US 20090083829A1

(19) United States

(12) Patent Application Publication

(10) Pub. No.: US 2009/0083829 A1

(43) Pub. Date: Mar. 26, 2009

Peterson

(54) COMPUTER SYSTEM

(75) Inventor: **David A. Peterson**, Cazenovia, NY (US)

Correspondence Address:
BOND, SCHOENECK & KING, PLLC
ONE LINCOLN CENTER
SYRACUSE, NY 13202-1355 (US)

(73) Assignee: **C & S OPERATIONS, INC.**, Syracuse, NY (US)

(21) Appl. No.: **12/234,131**

(22) Filed: **Sep. 19, 2008**

Related U.S. Application Data

(60) Provisional application No. 60/973,923, filed on Sep. 20, 2007.

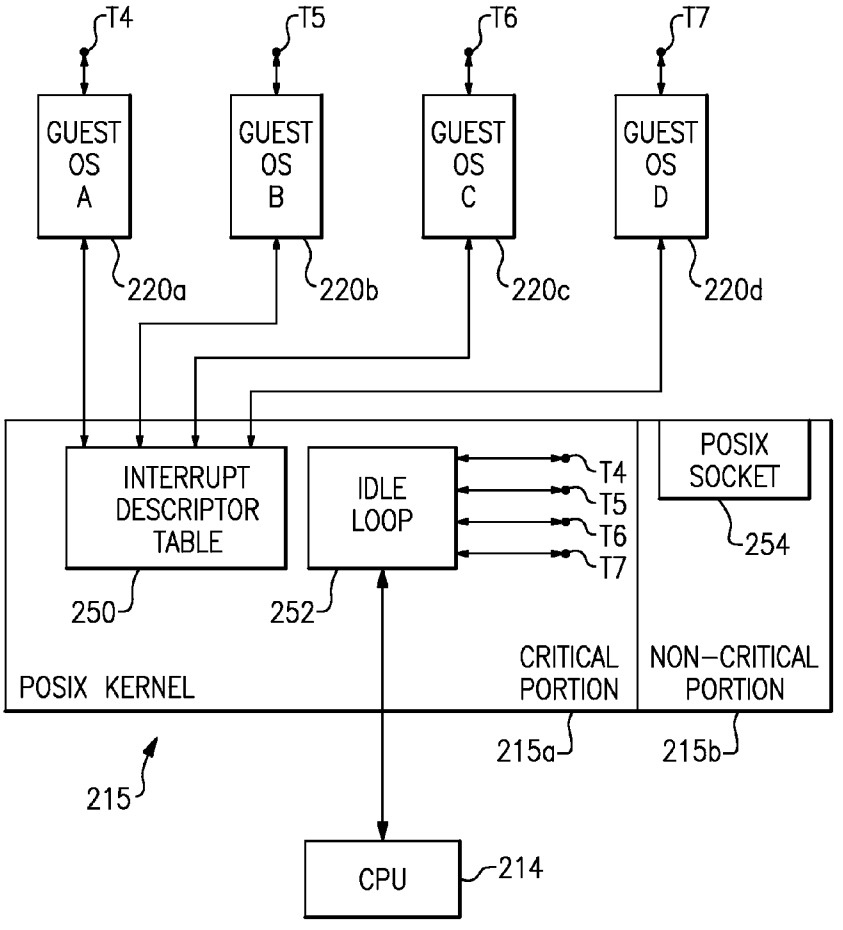
Publication Classification

(51) **Int. Cl.**
G06F 21/00 (2006.01)
G06F 9/54 (2006.01)

(52) **U.S. Cl.** **726/1**; 719/319

(57) **ABSTRACT**

The present invention is directed to computer systems, methods and/or hardware where one or more guest operating systems exchange instructions with the processing hardware (see DEFINITIONS section) through a controller kernel. Even though the instructions are exchanged through the controller kernel, rather than directly between the OS and the processing hardware, the controller kernel does not change the instructions out of native form. The controller kernel refrains from virtualizing or emulating the instructions. For this reason, the controller kernel cannot be considered to be and/or include middleware, a hypervisor or VMM. The use of the controller kernel can be helpful in computer systems with multiple guest OS's because it allows multiple containerized OS's to simultaneously run on a single set of processing hardware. For example, the multiple containerized OS's can be used to run multiple terminals. The use of the controller kernel may also be useful even if there is a single guest operating system. For example, a LINUX controller kernel has been found to speed up the operation of the Windows Vista operating system running as the guest OS, relative to the speed of Windows Vista running directly on the same processing hardware in the conventional way.



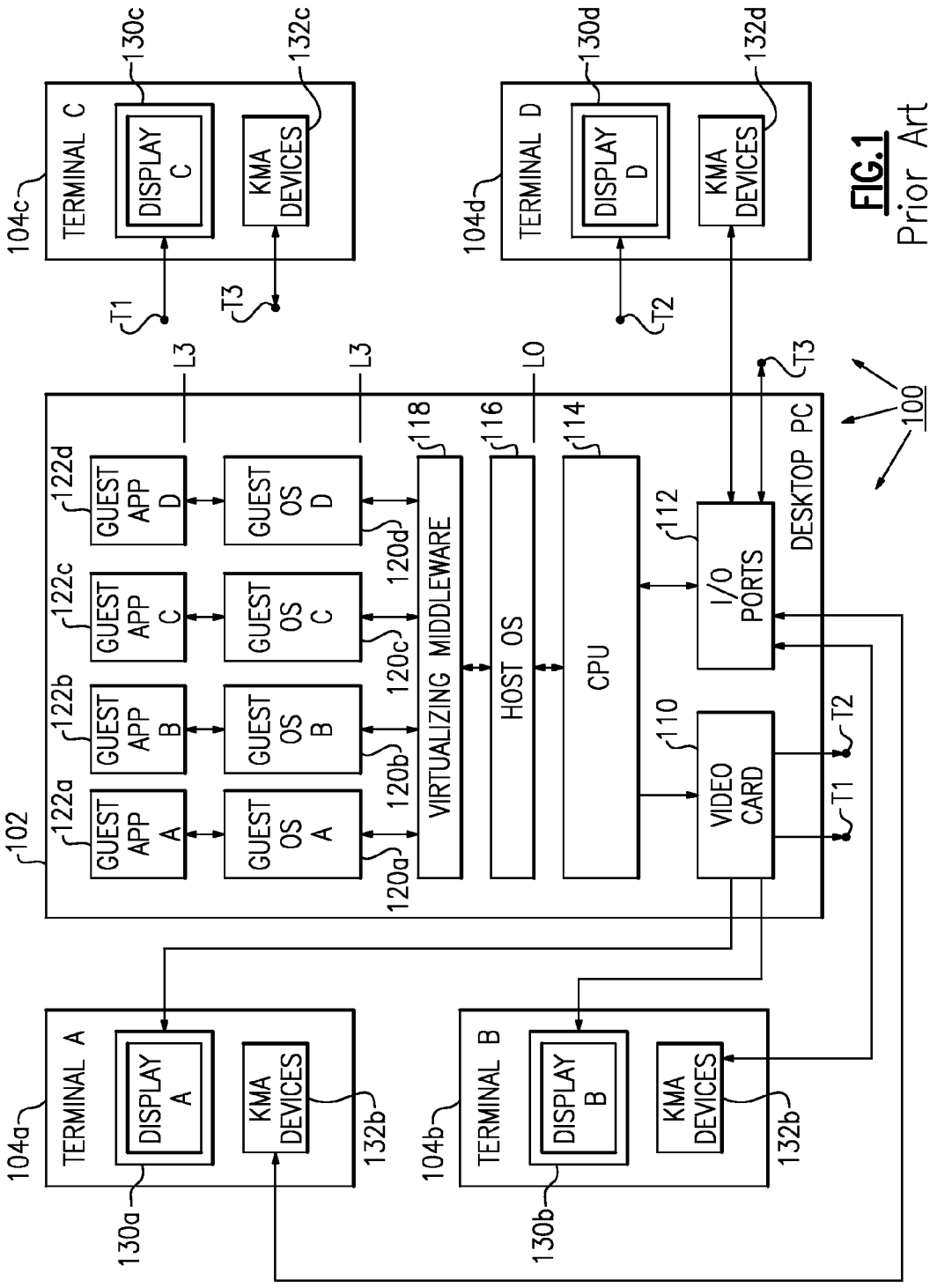


FIG.1
Prior Art

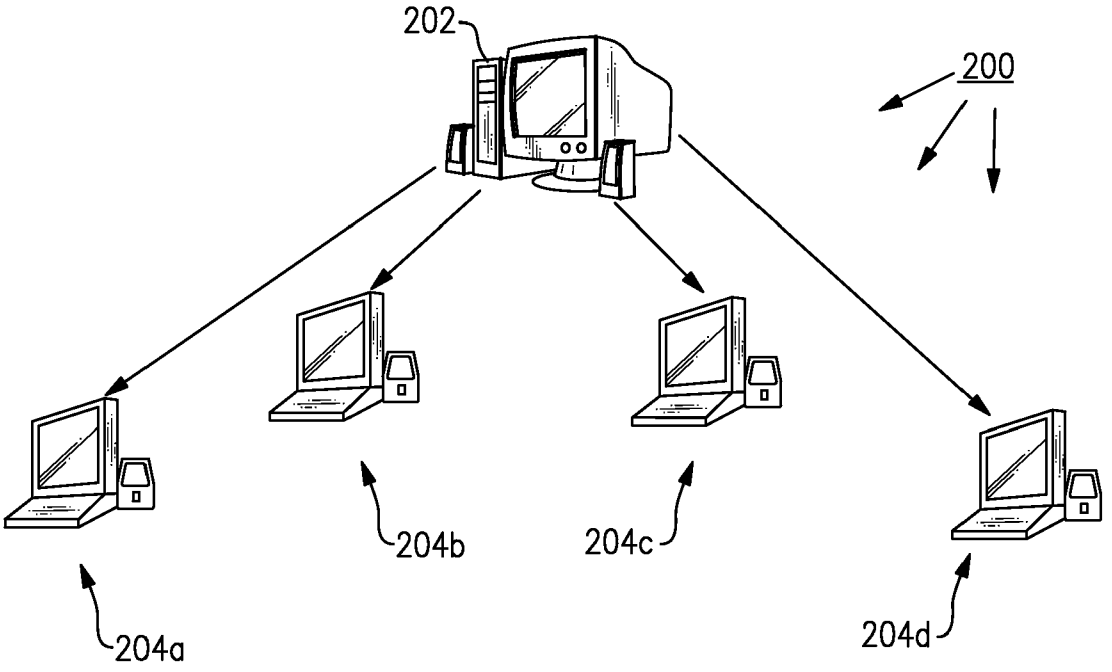


FIG.2

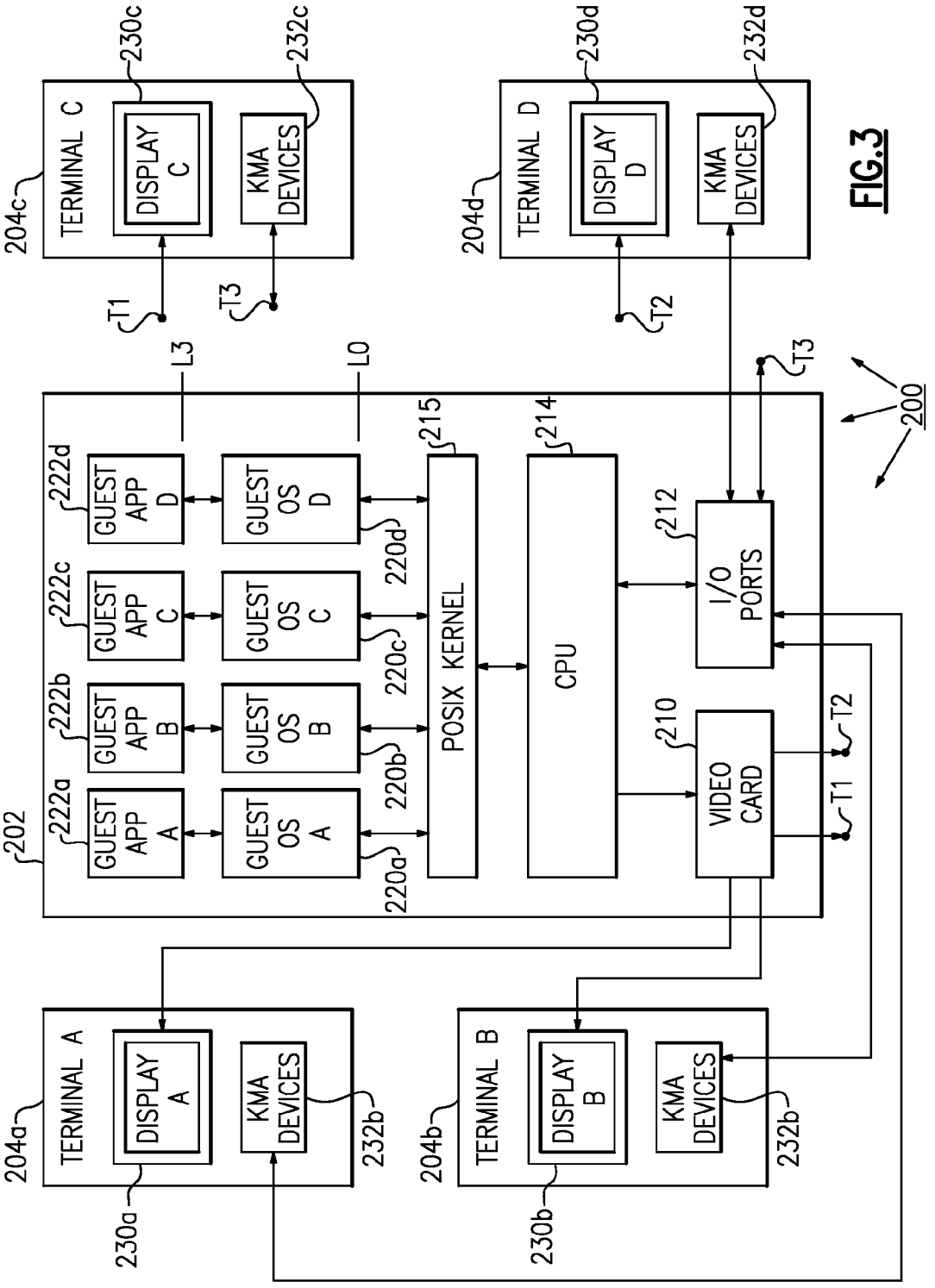

TERMINAL A
DISPLAY A
KMA DEVICES
TERMINAL B
DISPLAY B
KMA DEVICES
TERMINAL C
DISPLAY C
KMA DEVICES
TERMINAL D
DISPLAY D
KMA DEVICES
GUEST APP A
GUEST OS A
GUEST APP B
GUEST OS B
GUEST APP C
GUEST OS C
GUEST APP D
GUEST OS D
POSIX KERNEL
CPU
I/O PORTS
VIDEO CARD
T1
T2
T3
L3
L0
200
202
204a
204b
204c
204d
210
212
214
215
220a
220b
220c
220d
222a
222b
222c
222d
230a
230b
230c
230d
232b
232c
232d

