the standard practice with such a scheme, information encoded with the public key can only be decrypted using the private key, and information encoded with the private key can only be decrypted using the public key.

[0119] The acquisition of the limited use private key is performed outside the knowledge of the applications in the virtual environment 40. Once the key is obtained, data streams entering and exiting the virtual environment 40 can be independently encoded, thus not exposing the encoded applications to security risks arising from malicious codes executed in a different environment. This is because it is impossible for a private key of the applications' environment to be exposed to any risk.

[0120] Using some control mechanism (for example, CORBA (common object request broker architecture)) (but implementing the following protocol), the active repository (AR) 210 is instructed to download a new program to the virtual environment that is instantiated by ANC 200. The AR 210 is notified of the network location of ANC 200 or determines it by some secure location and instantiation mechanism. AR 210 becomes aware of the network address of ANC 200 in some way. AR 210 also becomes aware of the network address of key server 230a.

[0121] The following is an example of the transactions required to securely download a module after a limited use key is obtained.

[0122] 1. AR210 sends the following request to KServ1 (230a).

<Name-AR < Clef (Name-ANC) =? > $_{\text{Key-AR}}$ > $_{\text{Clef-KServ1}}$

where <y>$_x$ means item y encrypted using key x, and ? identifies the data to be returned in response to this request. Under the assumption that the private key remains truly secret, no entity other than KServ1 (230a) can see the contents of this message.

[0123] 2. When KServ1 (230a) receives this packet, it executes the following operation.

<< Name-AR << Clef (Name-ANC) =? >$_{\text{Key-AR}}$>$_{\text{Clef-KServ1}}$>$_{\text{Key-KServ1}}$

Since << y>$_{\text{Key}}$>$_{\text{Clef}}$ = y, this is simplified as follows.

Name-AR <Clef (Name-ANC) =? >$_{\text{Key-AR}}$

Using the public key of Name-AR yields the following equation:

<< Clef (Name-ANC) =? > $_{\text{Key-AR}}$ > $_{\text{Clef-AR}}$

This is simplified as follows.

Clef (Name-ANC) =?

This ensures that the request is obtained only if the packet is from the claimed Name entity.

[0124]

3. Next, the key server 230a checks Clef for Name-ANC and forms a response encrypted with its own key and the Clef of AR 210 as follows.

<Name-KServ1 <Clef (Name-ANC) = Clef-ANC>$_{\text{key-KServ1}}$>$_{\text{Clef-AR}}$

This ensures that only AR 210 can read this response. At first glance, it may seem strange to encrypt what is actually public information. However, this means that even a public key is disclosed only to authorized entities, and this improves security somewhat. More importantly, it prevents a person snooping “near” the key server 230a from inferring that the client of AR 210 is interested in ANC machine 200. At this level, even a trivial thing can allow a competitor to make inferences or to obtain a business advantage.

[0125] 4. AR 210 performs an obvious decryption and checks that Name in the response corresponds to a known key server before finally obtaining the access code of ANC machine 200.

[0126] If a new program that is sent to ANC 200 requires a secure access (e.g., access to ASD 240) to sensitive information from the software supplier but ASD 240's provider does not trust the security of AR210 long-term storage against snooping (or perhaps ASD 240's provider does not know in advance the set of Names for ANC where the application will be loaded), AR 210

sends a second inquiry such as the following to key server (230b) that is trusted by both AR 210 and ASD 240 to request a new ternary set.

[0127] 5. <Name-AR <New (Name-SecApp) =? >$_{Key\text{-}AR}$>$_{Clef\text{-}KServ2}$

where Name-SecApp is the Name of the application that will be instantiated on ANC 200.

[0128] 6. KServ2 (230b) extracts the request using Clef-Ar (which may have to be obtained from KServ1 (230a)) and generates a new authentication ternary set. KServ2 (230b), which stores the Name part and the Clef part in its database, returns the complete ternary set to AR.

<Name-KServ2 <{Name-SecApp, Clef-SecApp, Key-SecApp}>$_{Key\text{-}KServ2}$>$_{Clef\text{-}AR}$

Needless to say, it is important here to encrypt this response. In a stronger model, it is possible to further encrypt using Clef-ANC to prevent AR 210 from reading the data.

[0129] 7. AR uses the key returned from KServ1 (230a) and sends the reprogramming information to ANC 200.

<Name-AR <New Program>$_{Key\text{-}AR}$>$_{Clef\text{-}ANC}$

[0130] 8. This packet is intercepted at the "firewall" level 42 of ANC 200 and is passed to the authorized virtual machine environment 40 with access to the limited use Key-ANC.

<< Name-AR << New Program >>$_{Key\text{-}AR}$>$_{Clef\text{-}ANC}$>$_{Key\text{-}ANC}$

This is simplified as follows.

Name-AR <New Program>$_{Key\text{-}AR}$

At this point, the name of the transmitting entity is revealed (Name-AR). Depending on the security model, this Name is checked against an authorized list or is accepted at face value in open systems.

[0131] 9. Using exactly the same steps as steps 1 through 4, ANC 200 acquires Clef-AR and recovers New Program information.

[0132] So that the new program (New Program) is securely linked to the repository, AR 210 sends the authorization ternary set (possibly along with the name of KServ2 (230b) and its location information) to ANC 200. AR 210 is trusted not to keep a copy of this information (ultimately, AR 210 is trusted not to modify the stored application). Depending on the model, this new application-specific key can be passed to the newly loaded code or kept outside of the virtual environment. Using Key-SecApp and Clef-SecApp, a new application can establish a communication channel to ASD 240 through KServ2 (230b) without requiring ASD 240 to know Clef-ASD. This mechanism allows KServ2 (230b) and ASD 240 to revoke permissions

independently of the trust placed in ANC 200.

[0133] It should be noted that this model does not require that individual applications running in virtual environment 40 be configured in a special way or be written in a special language. Furthermore, it is also possible to use the same methods to provide a unique key set to the encryption subroutine (FIG. 12) that services the virtual environment. For example, it is possible for DTM 30 or the encryption/decryption filter 94 to use a limited-use key to encrypt and decrypt the data transfers in and out of the virtual environment 40. In such a case, unencrypted data flowing between virtual environment 40 and DTM 30 or filter 94 is inaccessible to all other virtual environments.

[0134] Needless to say, various modifications to the present invention are conceivable without deviating from the technical philosophy and technical scope of the present invention.

[Brief Description of the Figures]

FIG. 1(A) is a diagram showing a segment and descriptor table stored in a usual memory, and (B) is a diagram showing the contents of the descriptor table in (A).

FIG. 2 is a schematic diagram showing privilege rings and call gates in a machine using a hierarchical privilege protection scheme.

FIG. 3(A) is a diagram showing the basic components of the present invention including a virtual machine environment, and (B) is a diagram showing a hierarchy of virtual rings within the virtual machine environment of (A).

FIG. 4 is a diagram illustrating a partition of a memory dedicated to a virtual environment.

FIG. 5 is a diagram showing the disk partition layout of an auxiliary memory.

FIG. 6 is a diagram showing privilege rings and call gates for both the real environment and the virtual environment of the first embodiment.

FIG. 7 is a diagram showing a virtual environment descriptor table and a shadow descriptor table according to the first embodiment.

FIG. 8 is a diagram showing privilege rings and call gates for both a real environment and a virtual environment according to the second embodiment.

FIG. 9 shows an exemplary configuration for communicating with a virtual machine environment.

FIG. 10 shows an exemplary configuration of the present invention that uses a firewall operating from within a virtual environment.

FIG. 11 is a diagram showing an exemplary configuration for encrypting data stored in a disk partition from within a virtual environment.

FIG. 12 is a diagram showing an exemplary configuration for encrypting and decrypting data that is transferred between a virtual environment and an external network.

FIG. 13 shows an exemplary machine image.

FIG. 14 shows an exemplary network layout for active network computing.

[Legend]

4a. Memory

6a through 6e. Segment

8. Descriptor table

10a through 10e. Descriptor

20. Virtual machine manager

22. Gateway

30. Data transfer manager

32. Communication channel

34. Local hub

40. Virtual environment

42. Firewall

50. Core system

60. Auxiliary memory disk

70. VEDT

80. SDT

90. Encryption part

92. Decryption part

94. Encryption/decryption filter

102. Partition table

104. Real environment partition and file system

104a. Virtual machine core image

104b. Ring 0 task image of virtual machine manager

104c. Image of another real environment privilege task

104d. Data area of virtual machine manager

104e. Spool area of data transfer manager

106. Firewall partition

106a. Firewall spool

108a through 108d. Dedicated virtual environment partition

118. Virtual environment call gate

140. Specifications

150. Boot ROM image

160. Non-volatile RAM emulation image

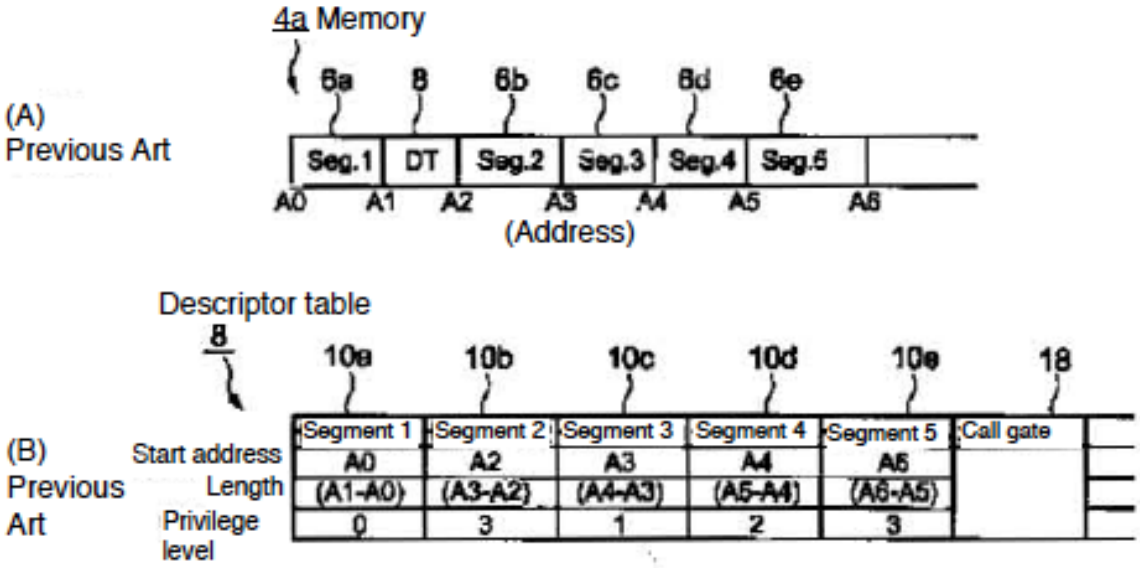
200. Active network component (ANC)

210. Active repository (AR)

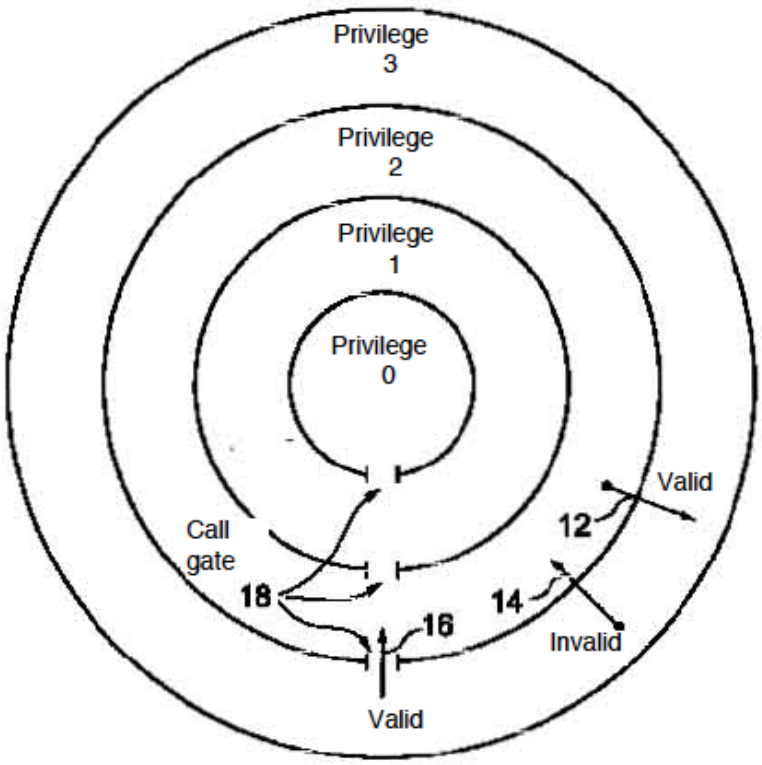
230. Key server

240. Application specific data (ASD)

[FIG. 1]