FIG.3

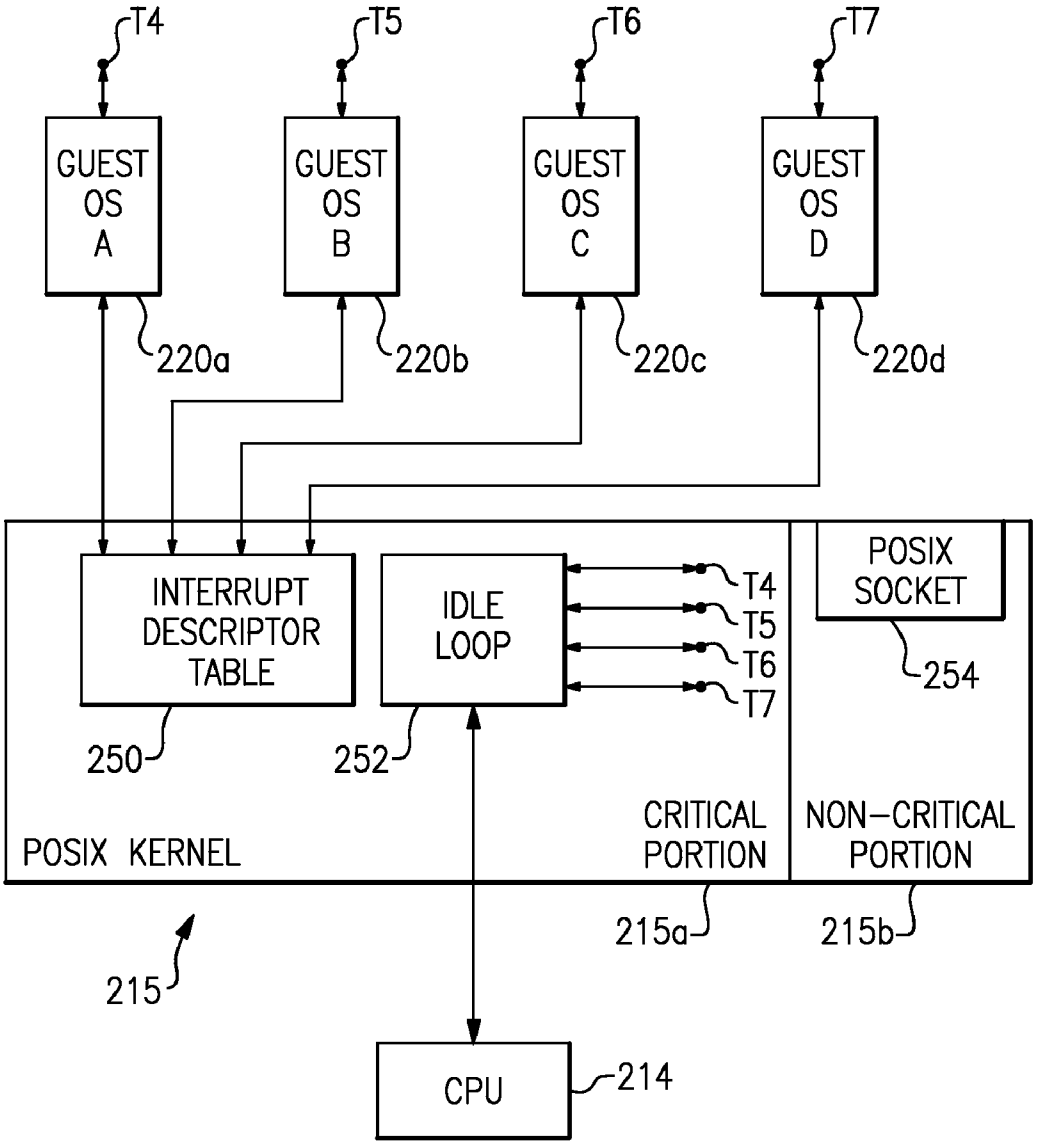

T4
T5
T6
T7
GUEST OS A
GUEST OS B
GUEST OS C
GUEST OS D
220a
220b
220c
220d
INTERRUPT DESCRIPTOR TABLE
IDLE LOOP
T4
T5
T6
T7
POSIX SOCKET
254
250
252
POSIX KERNEL
CRITICAL PORTION
NON-CRITICAL PORTION
215a
215b
215
CPU
214

FIG.4

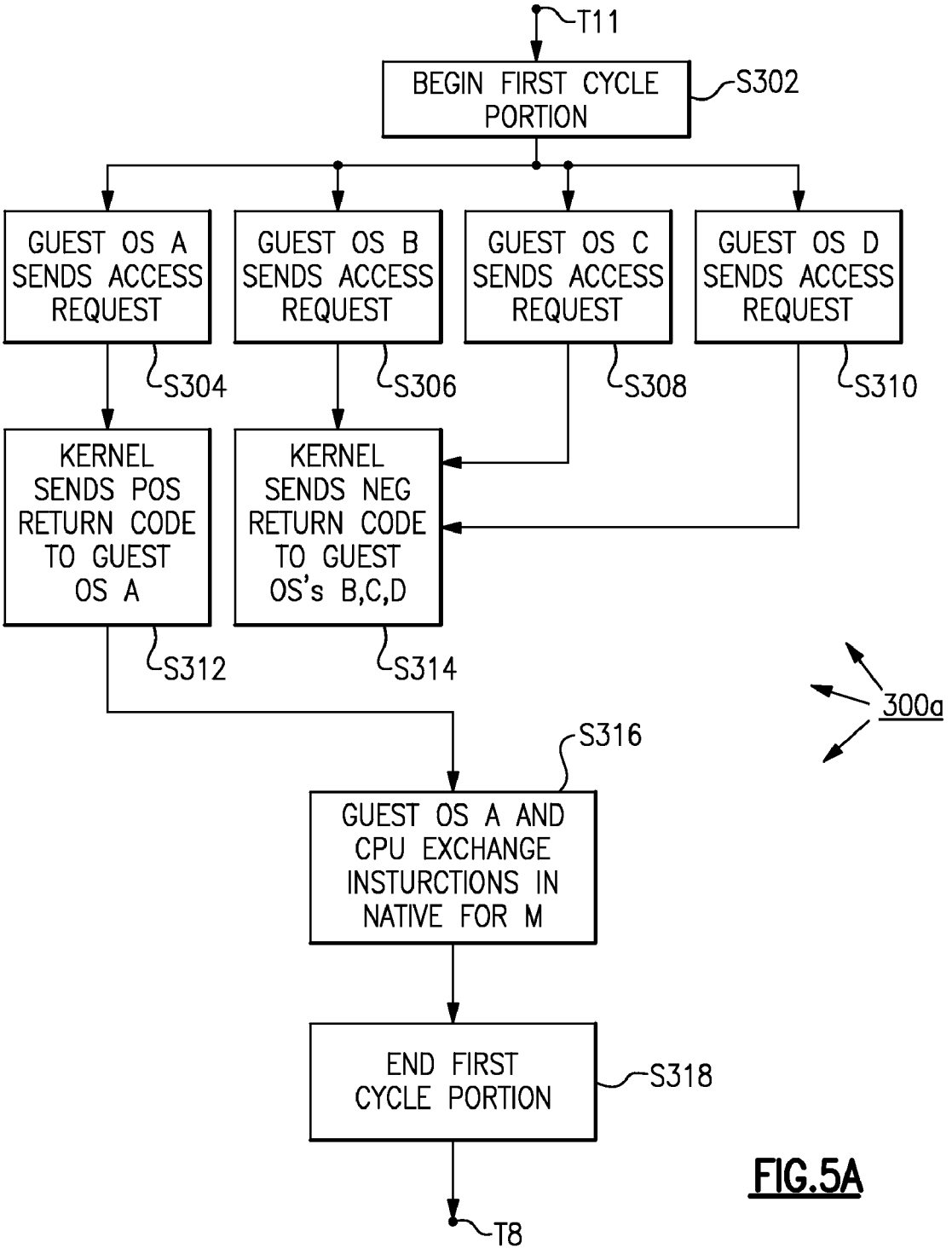

T11
BEGIN FIRST CYCLE PORTION
S302
GUEST OS A SENDS ACCESS REQUEST
GUEST OS B SENDS ACCESS REQUEST
GUEST OS C SENDS ACCESS REQUEST
GUEST OS D SENDS ACCESS REQUEST
S304
S306
S308
S310
KERNEL SENDS POS RETURN CODE TO GUEST OS A
KERNEL SENDS NEG RETURN CODE TO GUEST OS's B,C,D
S312
S314
300a
S316
GUEST OS A AND CPU EXCHANGE INSTURCTIONS IN NATIVE FOR M
END FIRST CYCLE PORTION
S318
T8

FIG.5A

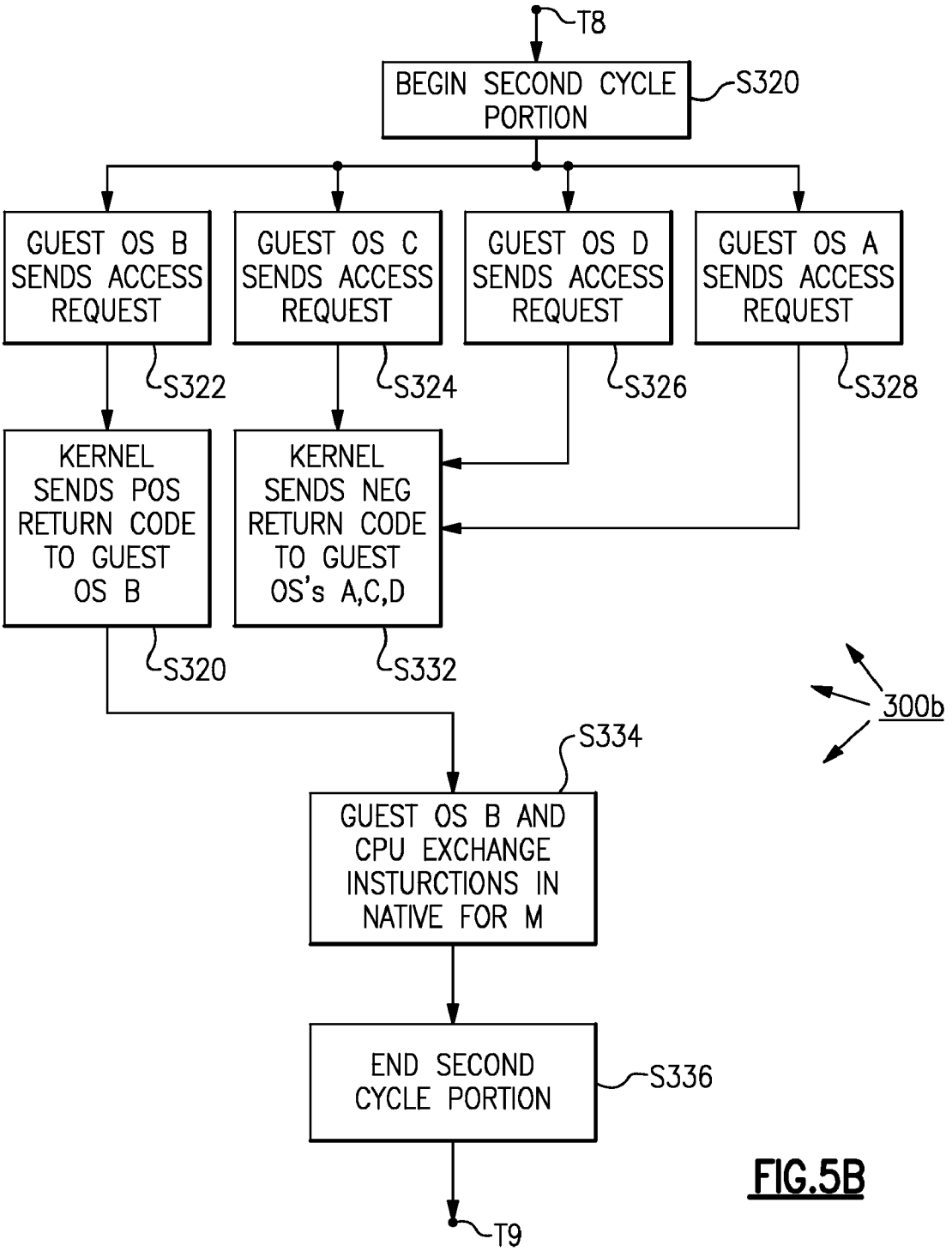

T8
BEGIN SECOND CYCLE PORTION
S320
GUEST OS B SENDS ACCESS REQUEST
GUEST OS C SENDS ACCESS REQUEST
GUEST OS D SENDS ACCESS REQUEST
GUEST OS A SENDS ACCESS REQUEST
S322
S324
S326
S328
KERNEL SENDS POS RETURN CODE TO GUEST OS B
KERNEL SENDS NEG RETURN CODE TO GUEST OS's A,C,D
S320
S332
300b
S334
GUEST OS B AND CPU EXCHANGE INSTURCTIONS IN NATIVE FOR M
END SECOND CYCLE PORTION
S336
T9