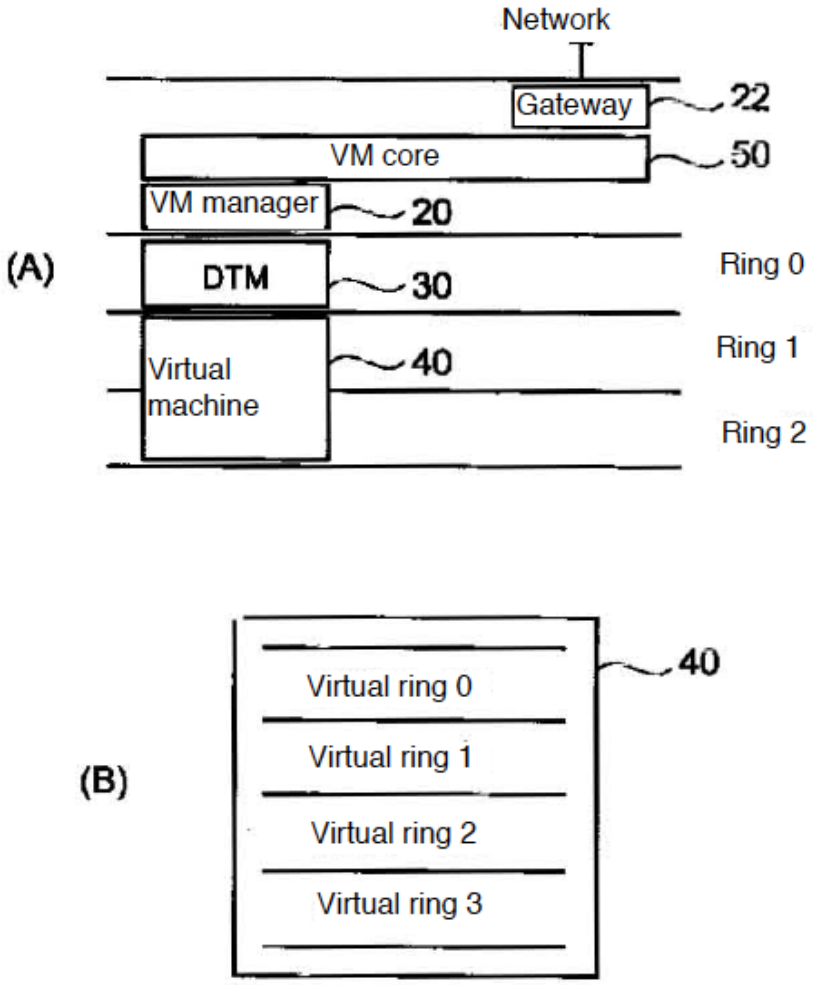


[FIG. 2]

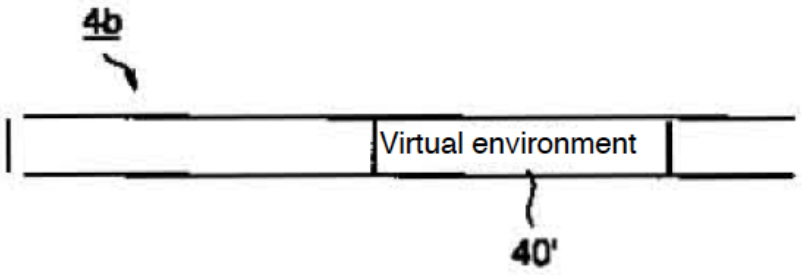


Previous art

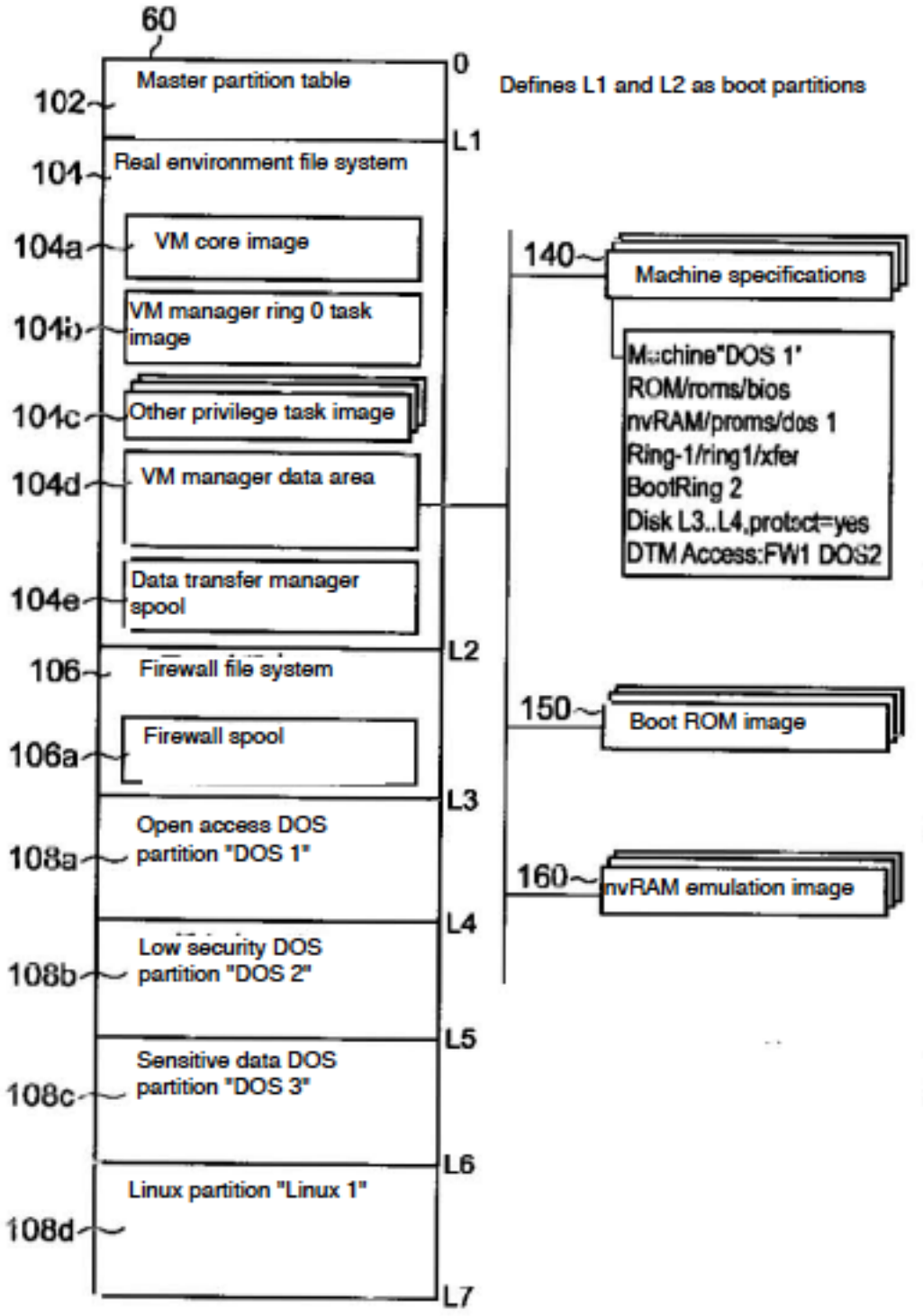
[FIG. 3]