FIG.5B

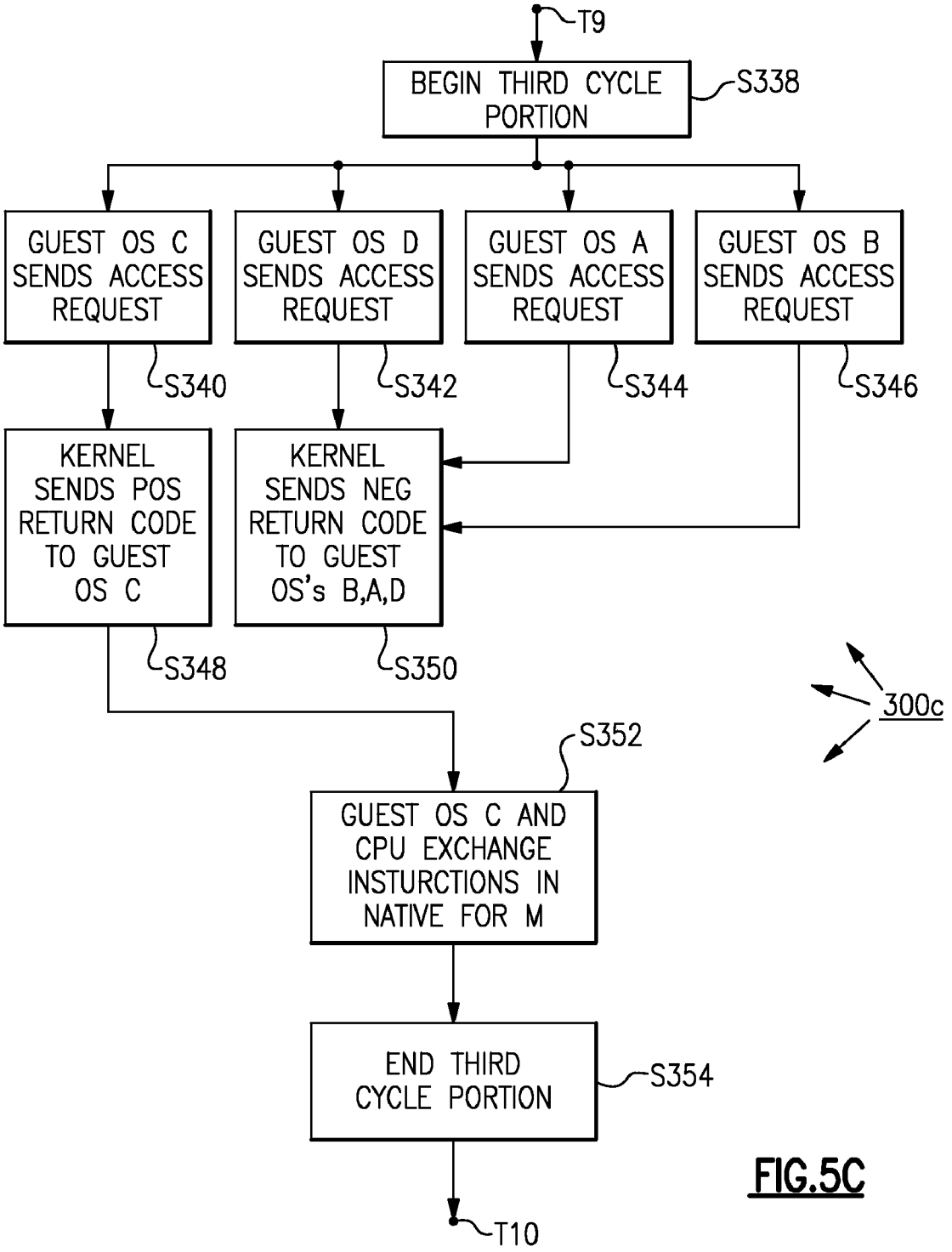

T9
BEGIN THIRD CYCLE PORTION
S338
GUEST OS C SENDS ACCESS REQUEST
GUEST OS D SENDS ACCESS REQUEST
GUEST OS A SENDS ACCESS REQUEST
GUEST OS B SENDS ACCESS REQUEST
S340
S342
S344
S346
KERNEL SENDS POS RETURN CODE TO GUEST OS C
KERNEL SENDS NEG RETURN CODE TO GUEST OS's B,A,D
S348
S350
300c
S352
GUEST OS C AND CPU EXCHANGE INSTURCTIONS IN NATIVE FOR M
END THIRD CYCLE PORTION
S354
T10

FIG.5C

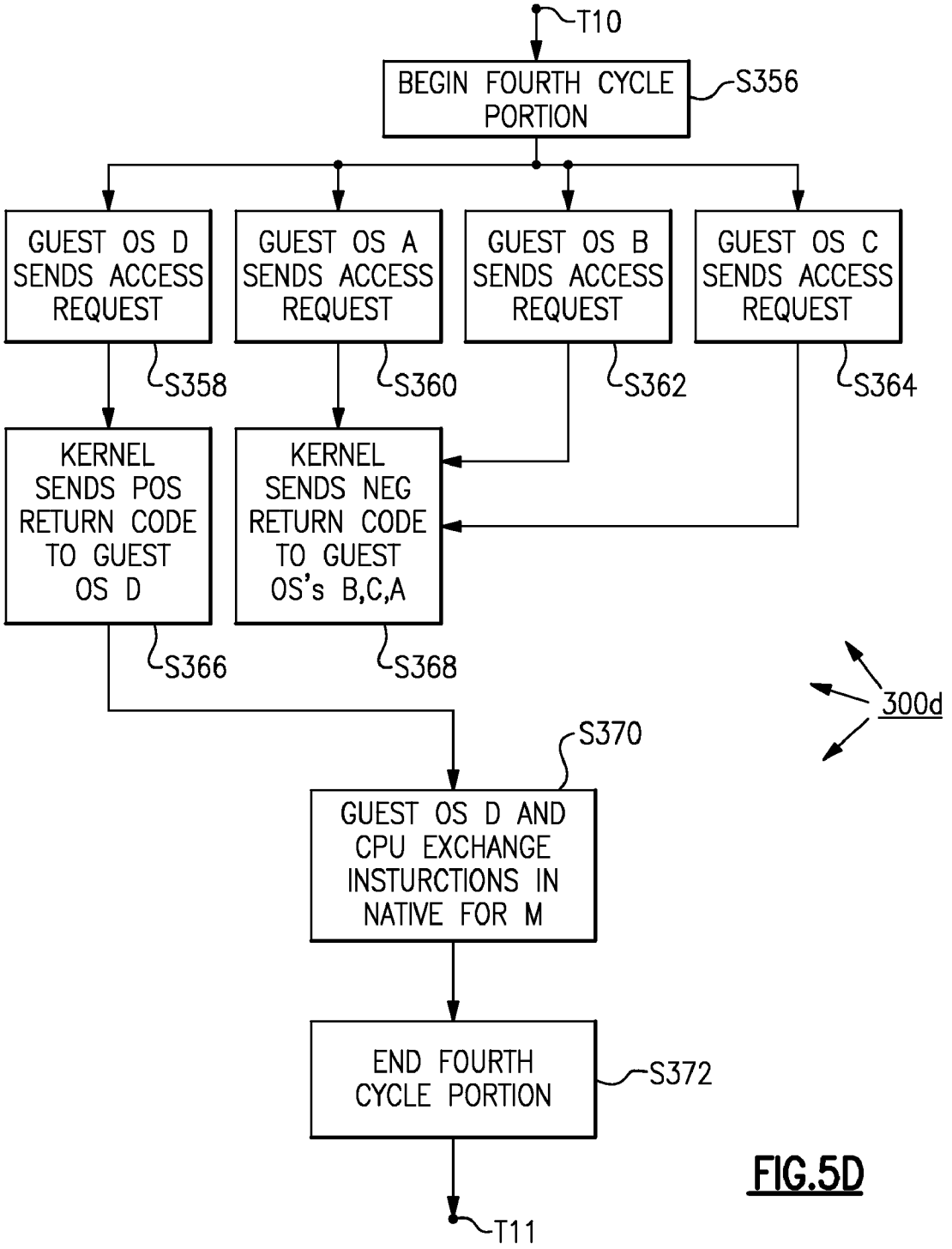

T10
BEGIN FOURTH CYCLE PORTION
S356
GUEST OS D SENDS ACCESS REQUEST
GUEST OS A SENDS ACCESS REQUEST
GUEST OS B SENDS ACCESS REQUEST
GUEST OS C SENDS ACCESS REQUEST
S358
S360
S362
S364
KERNEL SENDS POS RETURN CODE TO GUEST OS D
KERNEL SENDS NEG RETURN CODE TO GUEST OS's B,C,A
S366
S368
300d
S370
GUEST OS D AND CPU EXCHANGE INSTURCTIONS IN NATIVE FOR M
END FOURTH CYCLE PORTION
S372
T11

FIG.5D

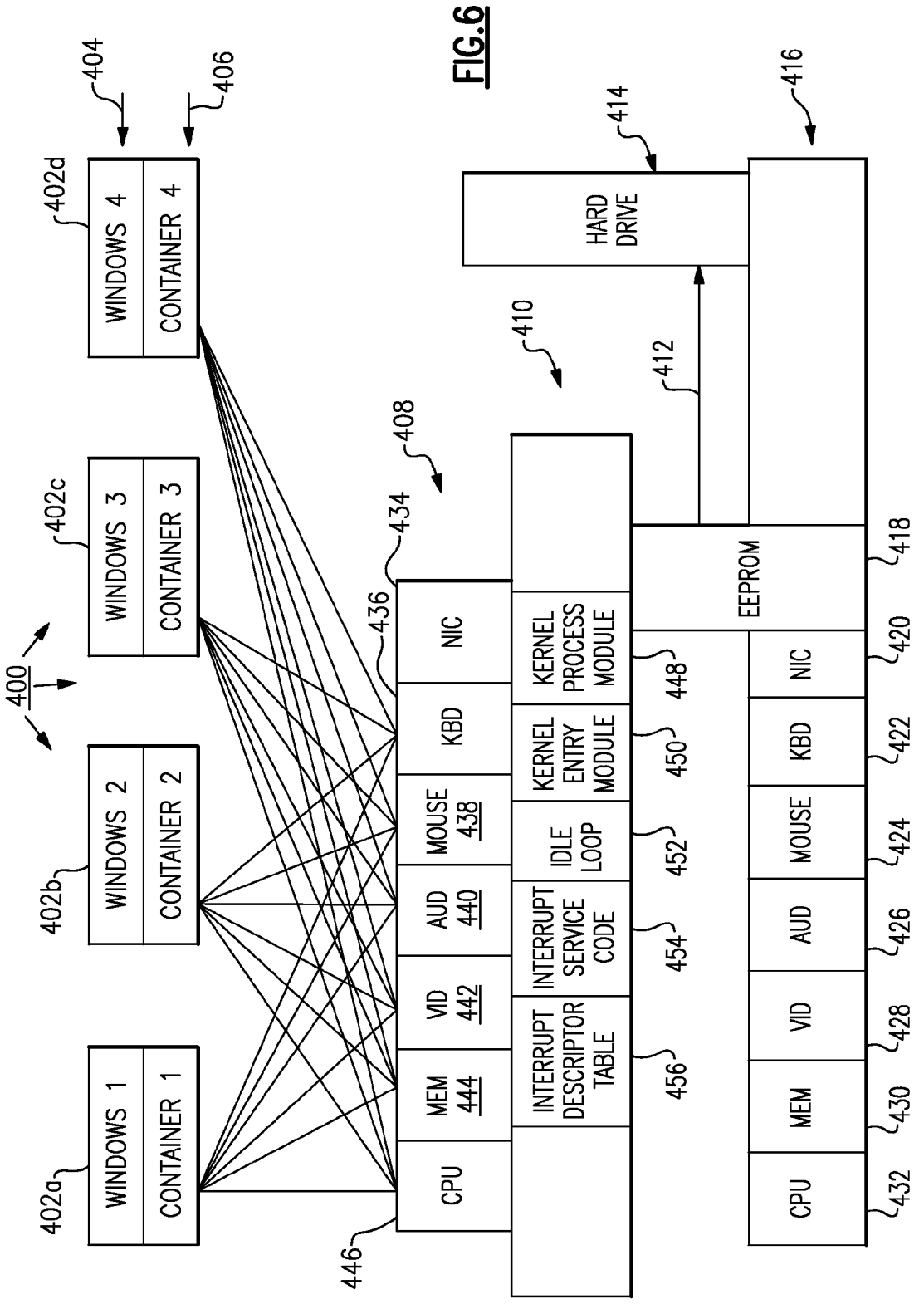

400
402a
WINDOWS 1
CONTAINER 1
402b
WINDOWS 2
CONTAINER 2
402c
WINDOWS 3
CONTAINER 3
402d
WINDOWS 4
CONTAINER 4
404
406
408
410
412
414
416
418
420
422
424
426
428
430
432
434
436
438
440
442
444
446
448
450
452
454
456
CPU
MEM 444
VID 442
AUD 440
MOUSE 438
KBD
NIC
INTERRUPT DESCRIPTOR TABLE
INTERRUPT SERVICE CODE
IDLE LOOP
KERNEL ENTRY MODULE
KERNEL PROCESS MODULE
HARD DRIVE
EEPROM
CPU
MEM
VID
AUD
MOUSE
KBD
NIC

FIG.6

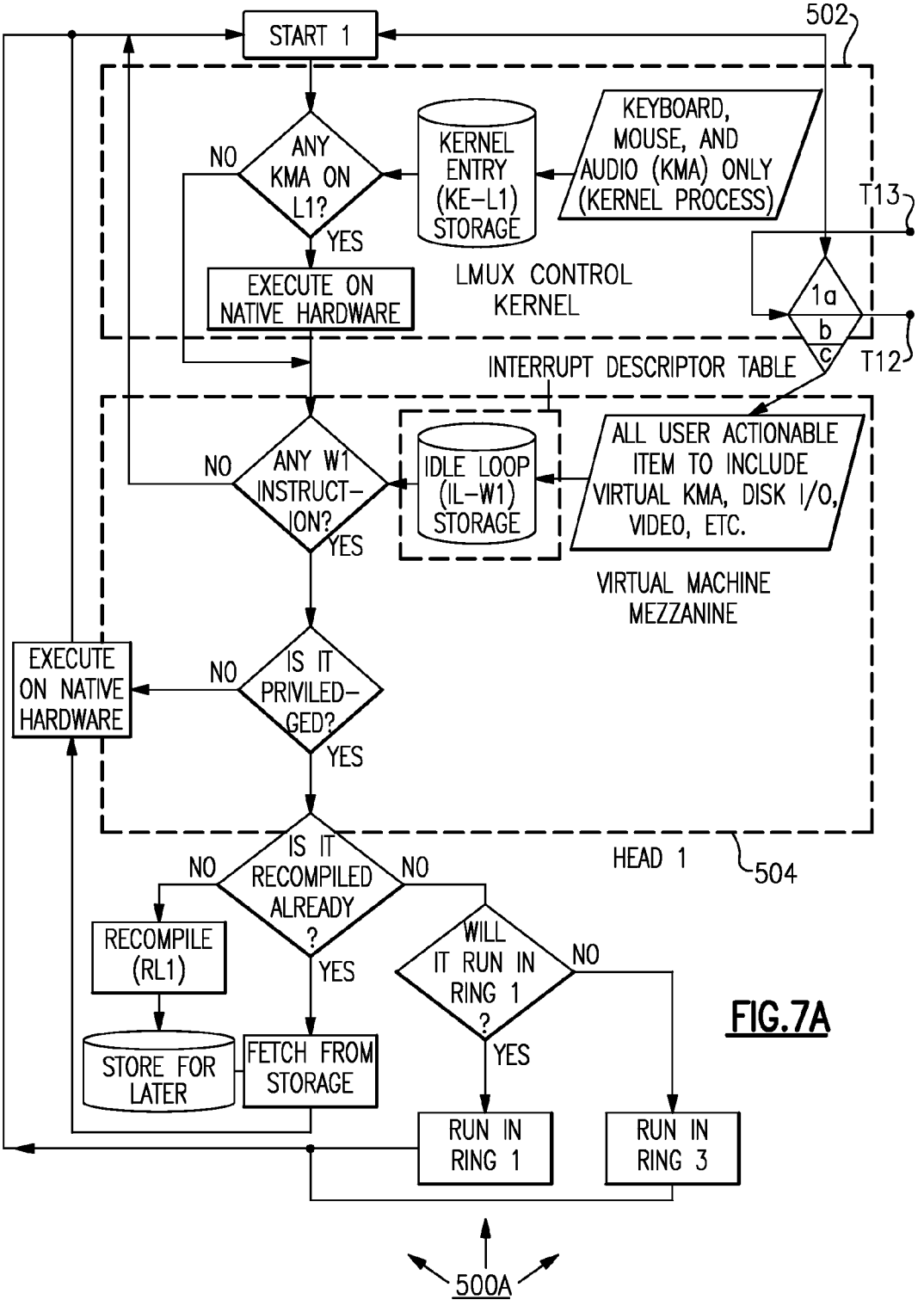

502
START 1
ANY KMA ON L1?
NO
YES
KERNEL ENTRY (KE-L1) STORAGE
KEYBOARD, MOUSE, AND AUDIO (KMA) ONLY (KERNEL PROCESS)
T13
EXECUTE ON NATIVE HARDWARE
LMUX CONTROL KERNEL
1a
b
c
T12
INTERRUPT DESCRIPTOR TABLE
ANY W1 INSTRUCT-ION?
NO
YES
IDLE LOOP (IL-W1) STORAGE
ALL USER ACTIONABLE ITEM TO INCLUDE VIRTUAL KMA, DISK I/O, VIDEO, ETC.
VIRTUAL MACHINE MEZZANINE
EXECUTE ON NATIVE HARDWARE
IS IT PRIVILED-GED?
NO
YES
IS IT RECOMPILED ALREADY ?
NO
NO
YES
HEAD 1
504
RECOMPILE (RL1)
WILL IT RUN IN RING 1 ?
NO
YES
STORE FOR LATER
FETCH FROM STORAGE
RUN IN RING 1
RUN IN RING 3
500A

FIG.7A

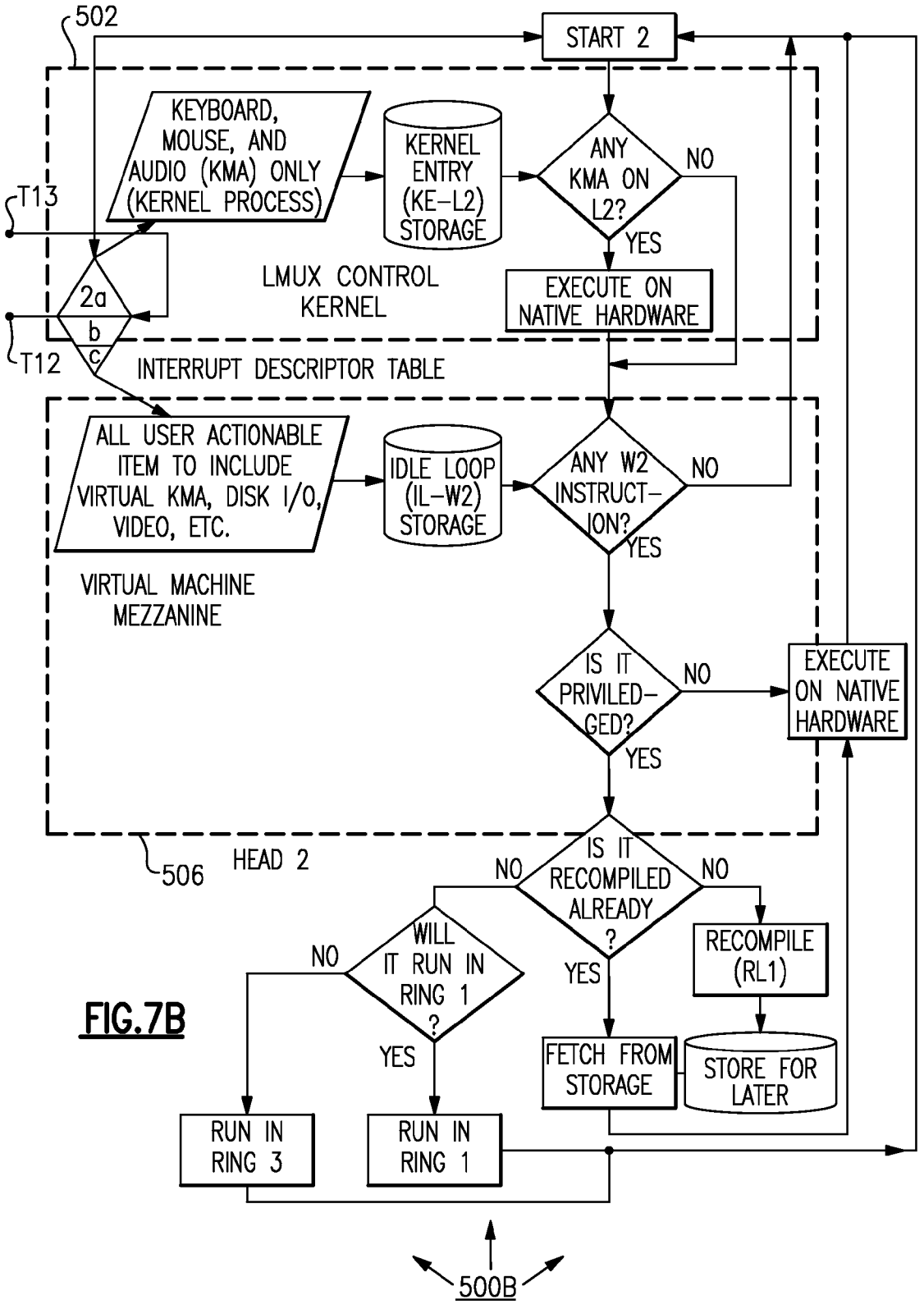

502
START 2
KEYBOARD, MOUSE, AND AUDIO (KMA) ONLY (KERNEL PROCESS)
KERNEL ENTRY (KE-L2) STORAGE
ANY KMA ON L2?
NO
YES
T13
2a
b
c
T12
LMUX CONTROL KERNEL
EXECUTE ON NATIVE HARDWARE
INTERRUPT DESCRIPTOR TABLE
ALL USER ACTIONABLE ITEM TO INCLUDE VIRTUAL KMA, DISK I/O, VIDEO, ETC.
IDLE LOOP (IL-W2) STORAGE
ANY W2 INSTRUCT-ION?
NO
YES
VIRTUAL MACHINE MEZZANINE
IS IT PRIVILED-GED?
NO
YES
EXECUTE ON NATIVE HARDWARE
506
HEAD 2
IS IT RECOMPILED ALREADY ?
NO
NO
YES
WILL IT RUN IN RING 1 ?
NO
YES
RECOMPILE (RL1)
FETCH FROM STORAGE
STORE FOR LATER
RUN IN RING 3
RUN IN RING 1
500B

FIG.7B

COMPUTER SYSTEM

RELATED APPLICATION

[0001] The present application claims priority to U.S. provisional patent application No. 60/973,923, filed on Sep. 20, 2007; all of the foregoing patent-related document(s) are hereby incorporated by reference herein in their respective entirety(ies).

BACKGROUND OF THE INVENTION

[0002] 1. Field of the Invention

[0003] The present invention relates to computer systems with a computer running multiple operating systems and more particularly to computer systems with a computer running multiple containerized (see DEFINITIONS section) operating systems to be respectively used by multiple terminals (see DEFINITIONS section).

[0004] 2. Description of the Related Art

[0005] It is conventional to have a computer, such as a modified PC desktop type host computer, which controls and operates a plurality of terminals. In fact, mainframe computers dating back to at least the 1970s operated in this way. More recently, each terminal has been given its own operating system and/or instance of an operating system. These kind of systems are herein called multi-terminal systems.

[0006] It is conventional to use a hypervisor to run multiple operating systems on a single computer. A hypervisor (or virtual machine monitor) is a virtualization platform that allows multiple operating systems to run on a host computer at the same time. Some hypervisors take the form of software that runs directly on a given hardware platform as an operating system control program. With this kind of hypervisor, the guest operating system runs at the second level above the hardware. Other hypervisors take the form of software that runs within an operating system environment.