Network
Gateway
22
VM core
50
VM manager
20
(A)
DTM
30
Ring 0
Virtual machine
40
Ring 1
Ring 2
(B)
40
Virtual ring 0
Virtual ring 1
Virtual ring 2
Virtual ring 3

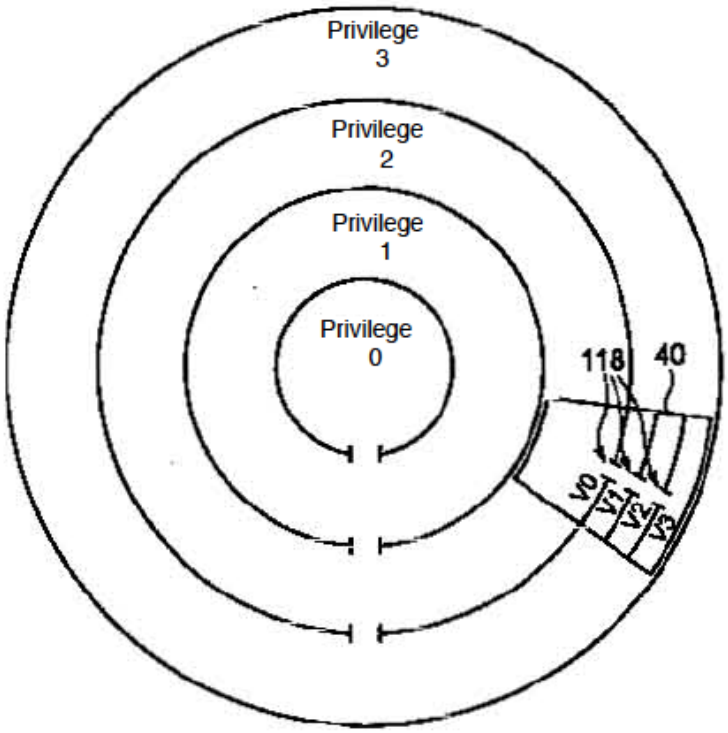
[FIG. 4]


4b
Virtual environment
40'

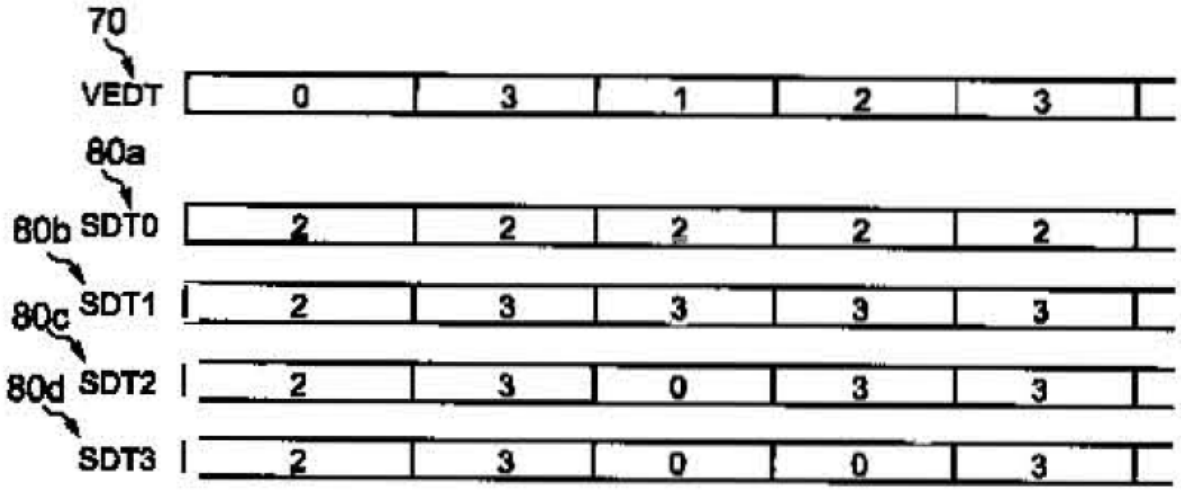
[FIG. 5]