[0007] Hypervisors have conventionally been used in multi-terminal systems where each terminal has a dedicated guest operating system on a single host computer. In these conventional multi-terminal systems, I/O devices communicate I/O data through the hypervisor to perform basic I/O operations (see DEFINITIONS section). More specifically: (i) data from the I/O devices is communicated through the hypervisor to the computing hardware of the host computer; and (ii) from the computing hardware (if any) is communicated through the hypervisor to the I/O devices. Because the hypervisor is a virtualization platform, this means that the I/O devices must be virtualized in the software of the hypervisor and/or the guest operating system so that the communication of I/O data through the hypervisor can take place.

[0008] FIG. **1** shows prior art computer system **100** including: desktop PC **102** and four terminals **104***a*, **104***b*, **104***c* and **104***d*. Desktop PC **102** includes: video card **110**; I/O ports **112**; CPU **114**; host operating system ("OS") **116**; virtualizing middleware **118**, four guest OS's (see DEFINITIONS section) **120***a*, **120***b*, **120***c*, **120***d*; and four guest applications **122***a*, **122***b*, **122***c* and **122***d*. Each terminal **104** includes: display **130** and keyboard-mouse-audio ("KMA") devices **132**. Host OS may be any type of OS, such as Windows, Apple or POSIX (see DEFINITIONS section). As shown in FIG. **1**, host OS **116** runs at security level (see DEFINITIONS section) L**0**, which may be, for example in an x86 CPU architecture, Ring Zero. This means that host OS **116** exchanges instructions directly with CPU **116** in native form (see DEFINITIONS section).

[0009] The guest OS's **120***a*, **120***b*, **120***c*, **120***d* are used to respectively control the four terminals **104***a*, **104***b*, **104***c*, **104***d*. This means that the four guest OS's: (i) control the visual displays respectively shown on displays **130***a*, **130***b*, **130***c*, **130***d*; (ii) receive input from the four keyboards **132***a*, **132***b*, **132***c*, **132***d*; (iii) receive input from the four mice **132***a*, **132***b*, **132***c*, **132***d*; and (iv) control audio for the four audio output devices (for example, speakers, headphones) **132***a*, **132***b*, **132***c*, **132***d*. The four guest OS's **120***a*, **120***b*, **120***c*, **120***d* are containerized virtual machines so that work by one user on one terminal does not affect or interfere with work by another user on another terminal. As shown in FIG. **1**, they can respectively run their own application(s) **122***a*, **122***b*, **122***c*, **122***d* in an independent manner.

[0010] However, the four guest OS's are virtual machines, running at a security level **13**, which is above the OS security level (see DEFINITIONS section) L**0**. For example, in an x86 architecture, the guest OS's **120***a*, **120***b*, **120***c*, **120***d* would be running at Ring Three. This is an indirect form of communication with the CPU **114**. Furthermore, the instructions exchanged between the guest OS's and the CPU are virtualized by virtualizing middleware **118**, which may take the form of a hypervisor or virtual machine manager ("VMM"). For example, some of the exchanged instructions relate to basic I/O operations. When the exchanged instructions are virtualized by virtualizing middleware **118**, the instructions are taken out of their native form and put in a virtualized form. This virtualized form is generally a lot more code intensive than native form. This virtualization makes operations slower and more prone to error than similar exchanges between a host OS, running at the OS security level and the CPU.

[0011] US published patent application 2008/0092145 ("Sun") discloses a system including secure operating system switching. Sun discloses that to perform secure OS switching, a logically independent piece of software referred to as the OS switcher is used. When the Sun CPU is executing the Sun OS switcher code, the CPU is in the switcher mode. Otherwise, the CPU is operating in legacy mode. Sun discloses that its OS switching emulates multiple computer systems in one, where at any time only one of them is active and others are suspended. Sun further discloses that special care is taken during OS switching, as OS kernels typically are not ready to deal with the sudden loss of hardware ownership or loss of CPU execution control. Sun further discloses that there are many possible ways to achieve strong security and isolation among multiple OS's in OS switching with VT-x. Sun further discloses that the legacy mode is mapped into the non-root operation mode in VT-x while the switcher code is implemented in the root operation mode (ring 0 specifically). In order to ensure the continuing running of the OS's in legacy mode, certain emulations are implemented in the OS switcher.

[0012] US published patent application 2006/0267857 ("Zhang") discloses a multi-terminal system wherein multiple terminals are connected through a single graphical user interface layer. The host computer includes an event queue module for receiving each input command from the input device(s) of each terminal.

[0013] US patent application 2007/0174414 ("Song") discloses a thin client/server computer system. Communication between the server (or host) and multiple thin clients is per-

formed by independent computing architecture (ICA) from Citrix Systems, Inc., or a remote desktop protocol from Microsoft Corporation. Each thin client includes a CPU, separate individual operating system, a high capacity memory, RAM, BIOS firmware and peripheral device connection hardware. The Song system includes a CPU at each terminal. For example, an execution result is sent from the host to the thin client as a bitmap image which the thin client processes locally, with its processing unit, so that can be displayed on a monitor at the thin client. Song does disclose that its host may include an EEPROM, but does not mention anything about partitions. It is believed that the X300 Access Terminal Kit sold by Ncomputing is an embodiment of the technology described in Song.

[0014] The Applica PC Sharing Zero Client Network Computing Remote Workstation powered by Applica Inc. (see www.applica.com website, cached versions 31 Jul. 2007 and earlier) discloses a multi-terminal system.

[0015] US patent application 2003/0018892 ("Tello") discloses a secure boot process for a personal computer. In the Tello process, a security kernel typically resides in the upper area in memory for encrypting/decrypting data from any application that is running under the operating system. In this way, two operating systems can work separately using the same hardware. In place of a standard BIOS, the Tello process utilizes a security engine including a kernel stored in a flash memory, a modified north bridge and a smart card for auto burning the flash memory portion of the security engine and key generation.

[0016] US patent application 2007/0097130 ("Margulis") discloses a multi-user host computer system. The Margulis system includes a host computer that processes applications and the desktop environments for multiple remote terminals. The host computer also includes a terminal services offload processor to supplement the processing of the host CPU. The terminal services offload processor is alleged to improve the video and graphics performance and to allow the multi-user host computer system to more efficiently support multiple users. The host computer includes a graphics processor that manages a virtual display for each remote terminal and provides selective updates of sub frame data. The sub frame data is encoded and transmitted (as appropriate) over the network to the remote terminals. Video data streams are optimized by the terminal services offload processor and optimized for the intended remote terminals and their network connections.

[0017] US patent application 2008/0168479 ("Purtell") discloses a computer system that system augments machine virtualization by entirely bypassing resource emulation for performance-critical features, such as 3D graphics acceleration, through the use of high-performance interfaces between the guest OS and the host OS. The Purtell system is alleged to ameliorate the performance penalties and functionality restrictions of conventional resource emulation. Purtell states: "Bypass virtualization avoids the performance penalties and functionality restrictions of conventional resource emulation by a VMM by bypassing the VMM—requests issued by a guest OS for a host OS resource are instead channeled through the Bypass Interface. The Bypass Interface intercepts the requests, forwards them to the host OS, which passes the request on to the actual resource, and then the returns the response from the host OS resource to the guest OS. Since is unnecessary to implement the Bypass Interface for every OS resource, problematic or performance-insensitive resources can be handled with machine virtualization."

[0018] Other publications potentially of interest include: (i) U.S. Pat. No. 5,903,752 ("Dingwall"); (ii) US patent application 2007/0028082 ("Lien"); (iii) US patent application 2008/0077917 ("Chen"); (iv) US published patent application 2007/0078891 ("Lescouet"); (v) US published patent application 2007/0204265 ("Oshins"); (vi) US published patent application 2007/0057953 ("Green"); (vii) US patent application 2007/0174410 ("Croft"); (viii) US patent application 2004/0073912 ("Meza"); and/or (ix) US patent application 2007/0043928 ("Panesar").

[0019] Description Of the Related Art Section Disclaimer: To the extent that specific publications are discussed above in this Description of the Related Art Section, these discussions should not be taken as an admission that the discussed publications (for example, published patents) are prior art for patent law purposes. For example, some or all of the discussed publications may not be sufficiently early in time, may not reflect subject matter developed early enough in time and/or may not be sufficiently enabling so as to amount to prior art for patent law purposes. To the extent that specific publications are discussed above in this Description of the Related Art Section, they are all hereby incorporated by reference into this document in their respective entirety(ies).

BRIEF SUMMARY OF THE INVENTION

[0020] The present invention is directed to computer systems, methods and/or hardware where one or more guest operating systems exchange instructions with the processing hardware (see DEFINITIONS section) through a controller kernel. Even though the instructions are exchanged through the controller kernel, rather than directly between the OS and the processing hardware, the controller kernel does not change the instructions out of native form. The controller kernel refrains from virtualizing or emulating the instructions. For this reason, the controller kernel cannot be considered to be and/or include middleware, a hypervisor or VMM. The use of the controller kernel can be helpful in computer systems with multiple guest OS's because it allows multiple containerized OS's to simultaneously run on a single set of processing hardware. For example, the multiple containerized OS's can be used to run multiple terminals. The use of the controller kernel may also be useful even if there is a single guest operating system. For example, a LINUX controller kernel has been found to speed up the operation of the Windows Vista operating system running as the guest OS, relative to the speed of Windows Vista running directly on the same processing hardware in the conventional way.

[0021] Another aspect of the present invention is Multi-Sharing Software Cursor (modified event device). Modified Linux kernel that creates a SW cursor for each input device. Hides the HW cursor and allows multiple monitors to be concurrently used (modified EVDEV—event device). Note that EVDEV is based on open source and not modularized, but a unique aspect is the installation script (copyrightable) that allows the EVDEV to be used in a manner for which it was not designed: controlling/handling multiple software cursors.

[0022] Another aspect of the present invention is Multi-Sharing. Separate desktops for the software cursor (modified zephyr) Modified Linux kernel from associating the same device (KMA) with a different control file.

[0023] Another aspect of the present invention is containerization. Containerized guest OS on each workstation The ability for the operating system to host individual guest oper-

ating systems. The controller kernel is used as a "traffic cop" to allow the loading of guest containerized OS's. It is a modified Linux kernel using propriety code in a module, using elements of Linux to achieve a function for which the individual elements were not designed.

[0024] Another aspect of the present invention is connection to Ring Zero. The controller kernel runs guest operating systems directly on Ring Zero, so that the "traffic cop" allows host operating systems to link other applications to Ring Zero for a small amount of time. Normally, Ring Zero is unmanaged and restricted to authorized code, and interacts most directly with hardware, thus running faster. In the case of running multiple operating systems, there is a need to manage the invention the multiple operating systems' use of Ring Zero. The benefit of the invention is that it allows the host OS's to work much faster when compared with other virtual machines, as fast as a normal desktop setup. Modified Linux kernel treats a host OS as an application—the controller kernel allows a Linux application that would normally run in a slower Ring Three in a Ring Zero.

[0025] Another aspect of the present invention is locating the controller kernel in BIOS enables the software cursor and separate desktops for the software cursor. The benefits of this aspect of the present invention are that the start time is decreased.

[0026] Various embodiments of the present invention may exhibit one or more of the following objects, features and/or advantages:

[0027] (1) decreased boot time;

[0028] (2) eliminate limitations on applications encountered with server based architectures;

[0029] (3) reduce PC administration (for example, virus updates, service pack updates);

[0030] (4) extend capabilities of single PC to run a plurality of terminals;

[0031] (5) reduce cost of acquisition and cost of ownership;

[0032] (6) allows legal sharing of certain software licenses;

[0033] (7) multiple terminals with familiar desktop display;

[0034] (8) multiple terminals with no custom configuration or special protocols; and

[0035] (9) system useful for libraries, classrooms, businesses, governmental applications retail terminals and/or retail kiosks.

[0036] According to one aspect of the present invention, a computer system includes processing hardware, a first guest operating system, and a controller kernel. The processing hardware defines an OS security level and at least a first additional security level above the OS security level. The controller kernel runs on the processing hardware. The controller kernel is programmed to allow the first guest operating system exchange instructions with the processing hardware through the controller kernel at the OS security level.

[0037] According to a further aspect of the present invention, a computer includes processing hardware, a first memory portion, and a controller memory portion. The processing hardware defines an OS security level and at least a first additional security level above the OS security level. The first memory portion is programmed with a first guest operating system. The controller memory portion is programmed with a controller kernel running on the processing hardware. The controller kernel is programmed to allow the first guest operating system exchange instructions with the processing hardware through the controller kernel at the OS security level.

[0038] According to a further aspect of the present invention, a method includes the following steps (not necessarily in the following order): (i) providing a computer system; (ii) running the controller kernel on the processing hardware; and (iii) exchanging instructions through the controller kernel between the first guest operating system and the processing hardware at the OS security level. At the providing step, the computer system includes processing hardware, a first guest operating system, and a controller kernel. The processing hardware defines an OS security level and at least a first additional security level above the OS security level.

[0039] According to a further aspect of the present invention, a computer system includes processing hardware, a first guest operating system, a second guest operating system and a controller kernel. The processing hardware defines an OS security level and at least a first additional security level above the OS security level. The first guest operating system and the second guest operating system are containerized with respect to each other. The controller kernel runs on the processing hardware. The controller kernel is programmed to perform cycles including: (i) a first cycle portion when the first guest operating system exchanges instructions with the processing hardware at the OS security level through the controller kernel, and (ii) a second cycle portion when the second guest operating system exchanges instructions with the processing hardware at the OS security level through the controller kernel.

[0040] According to a further aspect of the present invention, a computer includes processing hardware, a first memory portion, a second memory portion and a controller memory portion. The processing hardware defines an OS security level and at least a first additional security level above the OS security level. The first memory portion is programmed with a first guest operating system. The second memory portion is programmed with a second guest operating system. The first guest operating system and the second guest operating system are containerized with respect to each other. The controller memory portion is programmed with a controller kernel running on the processing hardware. The controller kernel being programmed to perform cycles including: (i) a first cycle portion when the first guest operating system exchanges instructions with the processing hardware at the OS security level through the controller kernel, and (ii) a second cycle portion when the second guest operating system exchanges instructions with the processing hardware at the OS security level through the controller kernel.

[0041] According to a further aspect of the present invention, a method includes the step of providing a computer system including processing hardware, a first guest operating system, a second guest operating system and a controller kernel. The processing hardware defines an OS security level and at least a first additional security level above the OS security level. The first guest operating system and the second guest operating system are containerized with respect to each other. The method further includes the step of running cycles by the controller kernel. Each cycle include the following sub-steps: (i) during a first cycle portion, exchanging instructions between the first guest operating system and the processing hardware at the OS security level through the controller kernel, and (ii) during a second cycle portion, exchanging instructions between the second guest operating system and the processing hardware at the OS security level through the controller kernel.

[0042] According to a further aspect of the present invention, a computer system includes processing hardware, a first guest operating system, a second guest operating system, a controller kernel, a first terminal hardware set and a second terminal hardware set. The first guest operating system and the second guest operating system are containerized with respect to each other. The controller kernel is programmed to control the exchange of instructions between the first guest operating system and the processing hardware and the exchange of instructions between the second operating systems and the processing hardware. The first terminal hardware set is controlled by the first guest operating system. The first terminal hardware set is in the form of an ultra thin terminal. The second terminal hardware set is controlled by the second guest operating system. The second terminal hardware set in the form of an ultra thin terminal.

[0043] According to a further aspect of the present invention, a computer includes processing hardware, a first memory portion, a second memory portion, a controller memory portion, a first terminal hardware set and a second terminal hardware set. The first memory portion is programmed with a first guest operating system. The second memory portion is programmed with a second guest operating system. The first guest operating system and the second guest operating system are containerized with respect to each other. The controller memory portion is programmed with a controller kernel programmed to control the exchange of instructions between the first guest operating system and the processing hardware and the exchange of instructions between the second operating systems and the processing hardware. The first terminal hardware set is controlled by the first guest operating system. The first terminal hardware set in the form of an ultra thin terminal. The second terminal hardware set is controlled by the second guest operating system. The second terminal hardware set is in the form of an ultra thin terminal.

[0044] According to a further aspect of the present invention, a method includes the step of: (a) providing a computer system including processing hardware, a first guest operating system, a second guest operating system (with the first guest operating system and the second guest operating system being containerized with respect to each other), a controller kernel, a first terminal hardware set (in the form of an ultra thin terminal), and a second terminal hardware set (in the form of an ultra thin terminal). The method further includes the following steps: (b) controlling, by the controller kernel, an exchange of instructions between the first guest operating system and the processing hardware; (c) controlling, by the first guest operating system, the first terminal hardware set based on the exchange of instructions occurring at step (b); (d) controlling, by the controller kernel, an exchange of instructions between the second guest operating system and the processing hardware; and (e) controlling, by the second guest operating system, the second terminal hardware set based on the exchange of instructions occurring at step (d).

[0045] According to a further aspect of the present invention, a computer system includes processing hardware, a first guest operating system, a second guest operating system, and a controller kernel. The processing hardware defines an OS security level and at least a first additional security level above the OS security level. the controller kernel runs on the processing hardware. The controller kernel is programmed to: (i) selectively allow the first guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the second guest operating system in a manner that allows the second guest operating system to continue running, and (ii) selectively allow the second guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the first guest operating system in a manner that allows the first guest operating system to continue running.

[0046] According to a further aspect of the present invention, a computer includes processing hardware, a first memory portion, a second memory portion, and a controller memory portion. The processing hardware defines an OS security level and at least a first additional security level above the OS security level. The first memory portion is programmed with a first guest operating system. The second memory portion is programmed with a second guest operating system. The controller memory portion is programmed with a controller kernel running on the processing hardware. The controller kernel is programmed to: (i) selectively allow the first guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the second guest operating system in a manner that allows the second guest operating system to continue running, and (ii) selectively allow the second guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the first guest operating system in a manner that allows the first guest operating system to continue running.

[0047] According to a further aspect of the present invention, a method includes the step of: (a) providing a computer system including processing hardware, a first guest OS, a second guest OS and a controller kernel. The processing hardware defines an OS security level and at least a first additional security level above the OS security level. the method further includes the steps of: (b) selectively allowing the first guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel; (c) during step (b), pre-empting the second guest operating system in a manner that allows the second guest operating system to continue running; (d) selectively allowing the second guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel; and (e) during step (d), pre-empting the first guest operating system in a manner that allows the first guest operating system to continue running.

BRIEF DESCRIPTION OF THE DRAWINGS

[0048] The present invention will be more fully understood and appreciated by reading the following Detailed Description in conjunction with the accompanying drawings, in which:

[0049] FIG. **1** is a schematic of a prior art computer system;

[0050] FIG. **2** is a perspective external view of a first embodiment of a computer system according to the present invention;

[0051] FIG. **3** is a schematic of the first embodiment computer system;

[0052] FIG. **4** is a more detailed schematic of a portion of the first embodiment computer system;

[0053] FIGS. **5**A, **5**B, **5**C and **5**D are a flowchart of a first embodiment of a method according to the present invention;

[0054] FIG. **6** is a of a second embodiment of a computer system according to the present invention; and

[0055] FIGS. **7**A and **7**B are a flowchart of a second embodiment of a method according to the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0056] FIG. **2** shows computer system **200** according to the present invention, including desktop PC **202** and four terminals **204***a*, **204***b*, **204***c* and **204***d*. Desktop PC **202** could alternatively be any other type of computer now known or to be developed in the future, such as a laptop, a tablet, a mini computer, a mainframe computer, a super computer, a blade, etc. Terminals **204** each includes I/O devices in the form of a display, a keyboard, a mouse and an audio device. The display is the primary output device and may be any type of display now known or to be developed in the future, such as an LCD display or a CRT display. Alternatively or additionally, other output devices could be present, such as printers, lights (LEDs) and/or vibrating output devices. The keyboard, mouse and audio speakers are the primary input devices, but they may include output capabilities as well. Alternatively or additionally, there may be other output devices of any type now known or to be developed in the future, such as drawing tablets, joysticks, footpads, eyetracking input devices, touchscreens, etc.

[0057] Preferably, the display of each terminal **204** is connected to be in display data communication with desktop PC **202** by a standard parallel display connection, but may be connected by any appropriate data connection now known or to be developed in the future, such as a wireless connection. Preferably, the input devices of terminal **204** are connected to desktop PC **202** by a USB connection. Alternatively, they may be connected by any means now known or to be developed in the future, such as PS2 connection or wireless connection. One or more USB hubs may be used between desktop PC **202** and the input devices of terminals **204**.

[0058] Terminals **204** are preferably ultra thin terminals (see DEFINITIONS section). Alternatively, some or all terminals **204** could include a client computer with memory and processing capability. Terminals **204** may also include an I/O port for a portable memory, such as a USB port for a detachably attachable USB flash memory or jump drive.

[0059] FIG. **3** is a schematic of system **200** including desktop PC **202**; terminals **204**; video card **210**; I/O ports **212**; CPU **214**; POSIX kernel **215**; four guest OS's **220***a*, **220***b*, **220***c*, **220***d*; four guest applications **222***a*, **222***b*, **222***c*, **222***d*; four displays **230***a*, **230***b*, **230***c*, **230***d*; and four sets of KMA devices **232***a*, **232***b*, **232***c*, **232***d*.

[0060] Video card **210** has at least four outputs to supply display data to the four display devices **230***a*, **230***b*, **230***c*, **230***d*. Although not shown, video card **210** may have at least one additional output for: (i) additional terminals; and/or (ii) use with the POSIX kernel and/or any host operating system that may be present. The video card may take the form of multiple video cards.

[0061] The CPU may be any type of processing hardware, such as x86 architecture or other Windows type, Apple type, Sun type, etc. The hardware structure of the CPU will determine the native form for the instructions that it gives and receives. For this reason, the guest OS's **220***a*, **220***b*, **220***c*, **220***d* must be fully compatible with CPU **214**. Importantly, there is substantially no virtualizing middleware layer in desktop PC **202** to correct for any incompatibilities.

[0062] The POSIX kernel is preferably a LINUX kernel because LINUX is open source and also because a LINUX kernel can be expanded to run LINUX applications. Alternative, the kernel may be written in other formats to be compatible with the CPU such as Windows or BSD.

[0063] The PC **202** preferably includes a software algorithm (not shown) that loads the POSIX kernel (Linux 2.6 preferably) onto an available motherboard EEPROM instead of the currently installed proprietary BIOS. The kernel, along with several other helpful C based programs preferably run in 32 bit mode, as opposed to the current method of running the BIOS in 16 bit mode. These programs preferably include BusyBox, uClibc, and XII. The result is a greatly decreased boot time. All of this is preferably run in the cache memory of the CPU instead of normal DRAM. The reason for this is that DRAM is normally initialized by the BIOS and can't be used until it is initialized. The first program that runs is also written in C and it is what initializes and uses this CPU memory.

[0064] Once this is loaded, a larger module is called. This would typically be invoked from the hard drive. The POSIX kernel **215** does not necessarily have any sockets or run any applications. It may only runs sub-modules that control multiple video, keyboard, mouse, and the audio devices for multiple, concurrent local connections. Current technology will allow only one user to use the system at a time using one set of keyboard, mice, and monitors. These modules have been modified to allow multiple inputs (keyboards and mice) and outputs (audio and video) devices to be used independently and concurrently. Preferably, the terminals **204** are not remotely located, but, in some embodiments of the invention, they may be.

[0065] Preferably, the terminals are located on the same machine and the output goes directly via the system bus to the associated devices resulting in multi-user system with very little slow-down. It utilizes the excess CPU power that is available to control multiple sessions just like in a "thin client" environment. The difference is that in a "thin client" environment the output is converted to TCP-IP protocol and sent via a network connection. This conversion and packeteering of video results in slow screen redraws. This ability to run multiple "sessions" is currently available with Linux (XII) and Windows (RDP), on remote machines but the remote machines must have the necessary hardware and software necessary to locally control the keyboard, mouse, audio and video devices. Because everything is preferably loaded from the local EEPROM, boot up from power-on to login is approximately 6 seconds. This compares favorably to current Windows, MacIntosh, or Linux startup times of 30-50 seconds.

[0066] These modifications allow for a natural separation of the "sessions" to a great degree. Because of this, the invention is able to take advantage of the scheduling components and modularity of Linux to use it as a supervisor for other operating systems to run concurrently. This can efficiently install one guest operating system (for example, guest Windows OS) in conjunction with each set of keyboards, mice, and monitors.

[0067] FIGS. **7**A and **7**B are a flowchart showing exemplary process flow for the exchange of instructions between the guest OS's **220** and the CPU **214** through the POSIX kernel **215** according to the present invention. This flowchart will now be discussed in narrative terms, after which discussion, FIG. **3** will be further discussed. Using a modified Linux interrupt service code, . . . /kernel/entry-v.s, the idle loop, . . . /kernel/process.c, and a modified Interrupt Descriptor Table, this can control and tell if a system "session" is: (i) running; (ii) not running; or (iii) pre-empted. The kernel has priority

for all actions, but since it is only providing low throughput I/O control and video rendering (video is mostly handled by the GPU on the video card), preemption by the host kernel is very low in proportion to time allowed for the "clients."

[0068] Since the architecture is the same for both the host (Linux kernel) and the local "client" (x86-32 bit or 64 bit) operating system, there is little need for emulation of hardware and most instructions can be run directly on the applicable hardware. All CPU requests can be dynamically scheduled by the controller kernel and run in Ring Zero of the machine. If a protected call, privileged instruction, system trap, or page fault is presented that will not run properly or does not have permission to run in this unified system then it is moved to Ring Three. Ring Three is normally unused on an Intel system. All memory calls are directed to protected and pre-allocated memory locations. All hardware except video, ethernet, and audio devices is directly accessed by the "client" OS. Video, ethernet, and audio devices are virtualized, off-the-shelf drivers. Raw I/O from these devices is sent through the modified Linux idle loop and Interrupt Descriptor Table to the "real" hardware in a prioritized fashion. This allows a number of segregated "sessions" to be run at near native speed.

[0069] This is done without hardware virtualization extension techniques as currently available with the Intel VT or AMD V/SVM CPU chips, hardware emulation (VMWARE, QEMU, Bochs, etc.), or hypervisors like Xen or KVM (these require modification of source code). Finally, products like Cooperative Linux and UserMode Linux work with Windows as the host and Linux as the "guest" because the guest in this case (Linux) can be modified to give up control of the hardware when Windows asks for it. Since Windows can't easily be modified this concept has not been realized in reverse, for example Linux as host and Windows as guest. This aspect of the present invention is the reverse of this in that Linux is the host and Windows is the guest.