60
Master partition table
102
0
Defines L1 and L2 as boot partitions
L1
Real environment file system
101
VM core image
104a
VM manager ring 0 task image
104b
Other privilege task image
104c
VM manager data area
104d
Data transfer manager spool
104e
L2
Firewall file system
106
Firewall spool
106a
L3
Open access DOS partition "DOS 1"
108a
L4
Low security DOS partition "DOS 2"
108b
L5
Sensitive data DOS partition "DOS 3"
108c
L6
Linux partition "Linux 1"
108d
L7
140
Machine specifications
M:chine"DOS 1"
ROM/roms/bios
nvRAM/proms/dos 1
Ring-1/ring1/xfer
BootRing 2
Disk L3..L4,protect=yes
DTM Access:FW1 DOS2
150
Boot ROM image
160
nvRAM emulation image

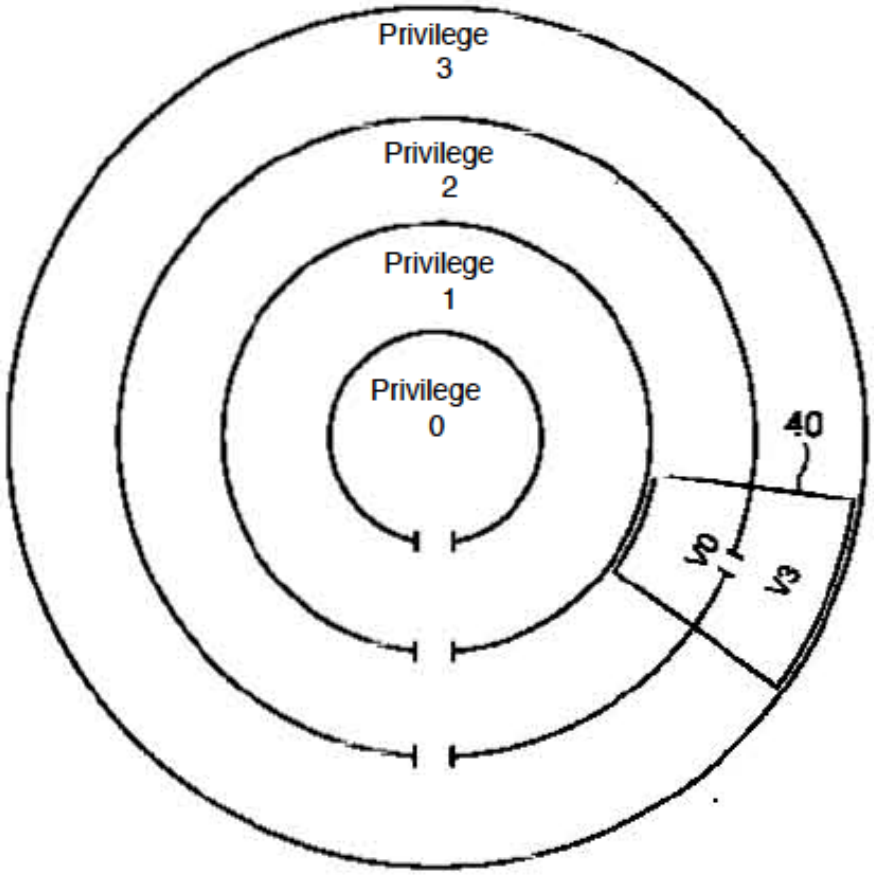
[FIG. 6]


Privilege 3
Privilege 2
Privilege 1
Privilege 0
118
40
V0
V1
V2
V3

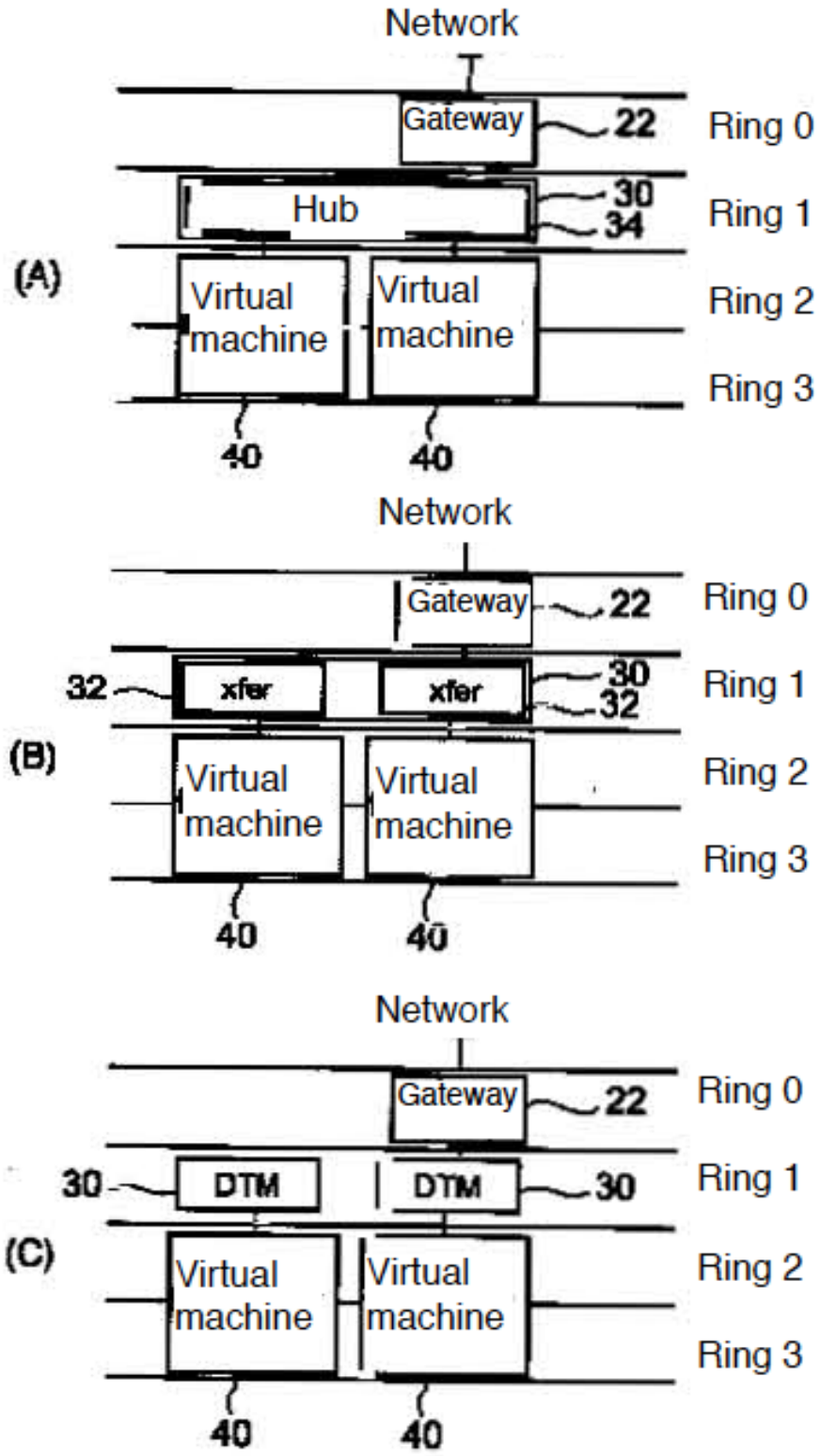
[FIG. 7]