[0070] It may be difficult to modify the guest OS (for example, Windows) to give up control when the host (supervisor) asks for it, we can use /kernel/process.c (idle loop) and /kernel/entry-v.s (interrupt service) and the Interrupt Descriptor Table to trap privileged instructions and force the guest (Windows) to wait, until it is no longer preempted. In other words, we have modified the controller kernel (Linux) to put the requests of the guest (Windows) into the Linux idle loop if the guest is preempted. Since the host is not running applications, since it is only controlling I/O, the wait time during this preemption period is very short and it is not apparent to the user. Finally, when privileged instructions are trapped to Ring Three, the instructions are recompiled (sometimes on the fly) using QEMU recompilation code so that the next time this situation repeats itself, the trap is not needed.

[0071] Now that the operation of POSIX kernel has been explained in detail, discussion will return to FIG. **3**. The guest OS's **220** are preferably Windows OS's, such as Windows XP or Windows Vista. Alternatively, any type of guest OS now known or to be developed in the future may be used. In some embodiments of the invention, there will be but a single guest OS. For example, Windows Vista has been found to run faster when run through the POSIX kernel according to the present invention. In some embodiments of the invention, the guest OS's will be different from each other. For example, there may be a Windows XP OS, a Windows Vista OS, an Ubuntu LINUX OS and a BSD OS. Systems with multiple OS's may be preferred in embodiments of the present invention where there are not multiple terminals, but rather a single set of I/O devices connected to desktop PC **202** in the conventional way. In these single terminal embodiments, a single user can switch between various operating systems at will, taking advantage of native applications **222** for a variety of operating systems on a single physical machine.

[0072] FIG. **4** shows a more detailed schematic of POSIX kernel **215** including: critical portion **215***a;* non-critical portion **215***b;* interrupt descriptor table **250**; idle loop **252**; and POSIX socket **254**. Critical portion **215***a* is critical because this is the portion that passes instructions in native form between CPU **214** and guest OS's **220**. In a sense, critical portion **215***a* takes the place of the virtualizing middleware of the prior art, with the important differences that: (i) the POSIX kernel passes instructions in native form, rather than translating them into virtualized or emulated form at intermediate portions of the exchange; and/or (ii) the POSIX kernel permits the guest OS's to run at an OS security level (for example, Ring Zero or Ring One), rather than a higher security level (see FIG. **3** at reference numeral L**0**). It is noted that applications running on top of the guest OS's will run at a higher security level (see FIG. **3** at reference numeral L**3**), such as, for example, Ring Three. In other words, despite the presence of the kernel, guest OS's run at the security level that a host OS would normally run at in a conventional computer.

[0073] In this preferred embodiment of the present invention, the POSIX kernel accomplishes the exchange of native form instructions using interrupt descriptor table **250** and idle loop **252**. Interrupt descriptor table **250** receives requests for service from each of the guest OS's. At any given time it will return a positive service code to one of the guest OS's and it will return a negative service code to all the other guest OS's. The guest OS that receives back a positive return code will exchange instructions in native form with the CPU through idle loop **252**. The other guest OS's, receiving back a negative return code from interrupt descriptor table **250** will be preempted and will remain running until they get back a positive return code.

[0074] Preferably, and as shown in the flow chart of FIGS. **5**A to D, the interrupt descriptor table cycles through all the guest OS's over a cycle time period, so that each guest OS can exchange instructions with the CPU in sequence over the course of a single cycle. This is especially preferred in embodiments of the present invention having multiple terminals, so that different users at the different terminals under control of their respective guest OS's can work concurrently. Alternatively, the interrupt descriptor table could provide for other time division allocations between the various guest OS's. For example, a user could provide user input to switch between guest OS's. This form of time division allocation is preferred in single terminal, multiple operating system embodiments. There may be still other methods of time division allocation, such as random allocation (probably not preferred) or allocation based on detected activity levels at the various terminals.

[0075] Non-critical portion **215***b* shows that the controller kernel may be extended beyond the bare functionality required to control the exchange of instructions between the guest OS's and the CPU. For example, a POSIX socket may be added to allow POSIX applications to run on the kernel itself. Although the kernel is called a kernel herein, it may be extended to the point where it can be considered as a host operating system, but according to the present invention, these extensions should not interfere (that is virtualize or

emulate) instructions being exchanged through the kernel in native form between the guest OS(es) and the CPU.

[0076] FIGS. **5**A to **5**D show an embodiment of process flow for one cycle for the exchange of instructions in native form between guest OS's **220** and CPU **214** through a kernel including an interrupt descriptor table and an idle loop. The process includes: a first portion (steps S**302**, S**304**, S**306**, S**308**, S**310**, S**312**, S**314**, S**316**, S**318**); a second portion (steps S**320**, S**322**, S**324**, S**326**, S**328**, S**330**, S**332**, S**334**, S**336**); a third portion (steps S**338**, S**340**, S**342**, S**344**, S**346**, S**348**, S**350**, S**352**, S**354**); and a fourth portion (steps S**356**, S**358**, S**360**, S**362**, S**364**, S**366**, S**368**, S**370**, S**372**).

[0077] The cycle has four portions because four guest OS's (and no host OS's) are running—each portion allows the exchange of instructions between one of the four guest OS's and the CPU so that all four operating systems can run concurrently and so that multiple users can respectively use the multiple operating systems as if they had a dedicated computer instead of an ultra thin terminal.

[0078] Preferably, the entire cycle allows each OS to get a new video frame about every 30 microseconds (MS). In this way, each terminal display gets a about 30 frames per second (fps), which results in a smooth display. Above 30 frames per second, there is little, if any, improvement in the appearance of the video, but below 30 fps, the display can begin to appear choppy and/or aesthetically irritating. Because the cycle time, in this four portion embodiment is preferably about 30 MS to maintain a good 30 fps frame rate in the displays, this means that each cycle portion is about 30/4 MS, which equals about 8 MS. With current CPUs, 8 MS out of 30 MS is sufficient to handle most common applications that would be run at the various guest OS's, such as word processing, educational software, retail kiosk software, etc. As CPU's get faster over time, due to improvements such as multiple cores, it will become practical to have a greater number of guest operating systems on a single desktop computer—perhaps as many as 40 OS's or more.

[0079] FIG. **6** is a schematic of a second embodiment computer system **400** according to the present invention including: guest OS **402***a;* guest OS **402***b;* guest OS **402***c;* guest OS **402***d;* hardware control sub-modules **408**; controller kernel **410**; hard drive **414**; hardware layer; and EEPROM **418**. Hardware control sub-modules **408** include the following sub-modules: network interface card (NIC) **434**; keyboard **436**; mouse **438**; audio **440**; video **442**, memory **444** and CPU **446**. Controller kernel **410** includes the following portions: kernel process module **448**; kernel entry module **450**; idle loop **452**; interrupt service code **454**; and interrupt descriptor table **456**. Hardware layer **416** includes the following portions: network interface card (NIC) **420**; keyboard **422**; mouse **424**; audio **426**; video **428**, memory **430** and CPU **432**.

[0080] As shown by the guest OS boxes **402**, the operating systems are containerized. As shown schematically by arrow **404**, the presentation layer in this embodiment is Windows. As shown schematically by arrow **406**, there are OS containers and virtual drivers for NIC, audio and video. Additionally, there may be additional modules, such as video acceleration modules. The hardware control sub-modules **408** are direct access drivers and may additionally include other sub-modules, such as a video acceleration module. The EEPROM **418** is the normal location for BIOS, but in this embodiment of the present invention is loaded with the controller kernel **410** and X**11**. EEPROM **418** invokes the hard drive after the initial boot up. The control kernel is invoked from hard drive **414** during the original EEPROM **418** boot. At the NIC portion **420**, it is noted that each card preferably has its own MAC address and own IP address.

[0081] FIGS. **7**A and **7**B, discussed above, show a more detailed embodiment of the process flow through an interrupt descriptor table and idle loop in a LINUX controller kernel according to the present invention. Figures &A and **7**B include LINUX control kernel level steps **502**; Head **1** steps **504** and Head **2** steps **506**.

Definitions

[0082] The following definitions are provided to facilitate claim interpretation:

[0083] Present invention: means at least some embodiments of the present invention; references to various feature(s) of the "present invention" throughout this document do not mean that all claimed embodiments or methods include the referenced feature(s).

[0084] First, second, third, etc. ("ordinals"): Unless otherwise noted, ordinals only serve to distinguish or identify (e.g., various members of a group); the mere use of ordinals implies neither a consecutive numerical limit nor a serial limitation.

[0085] Receive/provide/send/input/output: unless otherwise explicitly specified, these words should not be taken to imply: (i) any particular degree of directness with respect to the relationship between their objects and subjects; and/or (ii) absence of intermediate components, actions and/or things interposed between their objects and subjects.

[0086] containerized: code portions running at least substantially independently of each other.

[0087] terminal/terminal hardware set: a set of computer peripheral hardware that includes at least one input device that can be used by a human user to input data and at least one output device that outputs data to a human user in human user readable form.

[0088] ultra thin terminal: any terminal or terminal hardware set that has substantially no memory; generally ultra thin terminals will have no more processing capability than the amount of processing capability needed to run a video display, but this is not necessarily required.

[0089] basic I/O operations: operations related to receiving input from or delivering output to a human user; basic I/O operations relate to control of I/O devices including, but not limited to keyboards, mice, visual displays and/or printers.

[0090] guest OS: a guest OS may be considered as a guest OS regardless of whether: (i) a host OS exists in the computer system; (ii) the existence or non-existence of other OS's on the system; and/or (iii) whether the guest OS is contained within one or more subsuming OS's.

[0091] security level: a level of privileges and permissions for accessing or exchanging instructions with processing hardware; for example, some types of processing hardware define security levels as Ring Zero (level of greatest permissions and privilege), Ring One, Ring Two, and so on; not all security levels may be used in a given computer system.

[0092] OS security level: any security level defined in a given system that is consistent with normal operations of a typical operating system running directly on the processing hardware (and not as a virtual machine); for example, for an Intel/Windows type of processing hardware Ring Zero, Ring One and perhaps Ring Two would be considered as "OS security levels," but Ring Three and higher would not.

[0093] native form: a form of instructions that can be operatively received by and/or is output from processing hardware

directly and without any sort of translation or modification to form by software running on the hardware; generally speaking, different processing hardware types are characterized by different native forms.

[0094] POSIX: includes, but is not limited to, LINUX.

[0095] processing hardware: typically takes the form of a central processing unit, but it is not necessarily so limited; processing hardware is not limited to any specific type and/or manufacturer (for examples, Intel/Windows, Apple, Sun, Motorola); processing hardware may include multiple cores, and different cores may or may not be allocated to different guest operating systems and/or groups of operating systems.

[0096] Computer system: any computer system without regard to: (i) whether the constituent elements of the system are located within proximity to each other; and/or (ii) whether the constituent elements are located in the same housing.

[0097] Exchange instructions: includes: (i) two way exchanges of instructions flowing in both directions between two elements; and/or (ii) one way transmission of instructions flowing in a single direction from one element to another.

[0098] Memory portion: any portion of a memory structure or structures, including, but not necessarily limited to, hard drive space, flash drive, jump drive, solid state memory, cache memory, DRAM, RAM and/or ROM; memory portions are not limited to: (i) portions with consecutive physical addresses; (ii) portions with consecutive logical address; (iii) portions located within a single piece of hardware; (iv) portions located so that the entire portion is in the same locational proximity; and/or (v) portions located entirely on a single piece of hardware (for example, in a single DRAM).

[0099] cycle: any process that returns to its beginning and then repeats itself at least once in the same sequence.

[0100] selectively allow: the selectivity may be implemented in many, various ways, such as regular cycling, user input directed, dynamically scheduled, random, etc.

[0101] pre-empt: includes, but is not limited to, delay, queue, interrupt, etc.

[0102] To the extent that the definitions provided above are consistent with ordinary, plain, and accustomed meanings (as generally shown by documents such as dictionaries and/or technical lexicons), the above definitions shall be considered supplemental in nature. To the extent that the definitions provided above are inconsistent with ordinary, plain, and accustomed meanings (as generally shown by documents such as dictionaries and/or technical lexicons), the above definitions shall control. If the definitions provided above are broader than the ordinary, plain, and accustomed meanings in some aspect, then the above definitions shall be considered to broaden the claim accordingly.

[0103] To the extent that a patentee may act as its own lexicographer under applicable law, it is hereby further directed that all words appearing in the claims section, except for the above-defined words, shall take on their ordinary, plain, and accustomed meanings (as generally shown by documents such as dictionaries and/or technical lexicons), and shall not be considered to be specially defined in this specification. In the situation where a word or term used in the claims has more than one alternative ordinary, plain and accustomed meaning, the broadest definition that is consistent with technological feasibility and not directly inconsistent with the specification shall control.

[0104] Unless otherwise explicitly provided in the claim language, steps in method steps or process claims need only be performed in the same time order as the order the steps are recited in the claim only to the extent that impossibility or extreme feasibility problems dictate that the recited step order (or portion of the recited step order) be used. This prohibition on inferring method step order merely from the order of step recitation in a claim applies even if the steps are labeled as (a), (b) and so on. This broad interpretation with respect to step order is to be used regardless of whether the alternative time ordering(s) of the claimed steps is particularly mentioned or discussed in this document.