70
VEDT
0 3 1 2 3
80a
80b SDT0
2 2 2 2 2
80c SDT1
2 3 3 3 3
80d SDT2
2 3 0 3 3
SDT3
2 3 0 0 3

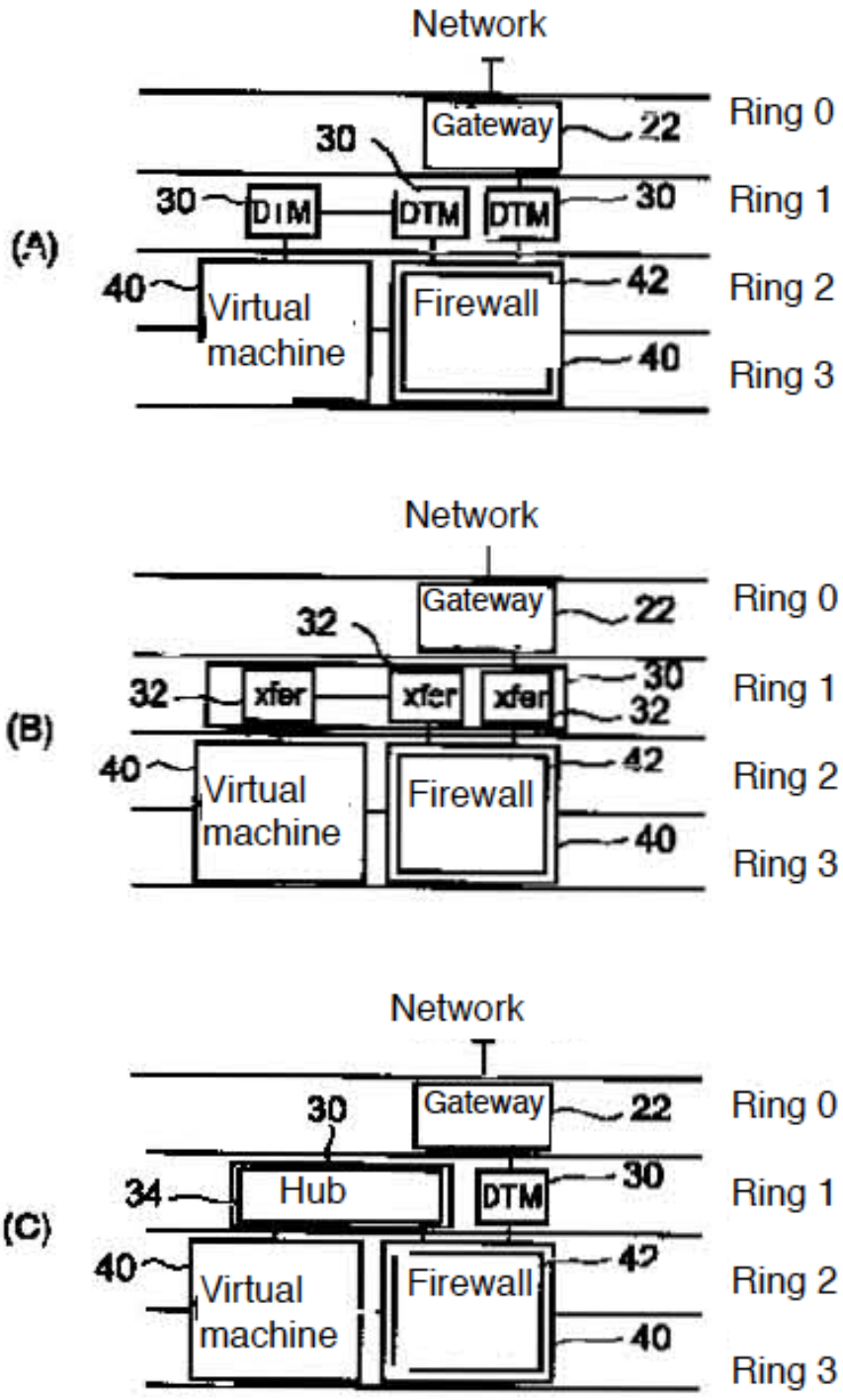
[FIG. 8]



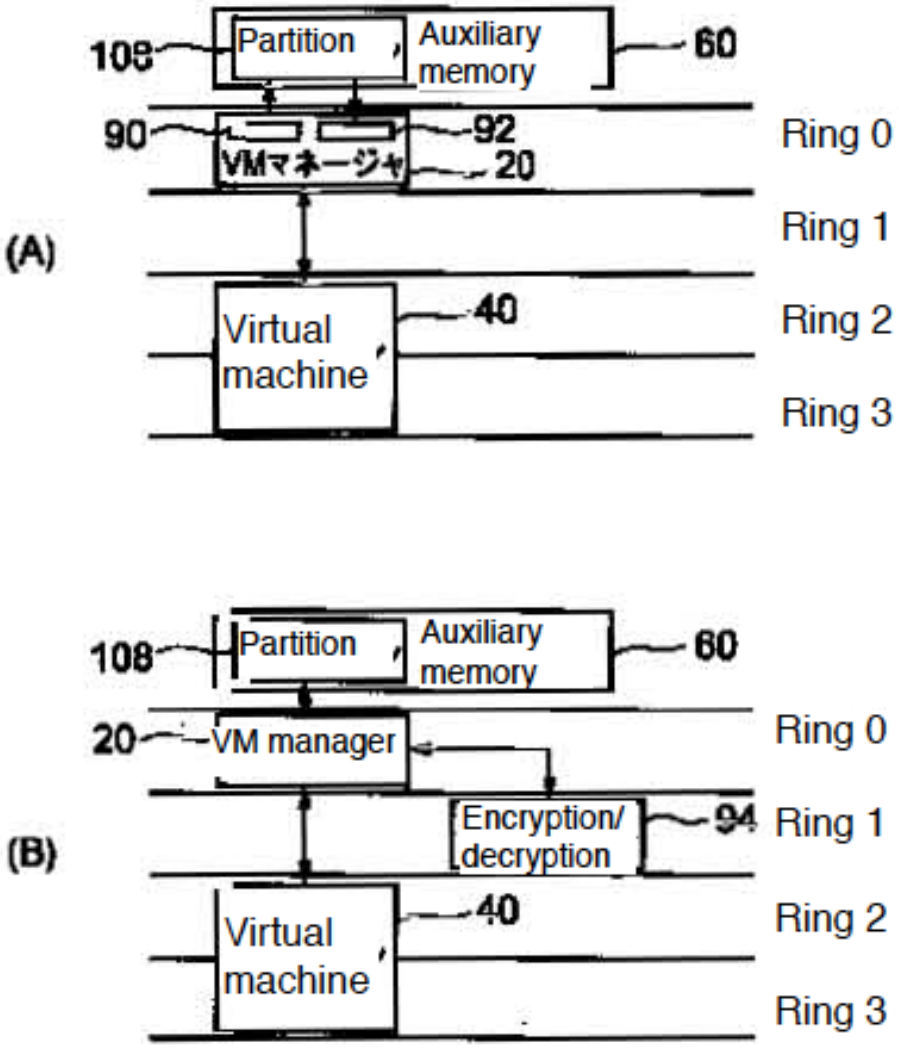
[FIG. 9]


Network
Gateway
22
Ring 0
Hub
30
34
Ring 1
(A)
Virtual machine
Virtual machine
Ring 2
Ring 3
40
40
Network
Gateway
22
Ring 0
32
xfer
xfer
30
32
Ring 1
(B)
Virtual machine
Virtual machine
Ring 2
Ring 3
40
40
Network
Gateway
22
Ring 0
30
DTM
DTM
30
Ring 1
(C)
Virtual machine
Virtual machine
Ring 2
Ring 3
40
40

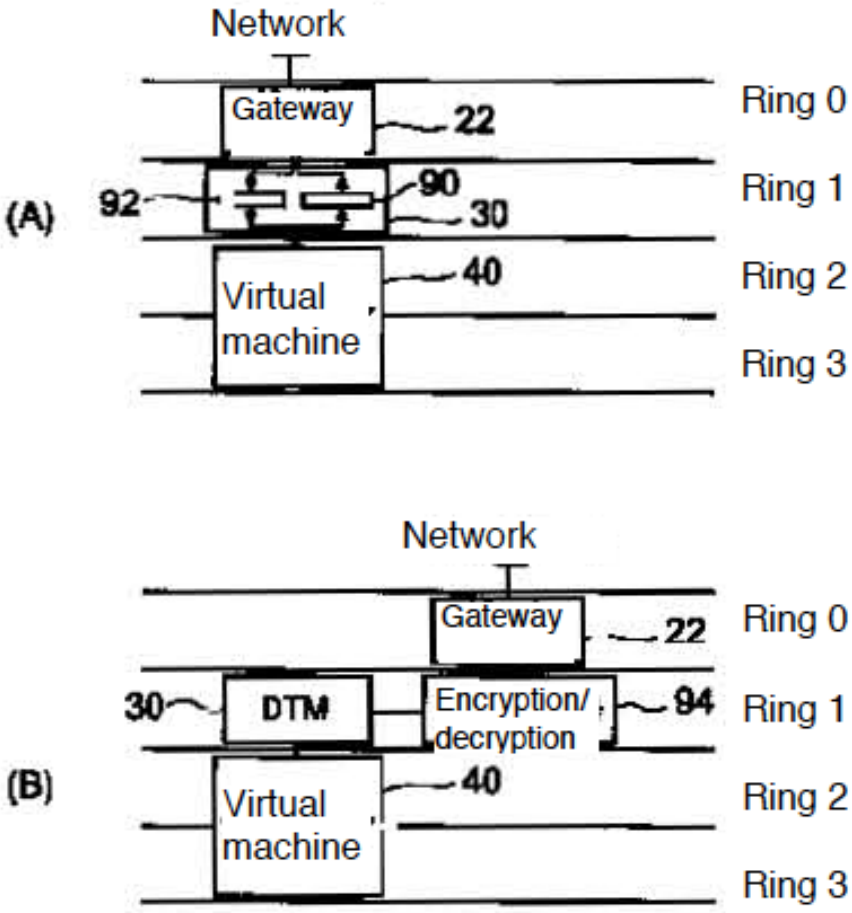
[FIG. 10]


Network
Gateway
22
30
Ring 0
30
DIM
DTM
DTM
30
Ring 1
(A)
40
Virtual machine
Firewall
42
Ring 2
40
Ring 3
Network
Gateway
22
32
Ring 0
32
xfer
xfer
xfer
30
32
Ring 1
(B)
40
Virtual machine
Firewall
42
Ring 2
40
Ring 3
Network
30
Gateway
22
Ring 0
34
Hub
DTM
30
Ring 1
(C)
40
Virtual machine
Firewall
42
Ring 2
40
Ring 3