What is claimed is:

1. A computer system comprising:

processing hardware that defines an OS security level and at least a first additional security level above the OS security level;

a first guest operating system;

a controller kernel running on the processing hardware, with the controller kernel being programmed to allow the first guest operating system exchange instructions with the processing hardware through the controller kernel at the OS security level.

2. The system of claim **1** wherein the controller kernel is programmed to allow the first guest operating system access to the processing hardware at the OS security level for basic I/O operations level.

3. The system of claim **2** further comprising a host operating system wherein:

the first guest operating system generates a plurality of first-guest-to-hardware instructions;

the processing hardware generates a plurality hardware-to-first-guest privileged instructions instructions;

the controller kernel is further programmed to selectively and temporarily pre-empt the first guest operating system by temporarily trapping at least some first-guest-to-hardware instructions and at least some hardware-to-first-guest privileged instructions;

the controller kernel is further programmed to deliver the trapped first-guest-to-hardware instructions to the processing hardware when the first guest operating system is no longer pre-empted; and

the controller kernel is further programmed to deliver the trapped hardware-to-first-guest instructions to the first guest operating system when the first guest operating system is no longer pre-empted.

4. The system of claim **3** wherein the controller kernel further comprises an interrupt descriptor table programmed to control the selective pre-emption of the first guest operating system.

5. The system of claim **4** wherein the kernel further comprises an idle loop that temporarily stores the trapped first-guest-to-hardware instructions and the trapped hardware-to-first-guest instructions during periods when the first guest operating system is pre-empted.

6. The system of claim **3** wherein the controller kernel is a POSIX kernel.

7. The system of claim **6** wherein the controller kernel is a LINUX kernel.

8. The system of claim **13** wherein the controller kernel comprises:

a modified interrupt service code;

an idle loop programmed to temporarily store the instructions that are exchanged between the first guest operating system and the processing hardware; and

a modified interrupt descriptor table.

9. The system of claim **1** further comprising a second guest operating system wherein:

the controller kernel is further programmed to allow the second guest operating system access to the processing hardware at the OS security level; and

the first guest operating system and the second guest operating system are containerized with respect to each other.

10. The system of claim **1** wherein:

the processing hardware defines a native form for instructions that the processing hardware receives from and sends to operating systems; and

the controller kernel is programmed so that the instructions communicated through the controller kernel between the first guest operating system and the processing hardware remain in the native form.

11. A computer comprising:

processing hardware that defines an OS security level and at least a first additional security level above the OS security level;

a first memory portion programmed with a first guest operating system;

a controller memory portion programmed with a controller kernel running on the processing hardware, with the controller kernel being programmed to allow the first guest operating system exchange instructions with the processing hardware through the controller kernel at the OS security level.

12. A method comprising the steps of:

providing a computer system comprising:

processing hardware that defines an OS security level and at least a first additional security level above the OS security level,

a first guest operating system, and

a controller kernel;

running the controller kernel on the processing hardware;

exchanging instructions through the controller kernel between the first guest operating system and the processing hardware at the OS security level.

13. A computer system comprising:

processing hardware that defines an OS security level and at least a first additional security level above the OS security level;

a first guest operating system;

a second guest operating system, with the first guest operating system and the second guest operating system being containerized with respect to each other; and

a controller kernel running on the processing hardware, with the controller kernel being programmed to perform cycles including at least:

a first cycle portion when the first guest operating system exchanges instructions with the processing hardware at the OS security level through the controller kernel, and

a second cycle portion when the second guest operating system exchanges instructions with the processing hardware at the OS security level through the controller kernel.

14. The system of claim **13** further comprising a third guest operating system, with the first guest operating system, the second guest operating system and the third guest operating system being containerized with respect to each other, wherein the cycles performed by the controller kernel further include at least a third cycle portion when the third guest operating system exchanges instructions with the processing hardware at the OS security level through the controller kernel.

15. The system of claim **14** further comprising a fourth guest operating system, with the first guest operating system, the second guest operating system, the third guest operating system and the fourth being containerized with respect to each other, wherein the cycles performed by the controller kernel further include at least a fourth cycle portion when the fourth guest operating system exchanges instructions with the processing hardware at the OS security level through the controller kernel.

16. The system of claim **13** wherein the OS security level is Ring Zero and/or Ring One.

17. The system of claim **13** further comprising:

a first terminal controlled by the first guest operating system; and

a second terminal controlled by the second guest operating system.

18. The system of claim **13** wherein:

the processing hardware defines a native form for instructions that the processing hardware receives from and sends to operating systems;

the controller kernel is programmed so that the instructions communicated through the controller kernel between the first guest operating system and the processing hardware remain in the native form; and

the controller kernel is programmed so that the instructions communicated through the controller kernel between the second guest operating system and the processing hardware remain in the native form.

19. The system of claim **13** wherein:

the controller kernel is programmed so that the instructions exchanged through the controller kernel between the first guest operating system and the processing hardware comprise instructions for basic I/O operations; and

the controller kernel is programmed so that the instructions exchanged through the controller kernel between the second guest operating system and the processing hardware comprise instructions for basic I/O operations.

20. The system of claim **19** wherein the controller kernel is programmed to:

during at least the first portion of the cycle, pre-empt the second guest operating system; and

during at least the second portion of the cycle, pre-empt the first guest operating system.

21. The system of claim **20** wherein the controller kernel further comprises an interrupt descriptor table programmed to control the selective pre-emption of the first guest operating system and the second guest operating system.

22. The system of claim **21** wherein the kernel further comprises an idle loop that temporarily stores instructions to and/or from the first guest operating system and the second guest operating system while they are respectively pre-empted.

23. The system of claim **19** wherein the controller kernel is a POSIX kernel.

24. The system of claim **23** wherein the first guest operating system is a Windows type operating system.

25. The system of claim **24** further comprises a POSIX application program, wherein:

the processing hardware defines a native form for instructions that the processing hardware receives from and sends to operating systems;

the instructions exchanged through the controller kernel between the first guest operating system and the processing hardware comprise native form video frame data;

the controller kernel comprises a socket programmed to run the POSIX application program;

the running of the POSIX application program generates POSIX application display data; and

the processing hardware incorporates the POSIX application display data into the native form video frame data.

26. The system of claim **25** wherein:

the controller kernel is a LINUX kernel; and

the POSIX application program is a LINUX application program.

27. The system of claim **13** wherein the controller kernel comprises:

a modified interrupt service code;

an idle loop; and

a modified interrupt descriptor table.

28. A computer comprising:

processing hardware that defines an OS security level and at least a first additional security level above the OS security level;

a first memory portion programmed with a first guest operating system;

a second memory portion programmed with a second guest operating system, with the first guest operating system and the second guest operating system being containerized with respect to each other; and

a controller memory portion programmed with a controller kernel running on the processing hardware, with the controller kernel being programmed to perform cycles including at least:

a first cycle portion when the first guest operating system exchanges instructions with the processing hardware at the OS security level through the controller kernel, and

a second cycle portion when the second guest operating system exchanges instructions with the processing hardware at the OS security level through the controller kernel.

29. A method comprising the following steps:

providing a computer system comprising:

processing hardware that defines an OS security level and at least a first additional security level above the OS security level,

a first guest operating system,

a second guest operating system, with the first guest operating system and the second guest operating system being containerized with respect to each other, and

a controller kernel;

running cycles by the controller kernel, with each cycle including the following sub-steps:

during a first cycle portion, exchanging instructions between the first guest operating system and the processing hardware at the OS security level through the controller kernel, and

during a second cycle portion, exchanging instructions between the second guest operating system and the processing hardware at the OS security level through the controller kernel.

30. A computer system comprising:

processing hardware;

a first guest operating system;

a second guest operating system, with the first guest operating system and the second guest operating system being containerized with respect to each other;

a controller kernel programmed to control the exchange of instructions between the first guest operating system and the processing hardware and the exchange of instructions between the second operating systems and the processing hardware;

a first terminal hardware set controlled by the first guest operating system, with the first terminal hardware set in the form of an ultra thin terminal; and

a second terminal hardware set controlled by the second guest operating system, with the second terminal hardware set in the form of an ultra thin terminal.

31. The system of claim **30** further comprising:

a third guest operating system, with the first guest operating system, the second guest operating system and the third guest operating system being containerized with respect to each other; and

a third terminal hardware set controlled by the third guest operating system, with the third terminal hardware set in the form of an ultra thin terminal; and

wherein the controller kernel further being programmed to control access by the third guest operating system to the processing hardware.

32. The system of claim **31** further comprising:

a fourth guest operating system, with the first guest operating system, the second guest operating system, the third guest operating system and the fourth guest operating system being containerized with respect to each other; and

a fourth terminal hardware set controlled by the fourth guest operating system, with the fourth terminal hardware set in the form of an ultra thin terminal; and

wherein the controller kernel further being programmed to control access by the fourth guest operating system to the processing hardware.

33. The system of claim **30** wherein:

the processing hardware defines a native form for instructions that the processing hardware receives from and sends to operating systems;

the controller kernel is programmed so that the instructions communicated between the first guest operating system and the processing hardware remain in the native form; and

the controller kernel is programmed so that the instructions communicated between the second guest operating system and the processing hardware remain in the native form.

34. The system of claim **30** wherein:

the first terminal hardware set comprises a first keyboard, a first mouse and a first visual display; and

the second terminal hardware set comprises a second keyboard, a second mouse and a second visual display.

35. The system of claim **30** wherein the controller kernel is a POSIX kernel.

36. The system of claim **35** wherein the first guest operating system is a Windows type operating system.

37. The system of claim **35** wherein the controller kernel is a LINUX kernel.

38. The system of claim **30** wherein the controller kernel comprises:

a modified interrupt service code;

an idle loop; and

a modified interrupt descriptor table.

39. A computer comprising:
processing hardware;
a first memory portion programmed with a first guest operating system;
a second memory portion programmed with a second guest operating system, with the first guest operating system and the second guest operating system being containerized with respect to each other;
a controller memory portion programmed with a controller kernel programmed to control the exchange of instructions between the first guest operating system and the processing hardware and the exchange of instructions between the second operating systems and the processing hardware;
a first terminal hardware set controlled by the first guest operating system, with the first terminal hardware set in the form of an ultra thin terminal; and
a second terminal hardware set controlled by the second guest operating system, with the second terminal hardware set in the form of an ultra thin terminal.

40. A method comprising the following steps:
(a) providing a computer system comprising:
processing hardware,
a first guest operating system,
a second guest operating system, with the first guest operating system and the second guest operating system being containerized with respect to each other;
a controller kernel,
a first terminal hardware set in the form of an ultra thin terminal, and
a second terminal hardware set in the form of an ultra thin terminal;
(b) controlling, by the controller kernel, an exchange of instructions between the first guest operating system and the processing hardware;
(c) controlling, by the first guest operating system, the first terminal hardware set based on the exchange of instructions occurring at step (b);
(d) controlling, by the controller kernel, an exchange of instructions between the second guest operating system and the processing hardware; and
(e) controlling, by the second guest operating system, the second terminal hardware set based on the exchange of instructions occurring at step (d).

41. A computer system comprising:
processing hardware that defines an OS security level and at least a first additional security level above the OS security level;
a first guest operating system;
a second guest operating system;
a controller kernel running on the processing hardware, with the controller kernel being programmed to:
selectively allow the first guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the second guest operating system in a manner that allows the second guest operating system to continue running, and
selectively allow the second guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the first guest operating system in a manner that allows the first guest operating system to continue running.

42. The system of claim **41** wherein the controller kernel being programmed to:
dynamically schedule the first guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the second guest operating system in a manner that allows the second guest operating system to continue running, and
dynamically schedule the second guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the first guest operating system in a manner that allows the first guest operating system to continue running.

43. The system of claim **41** wherein the controller kernel being programmed to:
selectively allow based on user input the first guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the second guest operating system in a manner that allows the second guest operating system to continue running, and
selectively allow based on user input the second guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the first guest operating system in a manner that allows the first guest operating system to continue running.

44. The system of claim **41** wherein the controller kernel comprises:
a modified interrupt service code;
an idle loop; and
a modified interrupt descriptor table.

45. A computer comprising:
processing hardware that defines an OS security level and at least a first additional security level above the OS security level;
a first memory portion programmed with a first guest operating system;
a second memory portion programmed with a second guest operating system;
a controller memory portion programmed with a controller kernel running on the processing hardware, with the controller kernel being programmed to:
selectively allow the first guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the second guest operating system in a manner that allows the second guest operating system to continue running, and
selectively allow the second guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel while pre-empting the first guest operating system in a manner that allows the first guest operating system to continue running.

46. A method comprising the steps of:
(a) providing a computer system comprising:
processing hardware that defines an OS security level and at least a first additional security level above the OS security level,

a first guest operating system,
a second guest operating system, and
a controller kernel running on the processing hardware;

(b) selectively allowing the first guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel;

(c) during step (b), pre-empting the second guest operating system in a manner that allows the second guest operating system to continue running;

(d) selectively allowing the second guest operating system to have access to the processing hardware at the OS security level under control of the controller kernel; and

(e) during step (d), pre-empting the first guest operating system in a manner that allows the first guest operating system to continue running.

* * * * *