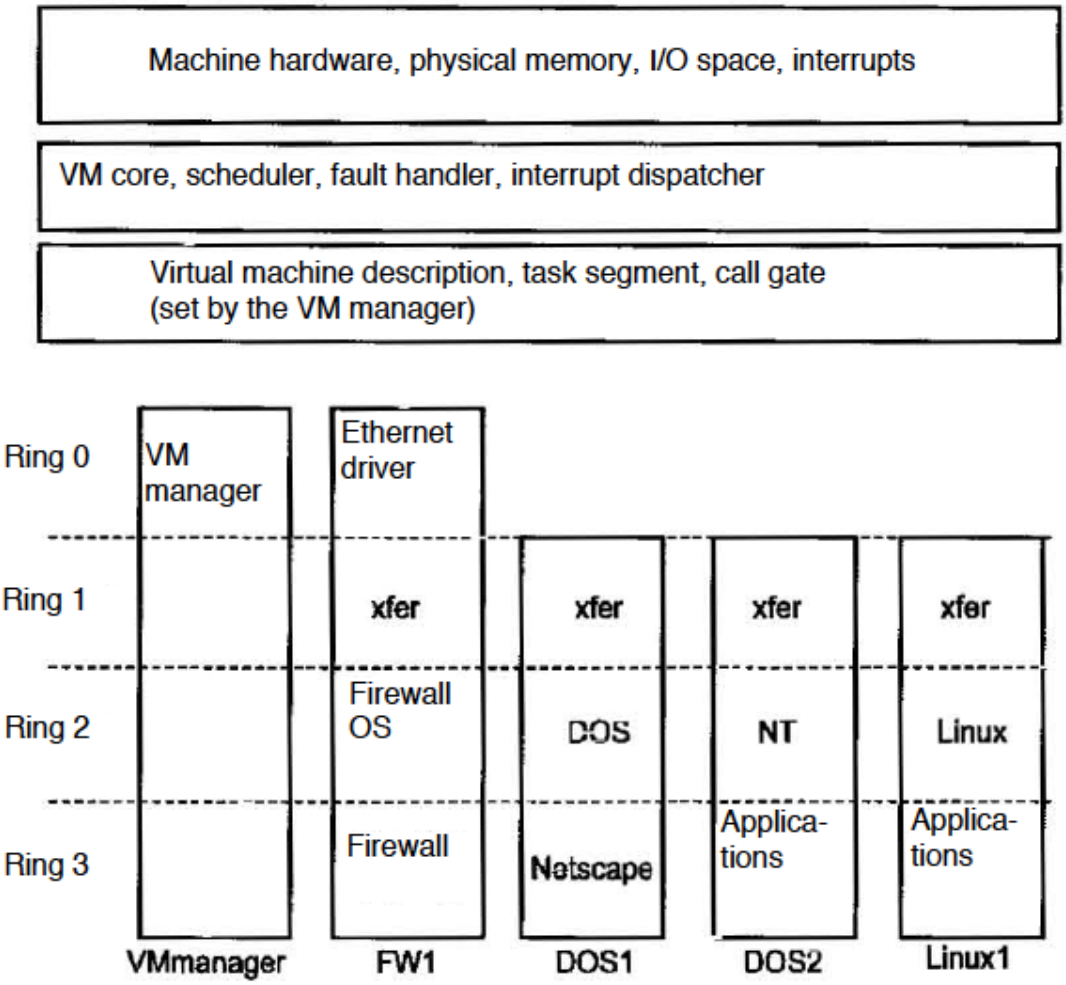
[FIG. 11]


(A)
108
Partition
Auxiliary memory
60
90
92
VMマネージャ
20
Virtual machine
40
Ring 0
Ring 1
Ring 2
Ring 3
(B)
108
Partition
Auxiliary memory
60
20
VM manager
Encryption/ decryption
94
Virtual machine
40
Ring 0
Ring 1
Ring 2
Ring 3

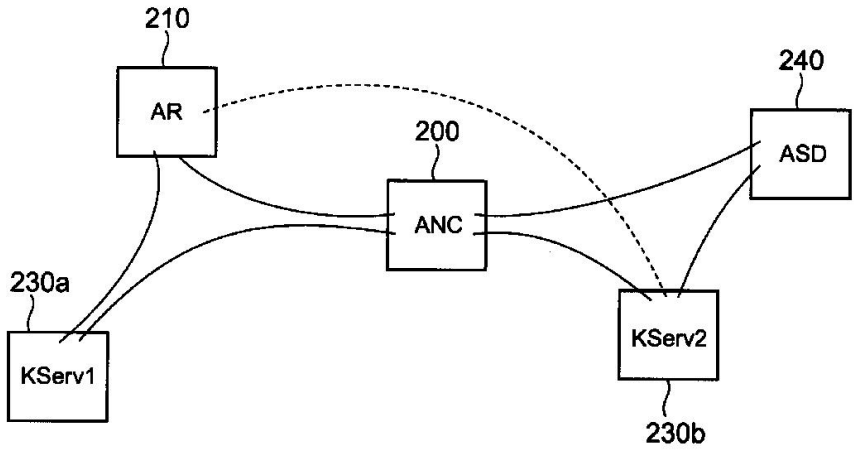
[FIG. 12]


Network
Gateway
22
Ring 0
Ring 1
92
90
30
(A)
Virtual machine
40
Ring 2
Ring 3
Network
Gateway
22
Ring 0
30
DTM
Encryption/ decryption
94
Ring 1
(B)
Virtual machine
40
Ring 2
Ring 3

[FIG. 13]

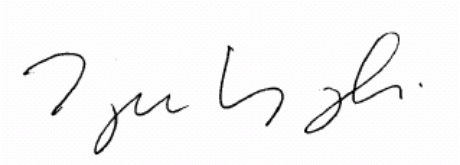

Machine hardware, physical memory, I/O space, interrupts
VM core, scheduler, fault handler, interrupt dispatcher
Virtual machine description, task segment, call gate
(set by the VM manager)
Ring 0
Ring 1
Ring 2
Ring 3
VM manager
Ethernet driver
xfer
Firewall OS
Firewall
xfer
DOS
Netscape
xfer
NT
Applica-tions
xfer
Linux
Applica-tions
VMmanager
FW1
DOS1
DOS2
Linux1

[FIG. 14]



Declaration of Toyu Yazaki

1. I, Toyu Yazaki, residing in Santa Rosa, California, hereby certify that I am competent to translate from Japanese into English.
2. I hereby certify that I prepared the translation appearing as JP2001-318797A and that it is, to the best of my knowledge and belief, a true and accurate English translation of JP2001-318797A.
3. I declare under penalty of perjury that all statements made herein of my own knowledge are true and all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment or both, under Section 1001 of Title 18 of the United States Code.



Executed on: August 26, 2020 Signed: _____
Toyu Yazaki